Oracle® US Federal Human Resources

Implementation Guide

Release 11*i*

Part No. B15542-01

November 2004

ORACLE®

Oracle US Federal Human Resources Implementation Guide, Release 11*i*

Part No. B15542-01

Primary Author: Louise Raffo

Contributing Author: Juliette Fleming, John Cafolla, Janet McCandless

Contents

Index

Send Us Your Comments

Oracle US Federal Human Resources Implementation Guide, Release 11*i*

Part No. B15542-01

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 US Federal HR Documentation Manager
- Postal service:
  Oracle US Federal HR 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 11*i* of the *Oracle US Federal Human Resources Implementation Guide*.

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

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

  If you have never used Oracle HRMS, Oracle suggests you attend one or more of the Oracle HRMS training classes available through Oracle University
- Oracle Self-Service Web Applications.

  To learn more about Oracle Self-Service Web Applications, read the *Oracle Self-Service Web Applications Implementation Manual*.
- The Oracle Applications graphical user interface.

  To learn more about the Oracle Applications graphical user interface, read the *Oracle Applications User's Guide*.

See Related Documents for more information about Oracle Applications product information.

See Related Documents on page xii 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

Oracle HRMS shares business and setup information with other Oracle Applications products. Therefore, you may want to refer to other user guides when you set up and use Oracle HRMS.

You can read the guides online by choosing Library from the expandable menu on your HTML help window, by reading from the Oracle Applications Document 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). 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.

Guides Related to This Product

OA Personalization Framework and OA Extensibility Framework

Learn about the capabilities of the 5.6 Framework technologies.

Oracle Human Resources Management Systems Enterprise and Workforce Management Guide

Learn how to use Oracle HRMS to represent your agency. This includes setting up your organization hierarchy, recording details about jobs and positions within your agency, defining person types to represent your workforce, and also how to manage your budgets and costs.

Oracle Human Resources Management Systems Workforce Sourcing, Deployment, and Talent Management Guide

Learn how to use Oracle HRMS to represent your workforce. This includes recruiting new workers, recording and managing the workforce using for example by processing Request for Personnel Actions and mass actions, and reporting on your workforce.

Oracle Human Resources Management Systems Compensation and Benefits Management Guide

Learn how to use Oracle HRMS to manage compensation. For example, read how to process compensation and awards, set up automated step increases, and federal benefits such as Federal Health Employee Benefits and Thrift Savings Plans. You can also learn about setting up earnings and deductions for payroll processing, managing leave and absences, and reporting on compensation.

Oracle Human Resources Management Systems Configuring, Reporting, and System Administration in Oracle HRMS

Learn about extending and configuring Oracle HRMS, managing security, auditing, information access, and letter generation.

Oracle Human Resources Management Systems Implementation Guide

Learn about the setup procedures you need to carry out in order to successfully implement Oracle HRMS in your enterprise.

Oracle Human Resources Management Systems FastFormula User Guide

Learn about the different uses of Oracle FastFormula, and understand the rules and techniques you should employ when defining and amending formulas for use with Oracle applications.

Oracle Human Resources Management Systems Deploy Self-Service Capability Guide

Set up and use self-service human resources (SSHR) functions for managers, HR Professionals, and employees.

Oracle Human Resources Management Systems Deploy Strategic Reporting (HRMSi)

Implement and administer Oracle Human Resources Management Systems Intelligence (HRMSi) in your environment.

Oracle Human Resources Management Systems Strategic Reporting (HRMSi) User Guide

Learn about the workforce intelligence reports included in the HRMSi product, including Daily Business Intelligence reports, Discoverer workbooks, and Performance Management Framework reports.

Implementing Oracle Approvals Management

Use Oracle Approvals Management (AME) to define the approval rules that determine the approval processes for Oracle applications. Download this guide from Oracle *MetaLink*, Note: 282529.1.

Oracle iRecruitment Implementation Guide

Set up Oracle *i*Recruitment to manage all of your enterprise's recruitment needs.

Oracle Learning Management User Guide

Set up and use Oracle Learning Management to accomplish your online and offline learning goals.

Oracle Learning Management Implementation Guide

Implement Oracle Learning Management to accommodate your specific business practices.

Oracle Time and Labor Implementation and User Guide

Learn how to capture work patterns such as shift hours so that this information can be used by other applications such as 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 an installation of Oracle Applications. This guide also introduces the concepts behind Applications-wide features such as Business Intelligence (BIS), languages and character sets, and Self-Service Web 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 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.

"About" Document

For information about implementation and user document, instructions for applying patches, new and changes setup steps, and descriptions of software updates, refer to the "About" document for your product. "About" documents are available on Oracle*MetaLink* for most products starting with Release 11.5.8.

Maintaining Oracle Applications

Use this guide to help you run the various AD utilities, such as AutoUpgrade, AutoPatch, 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.

Oracle Applications Developer's Guide

This guide contains the coding standards followed by the Oracle Applications development staff and describes the Oracle Application Object Library components that are needed to implement the Oracle Applications user interface described in the *Oracle Applications User Interface Standards for Forms-Based Products*. This manual also provides information to help you build your custom Oracle Forms Developer forms so that the forms 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 Oracle Applications products and how to apply this UI to the design of an application built by using Oracle Forms.

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.

Oracle Workflow Administrator's Guide

This guide explains how to complete the setup steps necessary for any Oracle Applications 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 Oracle Applications users can view and respond to workflow notifications and monitor the progress of their workflow processes.

Oracle Workflow API Reference

This guide describes the APIs provided for developers and administrators to access Oracle Workflow.

Oracle Applications Flexfields Guide

This guide provides flexfields planning, setup, and reference information for the Oracle HRMS 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 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 Manual

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

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

Introduction

Planning Implementation

The flexibility of Oracle HRMS enables you to develop an implementation project plan that meets your own specific business needs for Oracle Human Resources, Oracle Payroll, Oracle Advanced Benefits, Oracle Learning Management, and Oracle Self-Service Human Resources (SSHR).

With Oracle HRMS you choose the functions you want to implement initially. You implement other functions when you need to use them.

For example, you might decide to implement for HR users, and then to add payroll processing capabilities in a subsequent phase. Alternatively, you might decide to implement payroll functions during your initial phase. You could choose to extend your range of HR information and functions later.

Decision making is an important part of any implementation process and before you begin to configure Oracle HRMS you must decide how you want to use the system.

Adopting a staged, or *incremental*, approach to implementation lets you focus on those areas of the system you want to use.

Working in partnership with Oracle you can call on skilled consultants to provide you with all of the training, and technical and professional expertise you need. Together you can successfully implement an HRMS system that matches your specific business needs in the most efficient and cost-effective manner.

HRMS Configuration Workbench

You can manage your implementation using the HRMS Configuration Workbench. The Workbench delivers a configuration interview that helps you make the best configuration choices for your enterprise. The interview is based on the distilled knowledge of good practice from the experience of hundreds of customers working in different industries and geographies.

Use the Quick Start option in the Workbench to generate prototype configurations that include all the essential definitions for using HR and Payroll. You can create alternative prototypes by answering the interview questions slightly differently. The Quick Start option generates default settings for other required system components in HR and Payroll that you would typically set up manually. Work through your business processes using the prototypes to experience the effect of alternative configuration choices.

When you are satisfied with your prototype configuration you can proceed to use the full configuration management toolset in the Workbench. For the Full Implementation, you should make sure you complete all the details about your enterprise before you generate

the full configuration. You can load the full configuration only once, so you need to include all the required detail before you load it.

When you have loaded the full configuration, you have a basic implementation that matches the business processes of your enterprise. Evaluate what additional features you require, and follow the implementation steps to add features and extend your configuration.

Read more about the Configuration Workbench in the Getting Started guide on MetaLink, Note 281421.1.

2

Implementation Guide

Implementation Steps

Before You Start

Before you begin implementing Oracle HRMS, you must ensure your legislation-specific startup data is installed. The installation is normally done by the MIS Manager. You need this startup data before you use Elements, Payment Methods or Legislation Specific Flexfield Structures.

See *Installing Oracle Applications* for more information.

Also, check to see whether there are any post installation steps you need to perform before you start to implement Oracle HRMS.

See: Post Install Steps, page 2-1.

Post Install Steps

There are two generic post install utilities for Oracle HRMS in Release 11*i*:

- DataInstall enables you to specify all the legislations that you want to install for HR and Payroll, and HR only. This means that when you subsequently perform an installation or upgrade, you can install your legislations in a single operation. DataInstall provides a series of menus from which you can specify the legislation and product combinations.
- AutoPatch (adpatch) applies the installation or upgrade combinations that you have previously specified in DataInstall.

Canada and USA

If you are installing Oracle Payroll (Canada and US) you also need to install Quantum, a third party taxation product, produced by Vertex, that Oracle Payroll (Canada and US) uses.

France

If you are installing a French localization, there are two additional post install steps for that must be completed for Oracle HR for France. These are:

- Run the Seed French Data process
- Create a new EUL (End User Layer) in Discoverer and enable user access to database tables and views by running the Grant HR Access to Discoverer process

If you are installing the US Federal HR localization, there is one additional step to be able to produce bitmap reports.

To Run the DataInstall Utility (Required):

To specify legislations using DataInstall:

1. Run the DataInstall utility to select legislations using the command:

   ```
   jre oracle.apps.per.DataInstall <APPS Username> <APPS password>
   ```

   > **Note:** In multiple sets of book installs, supply the username and password of the first APPS account.

   The DataInstall Main Menu is displayed.

2. Choose option 1. This displays a screen showing a list of product localization combinations that you can choose.

   For each product or localization that already has legislation data on the database, the Action will be defaulted to upgrade. This cannot be changed.

   If the Legislation/Verticalization is Federal HR only

   If you are upgrading Oracle Federal HR, choose both Oracle Federal HR and Oracle Human Resources from the list of product localizations.

3. Select any new installations that you want to implement. For example, if you wanted to install Canada Payroll, number 3, you would type 3I. This would also set the action on Canada Human Resources to Install as dependencies are maintained.

   If you are installing an additional legislation, to correct a mistake use the Clear option. If you have selected to install an additional Payroll and HR legislation, clearing the Payroll legislation will clear the HR legislation also.

   You cannot use Force Install for upgrades. You only need to use Force Install if you want to reapply steps in the Global Legislation Driver that have already been applied.

4. If you select a localization other than US or GB, you are returned to the main menu.

   If you select a US or GB localization the DataInstall - College Data Option screen is displayed showing whether college data is currently installed for US and GB localizations. The install option is only available if you have no existing college data. If you have existing data then the localization will default to Upgrade, though this can be changed.

   Choose Remain if you want to keep the existing data and not apply the upgrade, or choose Clear to set the action to null.

   You cannot use Force Install at this point.

   Press Return to display the main menu and make further changes or exit.

5. If you are installing a US or Canadian localization and you have installed Oracle Payroll, select the JIT/Geocodes option from the DataInstall menu to load the latest JIT/Geocodes data.

   This option is also available to Oracle HR customers who wish to validate North American addresses using Vertex Geocodes data and/or maintain employee tax data

in Oracle HR. However, customers who do not have Oracle Payroll *must* obtain a license from Vertex before installing this data.

Press Return to display the main menu and make further changes or exit.

6. When you choose to exit the DataInstall Actions Confirmation screen is displayed.

   Select Y to save your changes and exit, or select N to exit without saving your changes.

   When you have exited, the DataInstall Actions Summary screen is displayed. This summarizes the actions that will be taken when the program exits, or when ADPATCH is run with the Global Legislation driver.

Run the Global Legislation Driver using AutoPatch (adpatch) (Required):

1. The Generic HR Post Install Driver delivers the generic entity horizon and all the selected localizations. To run it, type in the following commands:

   ```
   $ cd $PER_TOP/patch/115/driver

   $ adpatch
   ```

   Then apply the driver hrglobal.drv

2. After applying the Global Legislation Driver, examine the out file hrlegend.lst. This logs any localizations selected in the DataInstall utility but which have not been applied by this driver. Refer to the Installation Manual to ensure that everything has been applied correctly, or contact World-wide Support.

   If the Legislation is UK

3. Examine the following out files:

   - pegbutcl.lst. This file logs the step that removes previously seeded user tables for the UK legislation before delivering the latest version. It may also show where seed data names have been changed between releases.
   - perleggb.lst. This file logs the housekeeping step that gets rid of redundant UK seed data after delivery of the latest version. It also records the new balance feeds that have been inserted following an upgrade from Oracle Human Resources to Oracle HRMS.
   - The log file produced by the FFXBCP formula compilation step. The name of the FFXBCP log follows the naming convention of the <request_id> log, and is included in the last section of the adpatch log.

   These files are used by Oracle Support Services to diagnose problems with seed data following an upgrade. SQL errors indicate severe problems. Keep these files for reference in the event of any future problems with UK seed data.

Install Quantum for Oracle Payroll (Canada and US) (Conditionally Required):

1. Set up a directory structure to hold the Quantum product.

   By default, Oracle Payroll looks for the Quantum product in the $PAY_TOP/vendor/quantum directory, however, you can choose where it is placed and override the default location.

> **Tip:** You could create a $PAY_TOP/vendor/quantum_versions directory and a $PAY_TOP/vendor/quantum symbolic link pointing to the correct version of Quantum, since the Quantum products release cycle may be different from Oracle Payroll.

2. Unpack the Quantum Components from the CD.

   Oracle Applications provide a CD on which will be a ZIP file called pyvendor.zip in a directory called pay. On the ZIP file will be one directory per operating system that is supported by Oracle Payroll (US). Uncompress the pyvendor.zip file and move the required version into the directory structure created in Step 1. For example, uncompress the file then do the following:

   ```
   $ mv SOLARIS/2.2.4 $PAY_TOP/vendor/quantum_versions

   $ ln -s $PAY_TOP/vendor/quantum_versions/2.2.4 $PAY_TOP/vendor/qu
   antum
   ```

   The extraction from the compressed file will create a directory called (<operating system>/2.2.4) and two sub directories (lib and utils) along with a number of files in each directory. One of the files created is devenv, this devenv file is the same as the $FND_TOP/usrxit/devenv file except that some of the lines are uncommented. The uncommented lines relate to instructions on how the Oracle Payroll process PYUGEN should be linked. The lines that are uncommented are:

   ```
   VND_VERTEX='$(PAY_TOP)/vendor/quantum'

   VND_LINK='$(VND_VERTEX)/lib/libvprt.a \

             $(VND_VERTEX)/lib/libqutil.a \

             $(VND_VERTEX)/libloc.a \

             $(VND_VERTEX)/lib/libcb63.a'

   $ ln -s $PAY_TOP/vendor/quantum_versions/2.2.4 $PAY_TOP/vendor/qu
   antum

   VNDPAYSL='$(PAY_TOP)/lib/py3c.o $(PAY_TOP)/lib/py3v.o $(VND_LINK)
   '

   VNDPAYPL='$(PAY_TOP)/lib/py3c.o $(PAY_TOP)/lib/py3v.o $(VND_LINK)
   '

   export VND_VERTEX VND_LINK VNDPAYPL VNDPAYSL
   ```

   > **Note:** Some of these settings relate to the location of the Quantum product, thus if the Quantum product is not in $PAY_TOP/vendor/quantum this file needs to be edited.

   If you have made any changes to your $FND_TOP/usrxit/devenv file, you must merge these differences into the file. If you have not already made any changes then you can simply copy 2.2.4/devenv to $FND_TOP_usrxit/devenv.

3. Relink the Oracle Payroll executable PYUGEN using adrelink.

```
$ adrelink force=y ranlib=y "pay PYUGEN"
```

Ensure that the adrelink completed successfully by checking the log file.

4. Build the Quantum product's data files.

   To build Quantum's data files, firstly create a directory to hold the data files. Oracle Payroll assumes that these data files are in $PAY_TOP/vendor/quantum/data.

   Secondly, run the utility dbcreate that is in the Quantum utils directory. This utility will show a menu of either Payroll or Geocoder. Choose the Payroll option and at the prompt "Enter the Payroll datasource name:" enter the directory into which the data files are to be placed, for example, /apps/pay/11.5/vendor/quantum/data. Once the processing is complete, the menu will reappear and the utility can be exited.

   > **Note:** Ensure that the file permissions of the data files are set to readable for all the relevant users. If this is not done then Oracle Payroll will not be able to access these files.

5. Populate the Quantum data files.

   Once the data files have been created they need to be populated with taxation data. The taxation data is held in a file called qfpt.dat, which will be delivered in the pyvendor.zip file. Copy this file into the Quantum product area. Once this has been done the data file update utility can be run. This is located in the utils directory called vprtmupd. Select the Update Payroll Tax option from the menu, and answer the displayed questions. The first prompts for the datasource, this should be the location of the data files created in the previous step. The second is the location of the qfpt.dat file. For example:

   ```
   Enter Datasource: /apps/[ay/11.5/vendor/quantum/data

   Enter the path of the update file:  /apps/pay/11.5/vendor/quantum
   ```

   > **Note:** The update file supplied is a default file, it is not guaranteed to calculate taxes correctly. Its purpose is to allow you to perform testing prior to contacting Vertex to request the correct update file.

6. Register the Quantum Data Files location.

   If the data files for Quantum have not been placed in the default location ($PAY_TOP/vendor/quantum/data), then the location of these files must be supplied to Oracle Payroll. This is performed by placing a row in the PAY_ACTION_PARAMETERS table:

   ```
   SQL> insert into pay_action_parameters

   2 values ('TAX_DATA', '/apps/quantum/data');
   ```

Run the Seed French Data process :

1. This process creates and populates some of the user defined tables used by the various French reports for the Business Group of the current responsibility. It also delivers the example data for the Bilan Social. It should be run for each Business Group that contains data for the French legislation.

For information on the user defined tables created by this process see: User Defined Tables Used by the Bilan Social, *Oracle HRMS Enterprise and Workforce Management Guide* , and User Defined Tables, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Run the Seed French Data process in the Submit Requests window.

Create an EUL and Enable User Access to Database Tables and Views by Running the Grant HR Access to Discoverer process (France):

In order to use the supplied business area and Discoverer workbooks you must perform the following steps:

1. Create an EUL

   If you do not have an existing Oracle Discoverer EUL you must create one before you can import the HR France - Bilan Social business area.

   See: *Oracle Discoverer Administration Guide* for further information on creating an EUL.

2. Import the hrfrbsel.eex file

   Once you have a suitable EUL you must import the hrfrbsel.eex file. This will deliver the HR France - Bilan Social business area. This file is contained in the Bilan Social Discoverer Components zip file that can be obtained from Oracle World Wide Support.

   See: *Oracle Discoverer Administration Guide* for further information on importing files.

3. Run the Grant HR Access to Discoverer process

   The EUL user must be given the correct permissions in order to access the tables and views in the database that are used by the Bilan Social. To do this, you must run Grant HR Access to Discoverer process in the Submit Requests window.

   You will now be prompted to enter the following parameters:

   - the connect string for the database on which the Bilan Social data is stored
   - your EUL user name
   - your EUL password

   Choose Submit. The process will now run and assign the appropriate permissions to your EUL user.

US Federal HR Print Reports:

To be able to produce bitmap and postscript reports, you must relink ar60runb.

1. Chenv to the environment.
2. Make sure that FND_TOP and APPL_TOP are correct.
3. cd $FND_TOP/bin
4. adrelink.sh force=y "fnd ar60runb"

Implementation Checklist

Use the following checklists to record which parts of Oracle HRMS you want to use. Then refer to the implementation flowcharts to see the high level steps you must complete for each business function you have chosen to implement.

❒ Post Install Steps, page 2-1 (Required)

Refer to the Post Install Steps to see any steps you must perform before you implement Oracle HRMS.

❒ Administration, page 2-24 (Required)

Includes key and descriptive flexfields, Extra Information Types (EITs), currencies, "View All" HRMS User, and lookups.

❒ Enterprise and Workforce Management, page 2-7 (Required)

Includes organizations, jobs, positions, budgets, person types, collective agreements, complaint tracking, and government reporting.

❒ Compensation and Benefits, page 2-34 (Optional)

Includes grades and their relationship to pay, mass salary actions, compensation and awards, benefits eligibility, leave and absence management, and element sets.

❒ Workforce Sourcing and Deployment, page 2-48 (Required)

Includes person types, assignment statuses and special personal information.

❒ Talent Management, page 2-52 (Optional)

Includes recruitment, career management, evaluation and appraisals and succession planning.

❒ Workforce Intelligence, page 2-56 (Optional)

Includes predefined Discoverer workbooks and a predefined Discoverer End User Layer based on HRMS transactional tables.

❒ HR Information Systems, page 2-61 (Optional)

Includes reports, letter generation, configuration, task flows, user security, US Federal workflow and maintenance forms, audit requirements and Oracle Applications Help, and Web Applications Desktop Integrator (Web ADI). Includes setting the frequency for US federal processes and reports.

Administration

The administration steps are usually performed by the System Administrator. Sign on to the system using your System Administrator username and password. Contact your DBA if you do not know this information.

Define Key Flexfields

There are six Key Flexfield Structures you must define before you can define a Business Group in Oracle HRMS. These are:

- Job
- Position
- Grade

- People Group
- Cost Allocation
- Competence

You can also define the Collective Agreement Grades flexfield at this time, or you can do it after defining your Business Group.

The application comes with predefined information. The entire flexfield information for Grade is predefined as are the value sets for the Job and Position key flexfield segments. You can define additional segments of Job and Position, as well as those in People Group and Cost Allocation based on your agency's requirements.

Before you begin your implementation of these key flexfields you must clearly specify your requirements. This specification must include the following details for each key flexfield:

- The Structure Name and the number of Segments
- The Flexfield Segment Names, Order, Validation Options and Qualifiers
- The Flexfield Value Sets to be used and any lists of values

After you have completed the definition of a key flexfield, you need to run the Create Key Flexfield Database Items process concurrent process to generate Database Items for the individual segments of the Flexfield. This applies to your Job, Position, Grade , Competence, and People Group Key Flexfields only.

> **Important:** If you used the Configuration Workbench, you have already defined the structures for your Job, Position, and Grade key flexfields. You may want to add more validation, such as cross-validation. The Workbench created default structures for the other flexfields associated with a business group (People Group, Cost Allocation, and Competence). If you plan to use these flexfields in your implementation, you must update the default structures to display the segments you require.

Define Job Flexfield

After you have specified your requirements for recording and reporting Job information, follow this implementation sequence:

Step 1: Define Job Flexfield Value Sets

To validate the values which a user can enter for any segment, you must define a specific Value Set. A predefined value set for Occupational Series is provided, GHR_US_OCC_SERIES.

The attributes of the Value Set control the type of values that can be entered, and how many characters each segment can hold. The attributes of the Value Set will also control how the values are to be validated.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Sets window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 2: Define Job Flexfield Segments

Define the first segment of the Job key flexfield the Occupational Series using the supplied value set, GHR_US_OCC_SERIES. Define the remaining segments that you want to use for your Business Group

Note the Occupational Series segment number. You use this information later when you enter the US Federal Org Information for your business groups.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to create new job name combinations in the Job window.

> **Note:** You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

Use the Key Flexfield Segments window.

See: Setting up the Job Key Flexfield, *Configuring, Reporting, and System Administration Guide*, Oracle US Federal Human Resource Key Flexfields, *Configuring, Reporting, and System Administration Guide*

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 3: Define Job Flexfield Segment Values

If you have chosen Independent or Dependent validation for a Value Set used by a Job Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Segment Values window

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 4: Define Job Flexfield Cross Validation Rules

Define any Cross Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 5: Define Job Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Shorthand Aliases window

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 6: Freeze and Compile Your Job Flexfield Structure

You are now ready to freeze your Job Flexfield definition. Navigate to the Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. The application now freezes and compiles your Job Flexfield definition. Compiling the flexfield definition enables the Job Flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 7: Run Create Key Flexfield Database Items Process

If you want to make use of the individual segments of the flexfield as separate Database Items you can run this concurrent process from the Submit a New Request window. The only parameter associated with this process is the Key Flexfield Name.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Define Position Flexfield

After you have specified your requirements for recording and reporting Position information in your enterprise, follow this implementation sequence:

Step 8: Define Position Flexfield Value Sets

To validate the values which a user can enter for any segment, you must define a specific Value Set.

The following value sets have been predefined for the required US Federal HR segments for position:

- Position Title (GHR_US_POSITION_TITLE)
- Position Description Number (GHR_US_POS_DESC_NUM)
- Sequence Number (GHR_US_SEQUENCE_NUM)
- Agency/Subelement Code (GHR_US_AGENCY_CODE).

The attributes of the Value Set control the type of values that you can enter, and how many characters each segment can hold. The attributes of the Value Set also control how the application validates the values.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 9: Define Position Flexfield Segments

At a minimum, you must define the following four required segments using the supplied value sets.

- Use one of the first five segments (Segment 1, 2, 3, 4, or 5) for Position Title
- Use the remaining segments for Position Description Number, Sequence Number, and Agency/Subelement Code,

You can define up to 30 segments within the structure. For the segments that you add, you can define a list of valid codes or values.

Note the segment numbers for Position Title, Position Description Number, Sequence Number, and Agency/Subelement Code. You use this information later when you enter the US Federal Org Information for your business groups.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to create new position name combinations in the Position window.

Note: You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

Use the Key Flexfield Segments window

See: Setting up the Position Key Flexfield, *Configuring, Reporting, and System Administration Guide*, Oracle US Federal Human Resource Key Flexfields, *Configuring, Reporting, and System Administration Guide*

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 10: Define Position Flexfield Segment Values

If you have chosen Independent or Dependent validation for a Value Set used by a Position Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Define Segment Values window

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 11: Define Position Flexfield Cross Validation Rules

Define any Cross Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 12: Define Position Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Shorthand Aliases window

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 13: Freeze and Compile Your Position Flexfield Structure

You are now ready to freeze your Position Flexfield definition. Navigate to the Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. The application now freezes and compiles your Position Flexfield definition. Compiling the flexfield definition enables the Position Flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 14: Run Create Key Flexfield Database Items process

If you want to make use of the individual segments of the flexfield as separate Database Items you can run this concurrent process from the Submit a New Request window. The only parameter associated with this process is the Key Flexfield Name.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Define Grade Flexfield

The Grade key flexfield structure is predefined for you upon installation and is not extensible. The structure contains two segments:

- define the pay plan (GHR_US_PAY_PLAN) is the *first* segment
- define the grade (GHR_US_GRADE_OR_LEVEL) is the *second* segment

You only need to freeze and compile the predefined structure. Later on during implementation you associate the US Federal Grade flexfield with the Business Group you set up.

To view the structure, use the Key Flexfield Segments window.

See: Oracle US Federal Human Resource Key Flexfields, *Configuring, Reporting, and System Administration Guide*

Step 15: Freeze and Compile the Grade Flexfield Structure

Navigate to the Key Flexfield Segments window. Query the US Federal Grade Key Flexfield. (US Federal Grade Flexfield is the code name; US Government Grade Flexfield is the View name). Enter Yes in the Freeze Flexfield Definition field. The application now freezes and compiles your Grade Group Flexfield definition.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 16: Run Create Key Flexfield Database Items process

If you want to make use of the individual segments of the flexfield as separate Database Items you can run this concurrent process from the Submit a New Request window. The only parameter associated with this process is the Key Flexfield Name.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Define People Group Flexfield

People Group information is associated with employee and contingent worker assignments and is used to identify special groups of employees in your enterprise, such as members of a union.

> **Note:** You **must** define at least one segment for the People Group Key Flexfield.
>
> If you do not, you will not be able to use the Assignment window for employees, applicants, or contingent workers.

After you have specified your requirements to take best advantage of the flexibility of the application for recording and reporting People Group information in your enterprise, the implementation sequence you follow is:

Step 17: Define People Group Flexfield Value Sets

To validate the values which a user can enter for any segment, you must define a specific Value Set.

The attributes of the Value Set control the type of values that can be entered, and how many characters each segment can hold. The attributes of the Value Set will also control how the values are to be validated.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 18: Define People Group Flexfield Segments

Define a structure for your People Group Flexfield which contains the segments you want to use for your Business Group. You will use this structure to enter People Group details in the Assignment window.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to enter People Group information in the Assignment window.

> **Note:** You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

See: Oracle US Federal Human Resource Key Flexfields, *Configuring, Reporting, and System Administration Guide*

Step 19: Define People Group Flexfield Segment Values

If you have chosen Independent or Dependent validation for a Value Set used by a People Group Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Define Segment Values window

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 20: Define People Group Flexfield Cross Validation Rules

Define any Cross Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 21: Define People Group Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Shorthand Aliases window

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 22: Freeze and Compile Your People Group Flexfield Structure

You are now ready to freeze your People Group Flexfield definition. Navigate to the Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. The application now freezes and compiles your People Group Flexfield definition. Compiling the flexfield definition enables the People Group Flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 23: Run Create Key Flexfield Database Items process

If you want to make use of the individual segments of the flexfield as separate Database Items you can run this concurrent process from the Submit a New Request window. The only parameter associated with this process is the Key Flexfield Name.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Define Cost Allocation Flexfield

Cost Allocation information normally records the details of employee costing associated with payroll results. If you have installed Oracle Payroll, you can accumulate the costs associated with your payroll results and transfer these to your General Ledger system. If you have not installed Oracle Payroll you can use the costing flexfield to enter your cost allocation information.

After you have specified your requirements for recording and reporting costing information, follow this implementation sequence:

> **Warning:** You **must** define at least one segment for the Cost Allocation Key Flexfield. If you do not, you will experience problems using windows with the flexfield window.

Step 24: Define Cost Allocation Flexfield Value Sets

To validate the values which a user can enter for any segment, you must define a specific Value Set.

The attributes of the Value Set control the type of values that can be entered, and how many characters each segment can hold. The attributes of the Value Set will also control how the values are to be validated.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 25: Define Cost Allocation Flexfield Segments and Qualifiers

Define a structure for your Cost Allocation Flexfield which contains the segments you want to use for your Business Group. You will use this structure to enter your payroll costing details in Oracle HRMS.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to enter Costing details anywhere on the system.

Note: You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

The Cost Allocation Flexfield is the only key flexfield in Oracle HRMS that makes use of Qualifiers. You use Segment Qualifiers to control the level at which costing information can be entered to the system. Each Qualifier determines the level at which costing information can be entered. The following table illustrates the six possible choices for each segment:

Qualifier	Effect on window
Payroll	Enter segment values in the *Payroll* window.
Link	Enter segment values in the *Element Link* window.
Balancing	Enter balancing segment values in the *Element Link* window.
Organization	Enter segment values in the *Costing Information* window for the Organization.
Assignment	Enter segment values in the *Costing* window for the assignment.
Entry	Enter segment values in the *Element Entries* window.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 26: Define Cost Allocation Flexfield Segment Values

If you have chosen Independent or Dependent validation for a Value Set used by a Cost Allocation Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Define Segment Values window.

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 27: Define Cost Allocation Flexfield Cross Validation Rules

Define any Cross Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 28: Define Cost Allocation Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Shorthand Aliases window

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 29: Freeze and Compile Your Cost Allocation Flexfield Structure

You are now ready to freeze your Cost Allocation Flexfield definition. Navigate to the Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. The application now freezes and compiles your Cost Allocation Flexfield definition. Compiling the flexfield definition enables the Cost Allocation Flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Define Competence Key Flexfield

The Competence Key Flexfield is used to record information about a multi-level competencies. This enables you to record more details about a competence.

After you have specified your requirements to take best advantage of the flexibility of Oracle Human Resource Management Systems for recording and reporting competence information in your enterprise, the implementation sequence which you follow is:

Step 30: Define Competence Flexfield Value Sets

To validate the values that a user can enter for any segment, you must define a specific Value Set.

The attributes of the Value Set will control the type of values that can be entered, and how many characters each segment can hold. The attributes of the Value Set will also control how the values are to be validated.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 31: Define Competence Flexfield Segments

Define a structure for your Competence Flexfield that contains the segments you want to use. You will use this structure to enter your competence details in the Competence window.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to enter new details in the Competence window.

You must specify one of the segments as the Default Attribute using the flexfield qualifier. You must also attach the Others flexfield qualifier to all other segments in the structure.

If you intend to upload SkillScape competencies you should try to ensure that you set up segment 1 to record the competence name as this is the segment into which the competence name is automatically uploaded. If you define another segment to hold the competence name you must alter the file $PER_TOP/patch/115/sql/peducomp.sql so that the reference to segment1 is changed to the segment in which you hold the name.

Note: You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

Use the Key Flexfield Segments window.

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 32: Define Competence Flexfield Segments Values

If you have chosen Independent or Dependent validation for a Value Set used by a Competence Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Segment Values window.

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 33: Define Competence Flexfield Cross-Validation Rules

Define any Cross-Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window.

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 34: Define Competence Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Define Shorthand Aliases window.

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 35: Freeze and Compile Your Competence Flexfield Structure

You are now ready to freeze your Competence Flexfield definition. Navigate to the Define Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. The application now freezes and compiles your Competence Flexfield definition. Compiling the flexfield definition enables the flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window.

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Define Collective Agreement Grades Key Flexfield

The Collective Agreement Grades Key Flexfield records information about how an employee is graded or ranked in a collective agreement. The Collective Agreement Grades Key Flexfield enables you to specify any number of structures. Each grade structure is defined as a separate structure of the flexfield. You then link a specific structure to a collective agreement in the Agreement Grades window.

Note: Oracle US Federal HRMS does not use this flexfield. You may skip these steps and proceed to Defining Descriptive Flexfields.

It is not mandatory to define your collective agreement grades key flexfield now. You can do it after you have defined your Business Groups.

After you have specified your requirements for recording and reporting agreement grade information in your enterprise, the implementation sequence which you follow is:

Step 36: Design your Collective Agreement Grades Flexfield Structures

You need to design a Collective Agreement Grades Flexfield Structure for each Grade Structure you want to hold in Oracle Human Resources. For each structure you must include the following:

- The Structure Title (the Grade Structure) and the number of Segments.
- The Flexfield Segment Names (the Grade Factors), Order and Validation Options.
- The Flexfield Value Sets to be used and any lists of values.

Note: Your system administrator performs this step.

Step 37: Define Collective Agreement Grades Flexfield Value Sets

To validate the values that a user can enter for any segment, you must define a specific Value Set.

The attributes of the Value Set will control the type of values that can be entered, and how many characters each segment can hold. The attributes of the Value Set will also control how the values are to be validated.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 38: Define Collective Agreement Grades Flexfield Segments

Define a structure for your Collective Agreement Grades Flexfield that contains the segments you want to use. You use this structure to create your Reference Grades in the Define Agreement Grades window.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you cannot enter new details in the Define Agreement Grades window.

When you access the grades in the Assignment window they display in the numerical order defined in the Number column of the Segments Summary window.

Note: You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

Use the Key Flexfield Segments window.

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 39: Define Collective Agreement Grades Flexfield Segments Values

If you have chosen Independent or Dependent validation for a Value Set used by a Collective Agreement Grades Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Segment Values window.

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 40: Define Collective Agreement Grades Flexfield Cross-Validation Rules

Define any Cross-Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window.

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 41: Define Collective Agreement Grades Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Define Shorthand Aliases window.

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 42: Freeze and Compile Your Collective Agreement Grades Flexfield Structure

You are now ready to freeze your Collective Agreement Grades Flexfield definition. Navigate to the Define Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. Oracle Human Resource Management Systems now freezes and compiles your Collective Agreement Grades Flexfield definition. Compiling the flexfield definition enables the flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window.

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Descriptive Flexfields

Use descriptive flexfields in Oracle HRMS to define your own additional fields to the standard windows. For example, if you want to record *Driver's License Number* for any person you can define a segment of the *Additional Personal Details* flexfield to record this additional information.

After this, you can enter a *Driver's License Number* in the Person window after the standard Personal details.

> **Note:** The descriptive flexfield is defined at the level of the base-table. This means that any window which uses the base-table will display the same descriptive flexfield segments. In this example, the *Driver's License Number* will appear in the Contact window, as well as the Person window.

Before you begin to implement any descriptive flexfield you must clearly specify your requirements. You must include the following details:

- The Context and the number of Segments for each Context
- The Flexfield Segment Names, Order and Validation Options

- The Flexfield Value Sets to be used and any lists of values

You can define two types of descriptive flexfield Segments:

- **Global Segments**

  Segments always appear in the flexfield window.

- **Context-Sensitive Segments**

  Segments appear only when a defined context exists. You can prompt a user to enter the context, or you can provide the context automatically from a reference field in the same region.

  Note: Often you can choose between using a code, a 'base-table' field, and a field which contains a meaning or description. You should always use base-table fields as reference fields for Context-Sensitive segments. These fields usually have the same name as the column in the base table.

  Some of the Standard Reports supplied with the system include descriptive segment values. If you follow this suggestion, these reports will be able to use the prompts you define - otherwise they will apply a generic prompt to the data.

  Note: If you want to include descriptive flexfield Segment Values in the Lookups list for DateTrack History you need to modify the DateTrack History Views that are supplied with the system.

Define Descriptive Flexfields

Step 43: Register a Reference Field

You must use the *Application Developer* Responsibility to update the definition of the descriptive flexfield. From the Descriptive Flexfields window, navigate to the Reference Fields block and enter the name of the Reference Field you want to use.

Warning: Some descriptive flexfields are predefined and protected. These are used to deal with specific legislative and reporting needs of individual countries or industries.

Do not attempt to alter the definitions of these protected flexfields. These definitions are a fundamental part of Oracle HRMS. Any change to them may lead to errors in the operating of the application.

Oracle HRMS may use other segments of these flexfields in the future. Therefore, do not add segments to any protected flexfield. This can impair your ability to upgrade your system.

Use the Descriptive Flexfields window

Step 44: Define Flexfield Value Sets

If you want to validate the values which a user can enter for any segment you must define a specific Value Set.

- The attributes of the Value Set will control the type of values that can be entered, and how many characters each segment can hold.

- The attributes of the Value Set will also control how the values are to be validated.

  Note: Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Developer's Guide*

Step 45: Define Descriptive Flexfield Segments.

Define the segments of your descriptive flexfield for each Context.

You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

1. Use Global Context to define any segments which will always appear in the flexfield window.
2. Enter your own Context Name to define segments which will appear only for that context.
3. Freeze and compile your descriptive flexfield definitions.

   Warning: Any segment you define as "Required" is required for every record on the system. You can encounter two common problems:

- If you define a 'Required' segment after you have entered records: Existing records will not have any value in this segment and the system will prompt you with an error when you query an existing record.
  - Some descriptive flexfields are used in more than one block. For example, any 'Required' segments for Additional Personal Details must be entered for every Employee,Contingent Worker, Applicant or Contact.

  Use the Descriptive Flexfield Segments window.

See: Defining Descriptive Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 46: Define Flexfield Segment Values

If you have chosen Independent validation for a Value Set used by a descriptive flexfield Segment, you must define a list of valid values for the Value Set.

Use the Define Segment Values window.

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 47: Run Create Descriptive Flexfields Database Items Process

When you have defined your descriptive flexfields you should run the Create Descriptive Flexfields Database Items process to create database items for your non-context-sensitive descriptive flexfield segments.

You should rerun this process whenever you create additional non-context-sensitive descriptive flexfield segments.

Note: If you require Database Items for Context Sensitive flexfield segments you should consult your Oracle Support Representative for full details of how to add other Database Items.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Extra Information Types (EITs)

Extra Information Types are a type of descriptive flexfield that let you add an unlimited number of information types to six of the most important entities in Oracle HRMS.

For example, you might want to use the EIT on Assignment to hold information about project work within an assignment.

> **Note:** With Organizations you can group the EITs by classification so that when a user selects a classification they will see the EITs associated with the classification. This means that there are some additional steps to implement EITs for an Organization.

Define Extra Info Types (Excluding Organizations)

Step 48: Define Extra Information Types

Once you have decided which extra information types you require, select the relevant descriptive flexfield by title. Create a new record in the Context Field Values region and enter the name of your new Information Type in the Code field. Enter the segment values and compile the descriptive flexfield.

> **Important:** There are some predefined EITs in Oracle US Federal Human Resources. These definitions are a fundamental part of your installation and any change to them may lead to errors in the operation of the system. Do not attempt to alter the definitions of these protected flexfields or to add other segments to them. It is possible that Oracle will use other segments of these flexfields in the future. Any changes you make can affect your ability to upgrade your system in the future.

Use the Descriptive Flexfield Segments window.

See: Setting up Extra Information Types (Excluding Organization EITs), *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 49: Set Up Responsibility Access for Extra Information Types

EITs do not appear automatically in any responsibility. You must set up responsibility level access for EITs. Alternatively, use CustomForm security to add individual EITs to a specific taskflow window. This level of security is usually defined later in the implementation when you need to restrict access for users.

> **Note:** This security does not apply to EITs on organizations.

Use the Information Types Security window.

See: Setting Up Extra Information Types for a Responsibility, *Configuring, Reporting, and System Administration Guide*.

Define Extra Info Types for Organization

EITs for organization classifications are set up differently from other EITs. When you define them you must also associate them with the Classification of the

Organization. When a user selects the classification then the system will display the correct set of EITs.

Step 50: Define Organization Classification

Define a new organization classification if you want to group your EITs in a specific way. You do not need to do this, if you can use a classification that already exists.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 51: Set up Extra Information Types for an Organization Classification

Define a new EIT and then enter a row into the HR_ORG_INFORMATION TYPES table. Then specify for which organization classifications this EIT is available.

See: Setting Up Extra Information Types for an Organization Classification, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Administration

These are tasks for your System Administrator.

> **Important:** If you used the Configuration Workbench, you can skip these steps. The Workbench enables the currencies for the countries in which your enterprise operates, and creates a user called HRMS_USER for each business group.

Step 52: Enable Currencies

All major currencies are predefined with Oracle Applications. The codes used are the ISO standard codes for currencies. However, you must enable the specific currencies you want to use for your base currency, or for any compensation and benefit information.

The 'base currency' is the default currency used by your Business Group, USD.

> **Note:** Oracle HRMS does not use extended precision. You can control the precision in any calculation using a formula.

Use the Currencies window

See: Enabling Currencies, *Oracle HRMS Configuring, Reporting, and System Administration Guide*.

Step 53: Define 'View All' HRMS User

Before you can access any of the HRMS windows you must create a new Application User. with access to one of the default Responsibilities supplied with the system.

Use the Users window.

See: Users Window, *Oracle Applications System Administrator's Guide*

Enterprise and Workforce Management

Organization Structures

Using the Configuration Workbench, you create the key organization structures for your enterprise and map them into an organization hierarchy. You also define the locations for these organizations. The Workbench creates a "View All" responsibility for each business group, and sets the required user profile options.

You can create additional organizations and locations to represent internal divisions or departments, and external organizations for reporting or third-party payments.

If you have used the Configuration Workbench you can skip steps 1, 2 and 6 through 8. You might want to set additional user profile options (step 3).

Step 1: Adapt or Create Business Group

A business group is a special class of organization. Every business group can have its own set of default values, with its own internal organizations, positions, payrolls, employees, contingent workers, applicants, compensations and benefits.

A 'Setup' business group is supplied with Oracle HRMS. This business group is used by the default responsibility. You can use this business group with all of its default definitions as the starting point for your own business group, or you can define other business groups to meet your own needs.

> **Warning:** The Setup business group has a default legislation code of US and a default base currency of USD.
>
> If you intend to process payrolls in your business group, or you intend to implement legislation for another territory, you may need to create a new business group with a valid legislation code and base currency. The system uses these values to copy in the predefined data it needs to comply with local legislative and processing requirements.
>
> You cannot change these definitions after they have been saved.

Use the Organization window.

See: Adapting and Creating a New Business Group, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 2: Create a 'View All' Responsibility for the Business Group

If you are using the Setup Business Group supplied with Oracle HRMS, you can omit this step.

Use the Responsibility window.

See: Defining a 'View All' Responsibility, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 3: Set User Profile Option Values for Responsibility

Set the HR User Profile Options for the new responsibility. You must set up the HR: User Type option.

You can also set up other User Profile Options.

Use the System Profile Values window.

See: System Profile Values Window, *Oracle Applications System Administrator's Guide*

Step 4: Define Lookup Types and Values

Lookups supply many of the lists of values in Oracle HRMS. For example, the Job, Position, Person, and Assignment windows use Lookups for Extra Information.

Many Lookup Types have been predefined and include value sets. Others are predefined, such as Appropriation Codes and Bargaining Unit Status, but you need to define values for them.

For information about which Lookup Types are predefined and contain value sets and which ones are extensible, refer to the reference tables:

- Assignment Information Types, *Configuring, Reporting, and System Administration Guide*
- RPA Extra Information Type Descriptions, *Configuring, Reporting, and System Administration Guide*
- Elements, *Configuring, Reporting, and System Administration Guide*
- Location Information Type Description, *Configuring, Reporting, and System Administration Guide*
- Organization Information Type Description, *Configuring, Reporting, and System Administration Guide*
- Person Information Types, *Configuring, Reporting, and System Administration Guide*
- Position Information Types, *Configuring, Reporting, and System Administration Guide*
- RPA Extra Information Types and NOACs, *Configuring, Reporting, and System Administration Guide*
- Person Analysis Special Information Types, *Configuring, Reporting, and System Administration Guide*

Lookup Values are the valid entries that appear in the list of values. They make choosing information quick and easy, and they ensure that users enter only valid data into Oracle HRMS.

You can add new Lookups Values at any time to extensible Lookup types. You can set the *Enable Flag* for a Value to No, so that it will no longer appear in the list of values, or you can use the Start and End Dates to control when a value will appear in a list for all non-system Lookup Types.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 5: Set up Auto Orgs *Optional Step*

You can enable the automatic creation of HR organizations using the Auto Orgs functionality. If your enterprise has a close relationship between its financial structure and line management hierarchy, then this means you only have to maintain your financial structure in GL and the corresponding line manager hierarchy is automatically synchronized.

See: Implementing Automatic Company Cost Center Creation, *Oracle HRMS Enterprise and Workforce Management Guide*

For further information on setting up multiple organizations see *Multiple Organizations in Oracle Applications R11i,* available on Meta*Link*

Step 6: Create Locations

Create each work location used by your agency and associate it with a Duty Station, using the supplied Duty Stations Lookups. You define each location and address once only. This saves you time if you have several organizations with the same address.

The Location name is displayed on the Assignment form. To more easily identify the specific Duty Station Location, it is recommended that you enter the Duty Station number as the location name.

Use the Location window.

See:Setting up Locations, *Enterprise and Workforce Management Guide*

Step 7: Create Organizations

Organizations are the basic work structure of any enterprise. They usually represent the functional, management, or reporting groups which exist within a business group.

In addition to these internal organizations you can define other organizations for reporting purposes or to represent carriers for benefits.

> **Tip:** When you install Oracle HRMS you will find a predefined list of Organization Classifications. These values are defined for the Lookup Type ORG_CLASS, and provide options for all users of the Organization window.
>
> You can disable the Lookup values you will not use in your implementation in the Application Utilities Lookups window.

If you intend loading historic assignment details into Oracle HRMS, make sure you enter valid dates. You cannot assign an employee or contingent worker to an organization before the start date of the organization.

> **Tip:** Consider using a fixed date as a default for your initial setup, for example, 01-JAN-1951. This will simplify your data-entry.

Use the Organization window.

See: Creating an Organization, *Enterprise and Workforce Management Guide*.

Step 8: Enter Organization Classifications and Additional Information

Enter the appropriate classifications for each organization and details for any Extra Information types.

Use the Organization window.

See: Entering Organization Classifications, *Oracle HRMS Enterprise and Workforce Management Guide*

See: Entering Additional Information, *Enterprise and Workforce Management Guide*

Step 9: Enter Business Group Additional Information

For each business group you set up, you need to enter additional information, specifying the names of the key flexfield structures that you set up previously for

Job, Position, Grade, Group, and Costing. This information is required to process and update to the HR database Requests for Personnel Actions.

Use the Organization window.

See: Entering Businss Group Information, *Enterprise and Workforce Management Guide*

Step 10: Set up HR Organization and US Federal Org Reporting Information

You specify HR Organization for all organizations to which you intend to assign employees. For these organizations, you also enter the the US Federal Org Report information which is required when generating federal reports such as the Notification of Personnel Action and the Central Personnel Data File (CPDF) reports.

Use the Organization window.

See: Federal reports, see HR Organizations: Entering US Federal Reporting Information, *Enterprise and Workforce Management Guide*

Step 11: Define Organization Hierarchies

A business group can include any number of organizations. You can represent your management or other reporting structures by arranging these organizations into reporting hierarchies. An organization can belong to any number of hierarchies, but it can only appear once in any hierarchy.

> **Note:** You may find it easier to define the primary reporting hierarchy using the top organization and one other. Then you can add organizations into the hierarchy when you make your definitions in the Organization window.

Organization reporting lines change often and you can generate a new version of a hierarchy at any time with start and end dates. In this way, you can keep the history of your organizational changes, and you can also use this feature to help you plan future changes.

When you use DateTrack you see the 'current' hierarchy for your effective date.

> **Important:** Your primary reporting hierarchy will usually show your current management reporting structure. You can define other hierarchies for other reporting needs.

You can create organization hierarchies using the:

- Organization Hierarchy Window

  See: Creating Organization Hierarchies, *Oracle HRMS Enterprise and Workforce Management Guide* .

- Organization Hierarchy Diagrammers (they enable you to create your hierarchies graphically, and to make intuitive drag-and-drop changes).

  See: Adding Organizations or Positions to a Hierarchy, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 12: Run the Create Federal HR Valid Combinations

The application provides a concurrent manager process that supplies valid grade and pay plan combinations that are used during implementation when defining your agency's positions.

Use the Submit Request Window.

See: Running the Federal Valid Combinations Process, *Compensation and Benefits Management Guide*

Jobs

If you used the Configuration Workbench, you may already have loaded jobs from a spreadsheet in the Workbench. You can skip the Define Jobs step.

Step 13: Define Job Groups

As part of a working relationship, a person can simultaneously perform a number of roles in addition to being an employee or contingent worker. These can range from initiatives defined by the enterprise, such as fire warden, to legislative defined roles such as Health and Safety Representative. In Oracle HRMS, these are know as supplementary roles. Supplementary roles are set up as jobs in the Job window

Each job is held in a Job Group. The Job Group is used to store jobs of a similar type together in one group. All standard jobs created in Oracle HRMS, that is, those jobs that define the role the person is employed to fulfil, must be stored in the default HR Job Group. This Job Group is automatically created for your business group.

If you want to set up supplementary roles you must set up additional job groups to store these roles.

Use the Job Groups window.

See: Creating a Job Group, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 14: Define Jobs

Jobs provide a way to categorize related positions, independent of specific organizations. You use the Job window to define a job and associate it to an Occupational Series Code.

A 'Job Name' is a unique combination of values in the segments of the job flexfield structure that you have linked to your business group.

There are a number of ways to add information about a job. Primarily, you use Extra Information flexfields to store additional job-related information.

Use the Job window.

See: Defining a Job, *Enterprise and Workforce Management Guide*

Define a Context for Mass Actions

The Contexts form determines what information you can view, enter, and change on the Mass Assignment Update and Position Copy forms. A predefined global Context form contains the default position and assignment attribution that appear on the forms. When you create a new Context, you can choose the attributes to display based on a user's Application, Legislation, and Responsibility.

Step 15: Set up a new Context

Create a new context defining the Application, Legislation, and Responsibility.

Use the Contexts window.

See: Defining a Context for Mass Actions, *Configuring, Reporting, and System Administration Guide*

Step 16: Specify which Attributes to Include

Define the attributes to include in the Display, Change List, and Criteria columns.

Use the Contexts window.

See: Defining a Context for Mass Actions, *Configuring, Reporting, and System Administration Guide*

Positions

If you used the Configuration Workbench, you may already have loaded positions from a spreadsheet in the Workbench. You can skip the Define Positions step.

Step 17: Create Classified Position Descriptions (Optional)

Many agencies require classified position descriptions that describe the position's responsibilities, requirements, and working conditions. You can create as well as classify position descriptions.

Use the Position Description window.

See: Classifying Position Descriptions, *Enterprise and Workforce Management Guide*

Step 18: Define Position Hiring Statuses

Each position must have a hiring status: Proposed, Active, Frozen, Eliminated or Deleted. You can create user names for these system hiring statuses, and define more than one user name for each system name, if required.

Use the User Types and Statuses window.

See: Defining Hiring Statuses, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 19: Define Positions

In Oracle HRMS a position is a job within an organization. You use positions to define your structural information.

A 'Position Name' is a unique combination of values in the segments of the position flexfield structure that you have linked to your business group.

Use the Position window.

See: Define a Position, *Enterprise and Workforce Management Guide*

Step 20: Enter Additional Information about Positions

You use Extra Information flexfield to store additional position-related information.

See: Define a Position, *Enterprise and Workforce Management Guide*

Step 21: Set up the Synchronize Positions Process to Run Nightly

Oracle HRMS uses the Synchronize Positions process to update the non-datetracked Positions table (PER_ALL_POSITIONS) with changes made to the datetracked table (HR_ALL_POSITIONS_F). When you run the process, any datetracked changes with an effective date on or before today are applied to the non-datetracked table. Future dated changes are not applied until they become effective.

Running the Synchronize Positions process every night ensures that the application automatically updates the table with the position changes that become effective each day. If a power or computer failure disrupts this process, you can start it manually from the Submit a New Request window.

Warning: Ensure that the resubmission interval is set to run every night.

Use the Submit a New Request window.

See: Submitting a Request, *Oracle Applications User Guide*

Step 22: Create a Position Hierarchy

You structure position hierarchies to represent the management reporting lines for your agency's departments and sections. Hierarchies represent reporting relationships that cross organizations, position and organization information for the Requests for Personnel Action, and hierarchical relationships for the Organizational Component Translation report.

You can create position hierarchies using the:

- Position Hierarchy Window

  See: Creating a Position Hierarchy, *Enterprise and Workforce Management Guide*

- Position Hierarchy Diagrammers (they enable you to create your hierarchies graphically, and to make intuitive drag-and-drop changes).

  See: Adding Organizations or Positions to a Hierarchy, *Oracle HRMS Enterprise and Workforce Management Guide*

Complaint Tracking

Before your agency enters a complaint into the application, you must set up the Complaint Tracking window according to your agency's requirements.

Setup involves consideration of implementing security for users of Complaint Tracking windows and reports, adding agency-specific fields and Lookup values to windows, and setting up EEO Officials in the application.

Your setup may include one or more of the following steps, depending on your implementation:

Step 23: Define a Secure Responsibility for EEO Complaint Tracking Users

Use the Security Profile window.

See: Defining a Security Profile, *Configuring, Reporting, and System Administration Guide*

Step 24: Add lookup values for:

- Serviced HR Office (GHR_US_HR_OFFICE)
- Serviced EEO Office (GHR_US_EEO_OFFICE)
- Servicing Organization (GHR_US_SERVICED ORG)
- Servicing organizations are those involved in processing the complaints.
- Discriminating organizations (GHR_US_DISCRIMINATING_ORG)

Use the Lookup Values window.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Configuring, Reporting, and System Administration Guide*, Removing Lookup Values, *Configuring, Reporting, and System Administration Guide*

Step 25: Set up person types

By defining additional person types, such as ADR Facilitator or Administrative Judge, you can include information when you enter people in the application that allows you to query them by type of function.

See: Defining Person Types, *Enterprise and Workforce Management Guide*

Step 26: Define a request group for EEO reporting users.

See: Request Groups, *Configuring, Reporting, and System Administration Guide*

Step 27: Add agency-specific taskflows

Use the Define Taskflow window.

See: Defining Taskflows, *Configuring, Reporting, and System Administration Guide*

Step 28: Enter EEO Officials into the application

Complaint officials must be entered in the application before they can be assigned to a complaint.

Use the People window.

See: Entering a New Person, *Workforce Sourcing, Deployment, and Talent Management Guide*

See: Entering People Involved in Processing the Complaint, *Enterprise and Workforce Management Guide*

Step 29: Generate docket numbers

If you do not want to enter docket numbers manually in the Complaint Tracking window, you can generate your own docket numbers using the same SQL operation that you do to generate numbers for Request Number field in an RPA (Request for Personal Action).

New Hire Reporting

Step 30: Check GRE Federal and State Identification Numbers

Ensure that a federal identification number and a SUI identification number, if appropriate, is on record for each GRE that submits New Hire reports.

Use the Organization window.

See: GREs: Entering the IRS Identification Number, *Enterprise and Workforce Management Guide*

Step 31: Enter the GRE Contact Person

Enter the GRE contact person.

Use the Organization window.

See: Entering New Hire Report Information for a GRE, *Enterprise and Workforce Management Guide*

Step 32: Enter New Hire Information for Every Employee

When you use the online system to hire an employee you enter the appropriate New Hire status in the *Employment Information region* of the Person window.

- The default is null
- Enter **Incl** or **Excl**
- The status automatically changes to **Done** after a run of the New Hire report includes the employee.

> **Warning:** When you load your current employees into the database, the default New Hire Status is null. You must enter a value of **Done** or **Excl** in the New Hire Status field if you do not want to include them in your first run of the New Hire report.
>
> Do this manually, or as part of your data loading.

Use the Person window.

See: Entering Additional Personal Information (People Window), *Workforce Sourcing, Deployment, and Talent Management Guide*

Evaluation Systems

Step 33: Define Evaluation Types *Optional Step*

With Oracle HRMS you can record summary evaluation information for Jobs, or Positions in the Evaluation window.

Define the name of your evaluation system as a value for the Lookup Type EVAL_SYSTEM.

To record detailed evaluation scores for the Hay System or any other system you can enable the Additional Evaluation Details descriptive flexfield to hold and validate this information.

You can also hold comment or review information for each evaluation you undertake.

> **Note:** If you use more than one evaluation system you may want to define the segments as context sensitive to the System Name.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Human Resource Budgets

Step 34: Define Lookup Types and Values *Optional Step*

Oracle HRMS delivers the following seeded budget measurement units: Money, Hours, Headcount, Full Time Equivalent, and Percent Full Time Equivalent. You cannot extend the delivered budget measurement units, but you can copy and rename an existing measurement unit.

Define values for BUDGET_MEASUREMENT_TYPE.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 35: Define Period Types *Optional Step*

The most common period types are already predefined in Oracle HRMS. You can change the names of these predefined types but you cannot add any new types.

Use the Period Types window.

See: Renaming Period Types, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 36: Define Budgetary Calendars *Optional Step*

You use calendars to define the budget years for your staffing budgets.

Use the Budgetary Calendar window.

See: Defining Budgetary Calendars, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 37: Define Budget Sets *Optional Step*

A budget set is comprised of one or more elements. You define a budget set to record the money or hours or other budget measurement units in your budget. Oracle HRMS uses budget sets to track actual expenditures and commitments.

Use the Budget Set window.

See: Defining Budget Sets, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 38: Migrate an Existing Oracle HRMS Budget to the New Budget Tables *Optional Step*

If you created a budget in Oracle HRMS prior to Release 11*i*, you can use an existing budget as the basis for a new budget worksheet.

Run the Migrate Budget Data process from the Submit Requests window to migrate an existing budget to the new database tables for Budgets.

See: Migrating a Budget to Oracle HRMS, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 39: Set Up the HR Budget in Oracle General Ledger *Optional Step*

If you are transferring a budget from Oracle HRMS to Oracle General Ledger, you must first define the budget in Oracle General Ledger.

Use the Define Budget window in Oracle General Ledger to define the budget.

See: Setting Up an Oracle HRMS Budget for Transfer to Oracle General Ledger, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 40: Define Budget Characteristics *Optional Step*

You set up budget characteristics to define the Oracle HRMS work structure for which you are establishing a budget. The primary entities against which you can create a budget are job, position, grade, and organization. You can also create a budget for a combination of these entities.

Defining the characteristics of a budget also requires you to define the budget measurement units (Money or Headcount, for example). Optionally, you can select the elements that are used to process budget funding commitments during a budgetary

period. For budgets that are transferred to Oracle General Ledger, you can map Oracle HRMS Costing Segments to GL Chart of Account Segments.

Use the Budget Characteristics window.

See: Defining Budget Characteristics, *Oracle HRMS Enterprise and Workforce Management Guide*

Person Types

Step 41: Define Person Types *Required Step*

You can define your own names to identify the "types" of people in your system.

Note: Person Type is a common option for Form Customization.

Use the Person Types window.

See: Defining Person Types, *Enterprise and Workforce Management Guide*

Collective Agreements

A collective agreement is an agreement that defines the terms and conditions of employment for all employees that are covered by its terms. Agreements are typically negotiated and agreed by external bodies such as Trade Unions and Representatives of Employers.

Step 42: Setting Up Collective Agreements *Optional Step*

If your enterprise uses collective agreements, follow the steps in the referenced topic to enter a collective agreement, set up the eligibility criteria for the agreement, and to apply the values defined in the agreement to the eligible employees.

See: Setting Up a Collective Agreement, *Oracle HRMS Enterprise and Workforce Management Guide*

Medical Assessments, Disabilities and Work Incidents

Step 43: Define Lookup Types and Values *Optional Step*

If you want to record medical assessments, disabilities, or work incidents for the people in your enterprise, you must define Lookup Values for the Lookup Types that are used in those windows.

See: User and Extensible Lookups, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Compensation, Benefits, and Payroll

Oracle HRMS uses elements to represent all types of earnings, deductions and benefits. Elements hold the information you need to manage compensation and benefits.

Before you start defining elements, you should make all of your decisions about the definitions and rules for eligibility.

If you plan to load details of employee entry history you should consider using a fixed date, such as 01-JAN-1901, as a default for your initial setup definitions. This will simplify your data-entry.

Many implementation steps are shared by Standard and Advanced Benefits. Federal Employee Health Benefits, Thrift Savings Plans, and Thrift Savings Plan Catch-Up Contribution plans are predefined for you. See: Federal Employee Health Benefits Overview, *Compensation and Benefits Management Guide*, Thrift Savings Plan Overview, *Compensation and Benefits Management Guide*.

Define Grade Related Information *Optional Steps*

If you have agency-specific pay plans, you can add them to the system. The grade and step values are also extensible.

Step 1: Add the Pay Plan

The Federal Maintenance Pay Plan window lists the predefined pay plans. Use this window to add other pay plans and to indicate whether the pay plan is eligible for Within-Grade-Increases.

Use the Pay Plan window.

See: Add a Pay Plan, *Compensation and Benefits Management Guide*

Step 2: Review the Values for Grades or Levels

The product comes with an extensive list of values for grades and steps that you can extend, GHR_US_GRADE_OR_LEVEL.

Use the Lookup Values window.

See: Add Grades, *Compensation and Benefits Management Guide*

Step 3: Review the Values for Steps or Rates

The product comes with an extensive list of values for grades and steps that you can extend, GHR_US_STEP.

Use the Lookup Tables window.

See: Add Steps, *Compensation and Benefits Management Guide*

Step 4: Associate the Pay Plan and Grade

To create valid pay plan and grade combinations, you associate the pay plan with the appropriate grade or level.

Use the Grades window.

See: Associate Pay Plans and Grades, *Compensation and Benefits Management Guide*

Step 5: Create a Pay Table

A pay table can include one or more pay plans. For example, the Oracle Federal Standard Pay Table includes several pay plans. You can create your agency-specific basic and special rate pay tables.

Use the Table Structure window.

See: Set up Pay Tables, *Compensation and Benefits Management Guide*

Step 6: Enter the Pay Values

You enter and can later maintain the values for the pay tables.

Use the Table Values window.

See: Enter Pay Values, *Compensation and Benefits Management Guide*

Define Payroll Information

You must include a payroll in the employee assignment before you can make nonrecurring entries of any element for an employee. Nonrecurring entries are only valid for one payroll period.

Step 7: Define Payment Methods

Standard categories of payment methods such as Direct Deposit are predefined with your system. You can define your own names for each of these methods.

Use the Organizational Payment Method window.

Step 8: Define Consolidation Sets

When you define your business group the system will automatically generate a default Consolidation Set. If you have not installed Oracle Payroll you can skip this step.

Consolidation sets are used by Oracle Payroll where you want to gather the results from different payroll runs into a single set for reporting or transfer to other systems. You can define any number of additional consolidation sets.

Use the Consolidation Sets window.

Step 9: Define Payrolls

You define your own payroll groups to meet your business needs for processing and payment. Agencies must define at least:

- One biweekly payroll and name it BiWeekly.

  The biweekly payroll is the one that most federal offices use. The effective date for the payroll should begin at a date that accommodates all records that the agency plans to convert.

- One monthly payroll

  The monthly payroll is used as the default benefits assignment payroll for ex-employees and beneficiaries. The effective date for the payroll should begin at a date that accommodates all records that the agency plans to convert.

  > **Note:** The payroll calendar is different from the budgetary calendar in Oracle HR. You define your budgetary calendar for headcount or staffing budgets.

Use the Payroll window.

Administration Steps for Standard and Advanced Benefits

Step 10: Add the Benefits Tabbed Region to the People Window *Optional Step*

A person with a responsibility of system administrator or application developer can use the Menus window to add the benefits alternate region to the People window.

1. Query the BEN_MANAGER menu in the Menu field.
2. Add a new line and select HR View Benefits in the Function field.
3. Save your work

Use the Menus window.

See: Menus Window, *Configuring, Reporting, and System Administration Guide*

Step 11: Define a Monthly Payroll *Required Step*

You must define a monthly payroll for each business group you maintain. When you process employee terminations, a copy of the person's assignment record is created as a benefits assignment. Benefits assignments are used to maintain eligibility for continuing benefits, and always have a payroll with a monthly period.

> **Note:** If you have already defined payroll information, including monthly payrolls for each Business Group, you can skip this step.

Use the Payroll window.

See:Defining Benefits Defaults for a Business Group, *Enterprise and Workforce Management Guide*

Benefits Eligibility

You define participation eligibility profiles to determine eligibility for benefits. You can also use eligibility factors to determine variable contribution and distribution rates for a benefit.

Step 12: Define Benefits Groups *Optional Step*

You define a benefits group as a category of people who can be either included or excluded from receiving a benefit or a standard activity rate. A benefit group is one optional component of an eligibility profile or a variable rate profile.

Use the Benefits Groups window.

See: Defining Benefits Groups, *Oracle HRMS Compensation and Benefits Management Guide*

Step 13: Define Postal Code (ZIP) Ranges *Optional Step*

You define postal code (zip) ranges if you limit benefits eligibility based on a person's home address or if an activity rate varies based on a person's address.

Postal code ranges are also a component of service areas.

Use the Postal Zip Ranges window.

See: Defining Postal Zip Ranges, *Oracle HRMS Compensation and Benefits Management Guide*

Step 14: Define Service Areas *Optional Step*

You can define a service area to group people who live in a region by their postal codes. A service area is one optional component of an eligibility profile or a variable rate profile.

Use the Service Areas window.

See: Defining Service Areas, *Oracle HRMS Compensation and Benefits Management Guide*

Step 15: Define Regulations *Optional Step*

You define regulations as discrete rules, policies, or requirements that a governmental or policy making body defines regarding the administration of one or more benefits.

Use the Regulations window.

See: Defining Regulations, *Oracle HRMS Compensation and Benefits Management Guide*

Define Derived Eligibility Factors

A derived factor is a system calculated value that you can use to determine eligibility for a benefit or to determine an activity rate.

Step 16: Define Derived Compensation Level Factors *Optional Step*

Define compensation level factors to determine how the system derives a person's compensation level based on a person's stated salary or a balance type that you specify.

Use the Derived Factors window.

See: Defining Derived Factors: Compensation Level, *Oracle HRMS Compensation and Benefits Management Guide*

Step 17: Define Derived Percent of Full Time Employment Factors *Optional Step*

Define percent of full time factors to determine how the system derives a person's percent of full time employment.

Use the Derived Factors window.

See: Defining Derived Factors: Percent of Full Time Employment, *Oracle HRMS Compensation and Benefits Management Guide*

Step 18: Define Derived Hours Worked in Period Factors *Optional Step*

Define hours worked in period factors to determine how the system derives the number of hours a person worked in a given period.

Use the Derived Factors window.

See: Defining Derived Factors: Hours Worked in Period, *Oracle HRMS Compensation and Benefits Management Guide*

Step 19: Define Age Factors *Optional Step*

Define age factors to determine how the system derives a person's age.

Use the Derived Factors window.

See: Defining Derived Factors: Age, *Oracle HRMS Compensation and Benefits Management Guide*

Step 20: Define Length of Service Factors *Optional Step*

Define length of service factors to determine how the system calculates a person's length of service.

Use the Derived Factors window.

See: Defining Derived Factors: Length of Service, *Oracle HRMS Compensation and Benefits Management Guide*

Step 21: Define Combination Age and Length of Service Factors *Optional Step*

Define combination age and length of service factors to combine an age factor and a length of service factor.

Use the Derived Factors window.

See: Defining Derived Factors: Combination Age and Length of Service, *Oracle HRMS Compensation and Benefits Management Guide*

Define Eligibility Profiles *Optional Steps*

Step 22: Define an Eligibility Profile *Optional Step*

Defining an eligibility profile is the primary way in which you implement eligibility requirements for a benefit. You link an eligibility profile with a compensation object (a benefit that you define) so that when eligibility processes run, only the persons meeting the eligibility profile requirements are eligible to receive the benefit.

Use the Participation Eligibility Profiles window.

See: Defining an Eligibility Profile, *Oracle HRMS Compensation and Benefits Management Guide*

Step 23: Define Dependent Coverage Eligibility Profiles *Optional Step*

You define dependent coverage eligibility profiles to enforce eligibility requirements for dependents.

Use the Dependent Coverage Eligibility Profiles window.

See: Defining a Dependent Coverage Eligibility Profile, *Oracle HRMS Compensation and Benefits Management Guide*

Compensation Objects

You define compensation objects as the benefits that you offer to your employees and other eligible participants.

Compensation objects are arranged according to the compensation object hierarchy:

- Program
- Plan Type
- Plan
- Option

Definitions that you set at the program level cascade to the plan types, plans, and options in that program unless you override the definition at a lower point in the hierarchy.

Step 24: Define Reimbursable Goods and Service Types *Optional Step*

Define goods and services that you approve for reimbursement. You then associate one or more goods and services types with a reimbursement plan.

Use the Goods and Services window.

See: Defining Reimbursable Goods and Service Types, *Oracle HRMS Compensation and Benefits Management Guide*

Step 25: Define a Program or Plan Year Period *Optional Step*

You define a program or plan year period to set the coverage boundaries for the duration of a benefit program or plan.

Use the Program/Plan Year window.

See: Defining a Program or Plan Year Period, *Oracle HRMS Compensation and Benefits Management Guide*

Step 26: Define Plan Types *Optional Step*

You define plan types to categorize common types of benefits, such as medical plans or savings plans.

Use the Plan Types window.

See: Defining Plan Types, *Oracle HRMS Compensation and Benefits Management Guide*

Step 27: Define Options *Optional Step*

You define options to indicate the coverage levels available under a plan or to define investment options for a savings plan.

Use the Options window.

See: Defining Options, *Oracle HRMS Compensation and Benefits Management Guide*

Step 28: Define Plans *Optional Step*

A plan is a benefit in which an eligible participant can enroll. Common plans include medical, group term life insurance, and stock purchase plans.

Use the Plans window.

See: Defining a Benefits Plan, *Oracle HRMS Compensation and Benefits Management Guide*

Step 29: Define Reimbursement Plans *Optional Step*

Use reimbursement plans to define goods and services that eligible participants may purchase. The participant can submit a reimbursement claim for the cost of the good or service that was purchased out-of-pocket.

Use the Plan Reimbursement window.

See: Defining a Reimbursement Plan, *Oracle HRMS Compensation and Benefits Management Guide*

Step 30: Define Programs *Optional Step*

You define a program to group together the benefits that you offer as a package. A program typically is comprised of plan types, plans, and options.

Use the Programs window.

See: Defining a Benefits Program, *Oracle HRMS Compensation and Benefits Management Guide*

Elements

Before you define any elements, you should make all of your decisions about the definitions and rules for eligibility.

Define Input Value Validation *Optional Steps*

Step 31: Define Lookup Types and Values *Optional Step*

You define new Lookup Types to create additional lists of values to validate any element input value with a character datatype.

> **Note:** You can also use Lookup Types to validate a flexfield segment. Use the *Table Validation* option for the Value Set and use the Lookups table as the source of your list.

You can add new Lookup Values at any time. You can set the *Enable Flag* for a Value to No, so that it will no longer appear in the list of values, or you can use the Start and End Dates to control when a value will appear in a list.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 32: Define User Tables *Optional Step*

With Oracle HRMS you can set up any number of *User-Defined Tables*. A user-defined table is a "matrix" of columns that hold different values for the same row. You can access this information using the *GET_TABLE_VALUE* function in any formula.

For example, you may want to set up a single table to hold union pay rates, deductions, and benefit levels for different job groups. Use the rows to hold "Job Group" and the columns to hold the specific values for each job group.

You can define exact row values or an inclusive range of values.

Use the Table Structure window.

See: Setting Up User Tables, Columns and Rows: , *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 33: Define Table Values *Optional Step*

You now need to define the table values.

Use the Table Values window.

See: Entering Table Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 34: Define Element Validation Formulas *Optional Step*

When you define input values you can use a formula to validate any entry to that input value.

> **Important:** You must define the formula before you define the element input value. The type of formula is *Element Input Validation*.

Use the Formula window.

See: Writing Formulas for Validation, *Oracle HRMS FastFormula User Guide*

Define Other Elements *Optional Steps*

Step 35: Define Elements and Input Values

Elements are the basic components of all compensation and benefit types. The product comes with predefined elements that you use with a Request for Personnel Action (RPA). Nature of Action Codes (NOACs) determine how you initiate, update, or modify an element using the RPA. You can also create elements using the Element window.

See: Defining an Element, *Compensation and Benefits Management Guide*.

Step 36: Define Element Links

You can give an entry to an employee only when they are eligible for that element. Employees are eligible for an element when their assignment details match the link details.

You can link an element to any combination of organization, group, grade, job, position, payroll, location, or employment category. However, the Oracle US Federal Human Resources predefined elements must be linked to all payrolls.

Use the Element Link window.

See: Defining Element Links, *Compensation and Benefits Management Guide*

Step 37: Activate Predefined Elements

When you install Oracle HRMS a number of predefined elements are installed. These elements represent the legislative deductions that are processed in the payroll run.

To activate these predefined elements you need only define links for them. You must link the elements predefined for Oracle US Federal Human Resources to all payrolls. You cannot change the assignment components for these predefined links.

Use the Element Link window.

See: Defining Element Links, *Compensation and Benefits Management Guide*

Activity Rates and Coverage Calculations

Activity rate calculations determine the contribution rate necessary to purchase a benefit and the distribution rate for benefits that provide distributions.

Step 38: Calculate Variable Activity Rates *Optional Step*

You define variable activity rate calculations if an activity rate for a compensation object can vary by participant.

Use the Variable Rate Profiles window.

See: Defining General Information for a Variable Rate Profile, *Oracle HRMS Compensation and Benefits Management Guide*

Step 39: Calculate Coverages *Optional Step*

You define the amount of coverage available under a benefit plan for those plans that offer a range of coverage options. Your coverage calculation can include the minimum and maximum coverage level available regardless of the calculation result. For Advanced Benefits customers, coverage levels can vary based on life events.

Use the Coverages window.

See: Defining a Coverage Calculation for a Plan, *Oracle HRMS Compensation and Benefits Management Guide*

Step 40: Define Across Plan Type Coverage Limits *Optional Step*

You can define the minimum and maximum coverage amount that a participant can elect across plan types in a program.

Use the Coverage Across Plan Types window.

See: Defining a Coverage Limit Across Plan Types, *Oracle HRMS Compensation and Benefits Management Guide*

Step 41: Calculate Actual Premium Costs *Optional Step*

You need to maintain the criteria used to calculate the actual premium cost that a plan sponsor owes to a benefits supplier.

Use the Actual Premiums window.

See: Defining an Actual Premium, *Oracle HRMS Compensation and Benefits Management Guide*

Step 42: Define Period-to-Date Limits *Optional Step*

You define period-to-date contribution limits for those plans or options in plan that restrict participant contribution levels in a year period. When you define a standard contribution, you can associate a period-to-date limit for those plans or options in plan that require contribution restrictions.

Use the Period-to-Date Limits window.

See: Defining Period-to-Date Limits, *Oracle HRMS Compensation and Benefits Management Guide*

Step 43: Define Activity Rates for Standard Contribution/Distribution *Optional Step*

You define a standard activity rate calculation to calculate a benefit's contribution or a distribution amount.

Note: You must have already created the corresponding elements.

Use the Standard Contributions/Distributions window.

See: Defining Activity Rates for a Standard Contribution/Distribution, *Oracle HRMS Compensation and Benefits Management Guide*

Additional Setup for Health and Welfare

Step 44: Define Reporting Groups *Optional Step*

You can define a reporting group that you link to one or more programs and plans. When you run a report for a reporting group, the report results are based on the programs and plans that you include in the reporting group.

You can also define the regulatory bodies and regulations govern a reporting group.

Use the Reporting Groups window.

See: Defining a Reporting Group, *Oracle HRMS Compensation and Benefits Management Guide*

Step 45: Define Benefit Balances *Optional Step*

Benefit balances are useful for transitioning legacy system data to Oracle HRMS. You define a benefit balance type and then assign a value to that type for a given person using the Person Benefit Balances window.

Use the Benefit Balances window.

See: Defining a Benefit Balance, *Oracle HRMS Compensation and Benefits Management Guide*

Absence Management and Accruals

You can set up as many plans as you need to permit employees to accrue PTO to use for vacation or sick leave. Each plan has the units of Hours or Days, and can have its own rules regarding accrual frequency, accrual bands, ceilings, carryover, start dates, entitlement of employees with different assignment statuses, and so on.

> **Note:** Oracle Federal Human Resources includes predefined accrual elements that can accept reverse payroll data from an agency. However, you need to set up an absence type and link the predefined elements to it.

Set Up Absence Management

Step 46: Set Up Proration and Notifications *Optional Step*

If you want to associate recurring elements with absence types, you must set up proration and retro notifications. This ensures that absences that end in the middle of a payroll period are detected and processed by the payroll run, and that retrospective changes to absences are recorded in the Retro Notifications report.

> **Note:** Proration is available to Oracle Payroll users in selected localizations only.

See: Setting Up Absence Management, *Oracle HRMS Compensation and Benefits Management Guide*

Step 47: Define an Absence Element *Optional Step*

For each of your accrual plans, or each type of absence you are tracking, you define a nonrecurring element and input value to hold the actual time taken for vacation or sick leave.

If you use Oracle Payroll and the proration functionality is available for your localization, you can use a recurring element instead. This enables you to begin processing a long term absence before you enter an end date, and to apportion time correctly over payroll periods.

Use the Element window.

See: Defining and Linking an Absence Element, *Oracle HRMS Compensation and Benefits Management Guide*

Step 48: Link the Absence Element *Optional Step*

Link each absence element to define who is eligible to take this kind of absence.

Use the Element Link window.

See: Defining Element Links, *Oracle HRMS Compensation and Benefits Management Guide*

Step 49: Define Categories of Absence Types *Optional Step*

Define categories of absence types as values for the Lookup Type ABSENCE_CATEGORY, and your absence reasons as values for the Lookup Type ABSENCE_REASON.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 50: Define Absence Types and Associate with Absence Elements *Optional Step*

If you expect to record absent time using the Absence Detail window, define absence types, associating each with an absence element.

Use the Absence Attendance Type window.

See: Defining an Absence Type, *Oracle HRMS Compensation and Benefits Management Guide*

Step 51: Make Initial Element Entries *Optional Step*

For an absence type with a decreasing balance, use the Element Entries window or the Element Entry API to make initial element entries for employees eligible for the type. You can also initialize a decreasing balance by entering a negative value using BEE. For example, if you enter -16 hours using BEE, a decreasing balance starts at 16 hours. However, be aware that using BEE creates an absence record that will show on employees' absence history.

If you want to make batch entries, see Making Batch Element Entries Using BEE, *Oracle HRMS Configuring, Reporting, and System Administration Guide*.

Step 52: Create Payroll Formula to Calculate Absence Duration *Optional Step*

If you defined a recurring element, create a payroll formula that handles proration to process the element and calculate the appropriate absence duration in each pay period (taking into account the number of days or hours in a month, working and shift patterns, public holidays, and so on).

See: Sample Payroll Formulas Enabled for Proration, *Oracle HRMS FastFormula User Guide*

Set Up Accrual Plans

After completing the absence management setup steps, follow these additional steps to set up a PTO accrual plan:

Step 53: Define New Accrual Start Rules *Optional Step*

There are three seeded start rules: Hire Date, Beginning of Calendar Year, and Six Months After Hire Date. If you need other rules, define them as values for the Lookup Type US_ACCRUAL_START_TYPE.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 54: Decide on Accrual and Carry Over Formulas *Optional Step*

Decide which Accrual and Carry Over formulas to use. You can use the seeded formulas, configure them, or write your own.

Use the Formula window.

See: Writing Formulas for Accrual Plans, *Using Oracle FastFormula*

Step 55: Write Ineligibility Formula *Optional Step*

If your Accrual formula defines a period of ineligibility and you want to use Batch Element Entry (BEE) to enter absences against the accrual plan, define an Ineligibility formula. BEE calls this formula to check whether an employee is eligible to use accrued PTO.

> **Note:** If you use the seeded Accrual formulas, you do not need to define an Ineligibility formula. They use a period of ineligibility entered on the Accrual Plan form, and BEE validation can use the same value.

Use the Formula window.

See: Writing Formulas for Accrual Plans, *Using Oracle FastFormula*

Step 56: Define New Accrual Categories *Optional Step*

There are several seeded accrual categories. If you need additional categories, define them as values for the Lookup Type US_PTO_ACCRUAL.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 57: Select PTO Balance Type *Optional Step*

Oracle Payroll users: If you want to use a payroll balance to store gross accruals, decide whether the payroll run should update accruals as of the run's *date earned* (the date the payroll run uses to determine which element entries to process) or *date paid* (the date that appears on pay advices). Select your choice for the business group.

See: Business Groups: Selecting a PTO Balance Type, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 58: Create Balance Dimensions *Optional Step*

Oracle Payroll users: If you want to use a payroll balance to store gross accruals, consider whether you need to define a new balance dimension. Dimensions are predefined that reset the balance each year on 1 January, 1 June, or hire date anniversary. If you require your balance to accumulate over a different period of time, or reset at a different date, you must create your own balance dimension.

See: Balances in Oracle Payroll, page 2-160

Step 59: Define a PTO Accrual Plan *Optional Step*

Define the accrual plan, selecting the formulas and absence element it is to use.

Use the Accrual Plan window.

See: Defining a PTO Accrual Plan, *Oracle HRMS Compensation and Benefits Management Guide*

Step 60: Set Up Length of Service Bands *Optional Step*

Optionally, set up length of service bands for the plan.

Use the Accrual Bands window.

See: Setting Up Length of Service Bands, *Oracle HRMS Compensation and Benefits Management Guide*

Step 61: Review the Net Calculation Rules *Optional Step*

Review the net calculation rules for the plan. If necessary, create additional elements and associate them with the plan by selecting them in the Net Calculation Rules window.

See: Changing Net Accrual Calculations, *Oracle HRMS Compensation and Benefits Management Guide*

Element Sets and Batch Control Totals

Step 62: Define Element Sets *Optional Step*

In Oracle HRMS you can define a set of elements to:

- Restrict access to elements using *Form Customization*
- Distribute costs across a *Distribution Set* of elements
- Process a restricted set in a *Payroll Run*
- Enter values for a restricted set using BEE (Batch Element Entry)

You define an element set as a named list of elements such as Salary, or Salary and Bonus. You can also define an element set using the classification. For example, you can restrict access to all elements in the classification *Earnings*.

Use the Element and Distribution Set window.

See: Defining an Element or Distribution Set , *Oracle HRMS Compensation and Benefits Management Guide*

Step 63: Define Batch Control Types *Optional Step*

If you use Batch Element Entry, you can set up batch control types to sum the entries in any numerical input value. This enables users to validate a BEE batch against control totals to check for missing lines or miskeying of amounts.

Use the Application Utilities Lookups window.

See: Setting Up BEE Validation Using Control Totals, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Basic Benefits

If you are beginning a new setup for benefits administration, we recommend that you implement the Standard Benefits feature set. Federal Employee Health Benefits and Thrift Savings Plan benefits are implemented using Standard Benefits. Basic Benefits provides a limited set of features and is provided mainly for compatibility with earlier releases.

Step 64: Set Up Basic Benefits *Optional Step*

To set up basic benefits, you define elements for benefits plans, use element links to establish eligibility, and create benefit carriers as organizations.

See: Setting up Basic Benefits, *Compensation and Benefits Management Guide*

Workforce Sourcing and Deployment

Oracle HRMS enables you to define your own names to identify the 'types' of people in your system, and to identify the status of employees and contingent workers in each assignment using your own names.

Recruitment

Step 1: Define Assignment Statuses for Applicants

Assignment Statuses for applicants enable you to define the distinct stages of your own recruitment processes.

With Oracle HRMS you can use your own names to identify these stages. For example, you might want to define a special status to identify applicants who have been invited to a *First Interview* and applicants who have been *Rejected on Application*.

These user statuses enable you to track the recruitment circumstances of all your applicants.

> **Note:** Do not modify the predefined Oracle US Federal HR assignment statuses, because they are used in the Request for Personnel Action process.

Use the Assignment Statuses window.

See: Defining Assignment Statuses, *Workforce Sourcing, Deployment, and Talent Management Guide*

Setup for Employees and Contingent Workers

Step 2: Define Assignment Statuses for Employees and Contingent Workers *Required Step*

With Oracle HRMS you can identify the status of employees and contingent workers in each assignment using your own names.

> **Note:** Do not modify the predefined Oracle US Federal HR assignment statuses, because they are used in the Request for Personnel Action process.

Use the Assignment Statuses window.

See: Defining Assignment Statuses, *Workforce Sourcing, Deployment, and Talent Management Guide*

Special Personal Information (Personal Analysis Key Flexfield Structures)

Use the Personal Analysis Key Flexfield to record special personal information that is not included as standard information. You define each type of information as a

separate *Structure* of the flexfield. For example, you might set up a structure to hold medical information.

This flexfield is used in the following areas:

- Special Information details for People
- Matching requirements for Jobs and Positions

You need to design a Personal Analysis Flexfield Structure for each Special Information Type you want to hold in Oracle HRMS. For each structure you must include the following:

- The Structure Name and the number of Segments.
- The Flexfield Segment Names, Order and Validation Options.
- The Flexfield Value Sets to be used and any lists of values.

Defining the Flexfield Structure is a task for your System Administrator.

> **Note:** You cannot use the *Create Key Flexfield Database Items* process to create database items for the segments of your Personal Analysis Flexfield structures.

Step 3: Run the Create Federal HR Special Info Types Process

The application provides a concurrent manager process that configures the Personal Analysis key flexfield to store Oracle US Federal HR-related information and associates it to the People window. The Special Information stores information for education, conditions of employment, conduct performance, language, performance appraisal, and special consideration.

Use the Submit Request Window.

See: Running the Federal Special Information Types Process, *Use the Personal Analysis Key Flexfield to record special personal information that is not included as standard information. You define each type of information as a separate <emphasis role="italic">Structure</emphasis> of the flexfield. For example, you might set up a structure to hold medical information*

Step 4: Define Personal Analysis Flexfield Value Sets

If you want to validate the values which a user can enter for any segment you must define a specific Value Set.

The attributes of the Value Set will control the type of values that can be entered, and how many characters each segment can hold. The attributes of the Value Set will also control how the values are to be validated.

Value Sets can be shared by different segments of the same flexfield, or by segments of any other flexfield.

Use the Value Set window.

See: Defining Value Sets, *Oracle Applications Flexfields Guide*

Step 5: Define Personal Analysis Flexfield Segments

Define a structure for your Personal Analysis Flexfield which contains the segments you want to use. You will use this structure to enter details in the Special Information Types window.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to enter new details in the Special Information Types window.

> **Note:** You do not need to use a Value Set to validate a segment. If you do not specify a Value Set then a user can enter any alphanumeric value up to a limit of 150 characters.

Use the Key Flexfield Segments window.

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 6: Define Personal Analysis Flexfield Segment Values

If you have chosen Independent or Dependent validation for a Value Set used by a Personal Analysis Flexfield Segment, you must define your list of valid values for the Value Set.

Use the Segment Values window.

See: Defining Segment Values, *Oracle Applications Flexfields Guide*

Step 7: Define Personal Analysis Flexfield Cross Validation Rules

Define any Cross Validation Rules you want to use to control the combinations of segment values which a user can enter.

You define Rules to *Include* or *Exclude* combinations of segment values. For each segment, you can define a *Low to High* range of values.

Use the Cross-Validation Rule window

See: Defining Cross-Validation Rules, *Oracle Applications Flexfields Guide*

Step 8: Define Personal Analysis Flexfield Aliases

Define Aliases for common combinations of segment values if you want to provide these as default options.

Use the Shorthand Aliases window

See: Defining Shorthand Aliases, *Oracle Applications Flexfields Guide*

Step 9: Freeze and Compile Your Personal Analysis Flexfield Structure

You are now ready to freeze your flexfield definition. Navigate to the Define Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. Oracle Human Resource Management Systems now freezes and compiles your Personal Analysis Flexfield definition. Compiling the flexfield definition enables the flexfield window with the defaults, values and rules that you have defined.

Use the Key Flexfield Segments window

See: Defining Key Flexfield Structures, *Oracle Applications Flexfields Guide*

Step 10: Register Special Information Types for the Business Group

After you have defined your Personal Analysis Flexfield Structures you must link them to your Business Group.

You do this using your view-all responsibility.

- Select each Information Type you want to use in this Business Group.
- Select the categories for each type.

- *Job* for Job Requirements
- *Position* for Position Requirements
- *Skills* for use with Oracle Training Administration
- *Other* for use with Person Special Information
- *OSHA* for a special information type set up to record information about employees' work-related injuries or illness

> **Tip:** If you do not check the *Other* category, you cannot use the type to hold information for a person. This means that you could also use the Special Information Types to hold any type of information for a Job or a Position only.

Use the Special Information Types window.

See: Enabling Special Information Types, *Workforce Sourcing, Deployment, and Talent Management Guide*

Requirements Matching

If you have decided to set up competencies, you can enter these as requirements for jobs and positions and match them against people's competence profiles.

If you have other job and position requirements that you want to record, but not define as competencies, you can set them up using the Personal Analysis key flexfield. You can set up each type of requirement as a Special Information Type, which is one instance of the flexfield.

For each Special Information Type, you can also choose whether to enable entry of information for people so that you can match people against the job or position requirements. A standard report (Skills Matching) has been provided to match the requirements of a job and the Special Information details of people in the system.

> **Important:** US users: If you want to include essential job or position requirements in your ADA reporting, make sure you have entered these requirements for your jobs or positions.

Step 11: Define Requirements for Jobs *Optional Step*

You can define the attributes required by any employee who is assigned to a job. These attributes may be Essential or Desirable.

Definitions of requirements can use the same personal analysis flexfield structures and segments you have defined for special personal information.

Use the Job window.

See: Entering Job and Position Requirements, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 12: Define Requirements for Positions *Optional Step*

After you define positions in your enterprise, you can define the attributes required by any employee assigned to that position. These attributes may be Essential or Desirable. The requirements are based on the same personal analysis flexfield structures you have defined for special personal information.

Use the Position window.

See: Entering Job and Position Requirements, *Oracle HRMS Enterprise and Workforce Management Guide*

Create Restricted RPAS

The product comes with a standard RPA and a restricted version, Oracle Federal Restricted Request for Personnel Action. The standard unrestricted form includes all the fields that the Personnelist can access. The restricted form limits the data fields provided and masks commonly restricted data items, such as the social security number and date of birth.

Step 13: Create a new Restricted RPA form

Adding a new form is a task for the application administrator.

Use the Lookup Type window, GHR_US_RESTRICTED_FORM.

See: Creating a Restricted RPA, *Workforce Sourcing, Deployment, and Talent Management Guide*

Step 14: Add fields to the Restricted RPA form

Restricted forms don't change the underlying process methods, only the view of the data. You can limit users' access and view of specific fields and data items. Fields can be coded as Non Display or Display Only.

Use the Restricted Form Process Methods window.

See: Creating a Restricted RPA, *Workforce Sourcing, Deployment, and Talent Management Guide*

Talent Management

Competencies and Qualifications

If you are developing the competence approach as part of your performance management system, you must identify your enterprise's strategic business goals or objectives you want the competence approach to address. You can then set up your methods of measurement, create your competencies and create your assessment and appraisal templates.

If you are using Oracle Self-Service Human Resources to provide self-service human resource management for managers and employees, you also need to perform additional implementation steps.

See: Implementation Steps for Self-Service HR (SSHR), *Oracle HRMS Deploy Self-Service Capability Guide*

Step 1: Define HR:Global Competence Flex Structure Profile Option *Optional Step*

Define the competence key flexfield structure to be used when creating global competencies. If you do not have a value in this field then you will not be able to create global competencies.

Use the System Profile Values window.

See: User Profiles, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 2: Create Rating Scales *Optional Step*

Create rating scales if you want to describe your enterprise's competencies in a general way.

Use the Rating Scales window.

See: Creating a Rating Scale, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 3: Create Competencies *Optional Step*

Create competencies that best meet the needs of your enterprise. If you are using the individual method, you need to set up the proficiency levels for each competence you create.

For a unit standard competence, you enter its Qualifications Framework details, identify the qualifications to which achievement of the competence can contribute, and specify its outcomes and assessment criteria. You must also set the Cluster field to Unit Standard.

Use the Competencies window.

See: Creating a Competence, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 4: Create Competence Types *Optional Step*

You might want to group related competencies together, for example, for advertising a vacancy, or for reporting purposes.

Create the competence types you require using the Lookup COMPETENCE_TYPE.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 5: Group Competencies into Types *Optional Step*

You now need to group related competencies together.

Use the Competence Types window.

See: Grouping Competencies into Types, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 6: Define Competence Requirements *Optional Step*

To ensure your enterprise meets its current and future goals, you will need to define your competence requirements.

Use the Competence Requirements window.

Defining Competency Requirements - Core or Generic Competencies, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Defining Competency Requirements - No Core Competencies, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 7: Enter Work Choices for a Job or Position *Optional Step*

You can enter work choices that can affect an employee's, applicant's, contractor's, or ex-employee's capacity to be deployed within your enterprise (or a customer's). Work Choices include willingness to travel, willingness to relocate, and preferred working

hours and work schedule. You can enter work choices for a job or position, and compare these with the personal work choices entered for people.

Use the Work Choices window.

See: Entering Work Choices for a Job or Position, *Oracle HRMS Enterprise and Workforce Management Guide*.

Step 8: Define Functions (to Implement the Competence Approach in OLM) *Optional Step*

If you have Oracle Human Resources and Oracle Learning Management (OLM) installed in your enterprise, you can hold the qualifications, attributes, and knowledge that students can expect to attain by attending training courses as competencies, skills or a mixture of both (competencies and skills).

You can use parameters to enable selected users to add competencies gained through a course directly to a learner's Competence Profile.

Use the Form Functions window.

See: Form Functions Window, *Oracle Applications System Administrator's Guide*

Step 9: Create Qualification Types *Optional Step*

You can enter all the qualification types your enterprise recognizes.

For Qualifications Framework qualifications, you record the Qualifications Framework details and identify the unit standard competencies that lead to the qualification.

Use the Qualification Types window.

See: Creating Qualification Types, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 10: Create Schools and Colleges *Optional Step*

You need to create schools and colleges that deliver the qualifications your enterprise recognizes. These are then used to record where a person gained the qualification. If you have not automatically loaded these schools and colleges into Oracle Human Resources, you can enter them manually.

> **Note:** Schools and colleges you enter are available to all business groups you create, therefore only load or enter them once.

Use the Schools and Colleges window.

See: Creating Schools and Colleges, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Evaluations and Appraisals

Step 11: Implement Oracle Self-Service Human Resources (SSHR) *Optional Step*

You must also perform other SSHR implementation tasks, such as configuring SSHR web processes using Oracle Workflow, before you can create your appraisal and assessment templates.

See: Implementation Steps for Self-Service HR (SSHR), *Oracle HRMS Deploy Self-Service Capability Guide*.

Step 12: Create an Assessment Template *Optional Step*

You can create assessment templates for all the different evaluations your enterprise performs.

Use the Assessment Template window.

See: Creating an Assessment Template, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 13: Create an Appraisal Template *Optional Step*

You can create appraisal templates to provide instructions to appraisers, to identify which questions belong to which appraisal and to identify which performance rating scale to use.

You can use one of the example appraisal templates we provide and modify them to suit your own needs, or you can create your own.

Use the Appraisal Template window.

See: Creating or Changing an Appraisal Template, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Career and Succession Planning

The flexibility provided by Oracle Human Resources means you can handle your enterprise's career and succession plans using one of a number of models. Which model you decide to use depends upon whether your enterprise's career and succession planning is based upon jobs or positions, and whether your enterprise is using a Windows interface only, or a mixture of the Web and Windows.

Career Paths show the progression paths which are available within your enterprise. You can map out career paths for both jobs and positions.

By planning successors for jobs and positions you always have a shortlist of qualified candidates. You can also identify training and development needs to prepare an employee for a job or position and model different succession options.

Model Career and Succession Plans Based on Jobs (Option 1)

If your enterprise's career and succession planning is based upon jobs, you can use career paths to show possible progressions to one job from any number of other jobs.

> **Important:** In the US, for AAP-Workforce Analysis reporting use the career path functionality to build the *lines of progression* for the jobs included in your AAP plans.

Use the Career Path Names and Map Career Paths windows.

See: Defining Career Paths, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 14: Create and Map Career Paths *Optional Step*

Career paths are based on the structures of your enterprise rather than the people you employ. You may also want to record personal aspirations and progression paths for individual employees. There are several ways to do this.

Use the Career Path Names and Map Career Paths windows.

See: Defining Career Paths, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Step 15: Enter Work Choices *Optional Step*

You can use work choices to help identify a person's career plan.

Use the Work Choices windows.

See: Entering Work Choices for a Job or Position, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Model Career and Succession Plans Based on Positions (Option 2)

If your enterprise's career and succession planning is based upon positions, you can create additional position hierarchies to show any type of progression. These might represent existing line management structures, or even cut across departmental or job-type boundaries.

Step 16: Create Position Hierarchies *Optional Step*

Optionally, create position hierarchies to show career paths, if you want to show typical career progression.

Use the Position Hierarchy window.

See: Creating a Position Hierarchy, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 17: Use Succession Planning (SSHR with a Line Manager Responsibility) *Optional Step*

If you are using SSHR you can use the Succession Planning function to record one or more next positions for each employee. And create, and rank, a group of qualified employees if a position becomes available.

Use the Succession Planning function in SSHR.

Step 18: Use Suitability Matching (SSHR with a Line Manager Responsibility) *Optional Step*

If you are using SSHR you can use the Suitability Matching function to compare the competence profile of an employee, or employees, with the competency needs of a position.

Use the Suitability Matching function in SSHR.

Step 19: Use Attachments or Special Information Types *Optional Step*

Consider holding succession plan information against people as attachments or using a special information type.

Use the Personal Analysis Key Flexfield.

See: Special Information Types, *Oracle HRMS Workforce Sourcing, Deployment, and Talent Management Guide*

Workforce Intelligence

These implementation steps are required to enable you to view data in the HRMS Discoverer business areas and workbooks. They assume that you already have installed Discoverer. For information on Discoverer installation, see: *Discoverer Administration Guide*.

Discoverer Workbooks

Follow the steps below to implement Workforce Intelligence Discoverer workbook reports. If you do not complete these steps, reports will be available to you, but they will not display data correctly. You need to perform some of these steps periodically, so that the reports reflect changes in your enterprise data. See: Programs to Populate Workforce Intelligence Discoverer Reports, *Configuring, Reporting, and System Administration Guide*

Set Up and Configure Workforce *Required Steps*

Workforce is not necessarily a count of the number of employees within your enterprise. Instead, it is a count based on employee assignments and budget measurement type. Calculations depend *either* on your budget measurement values for assignments, or they use a FastFormula..

Step 1: Set budget measurement values *Optional Step*

Set budget measurement values for each employee assignment within Oracle Human Resources. Reports and performance measures then calculate workforce using the budget values.

If you do not set a budget measurement value for an assignment, and a Business Group default does not exist, the reports and performance measures either calculate workforce using Oracle FastFormula, or they will not include the workforce for an assignment.

Use the Assignment Budget Values window.

See: Entering Assignment Budget Values, *Oracle HRMS Enterprise and Workforce Management Guide*

Step 2: Setup the Workforce FastFormula Templates *Optional Step*

If you want to configure how workforce is counted do not set a budget measurement type and assignment measurement value for an assignment. The reports will then use Oracle FastFormula to calculate workforce.

HRMSi provides two predefined workforce formulas:

- TEMPLATE_HEAD
- TEMPLATE_FTE

Use the Formula window.

See: Configuring Workforce Calculations using Oracle FastFormula, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Set Up a Currency Conversion Rate Type *Required Steps*

Step 3: Enter a Currency Conversion Rate Type *Required Step*

Workforce Intelligence uses the conversion rates set up in the GL Daily Rate window. You can enter a specific conversion rate type for Workforce Intelligence, such as Corporate or Spot.

Use the Oracle Human Resources Table Values window.

See: Entering a Conversion Rate Type, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Set Up and Configure Training Hours *Optional Steps*

Within Oracle Training Administration OTA you can record the duration of a training event using a time period of your choice.

For example, rather than recording an event in hours you might record it in weeks or months. To enable the workbooks to display the number of hours of a training event, a predefined Oracle FastFormula, TEMPLATE_BIS_TRAINING_CONVERT_DURATION, converts your time periods into hours.

OTA is installed with four predefined time periods. If you record the duration of events using these predefined time periods the formula automatically converts them into the following hours:

- D (Day) = 8 Hours
- W (Week) = 40 Hours
- M (Month) = 169 Hours
- Y (Year) = 2028 Hours

> **Note:** You set up time periods in Oracle Training Administration using the Lookup type FREQUENCY.

Step 4: Amend the Default Training Hours *Optional Step*

Amend the FastFormula TEMPLATE_BIS_TRAINING_CONVERT_DURATION if you have set up different time periods using the Lookup type FREQUENCY.

Use the Formula window.

See: Amending the Default Training Hours, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 5: Add Additional Training Time Periods *Optional Step*

Amend the FastFormula TEMPLATE_BIS_TRAINING_CONVERT_DURATION if the number of hours per time period does not match those of your enterprise.

Use the Formula window.

See: Adding Additional Training Time Periods, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Set up Cross Business Group Reporting *Optional Steps*

If users are using a local security profile they will only be able to see information in a specific business group. The business group is defined in the security profile attached to the responsibility.

For Discoverer reports, you may want to enable users to see data that spans business groups.

Step 6: Provide users with a global security profile *Optional Step*

If you want to enable cross business reporting, provide users with a global security profile. A global security profile provides cross business group reporting because it does not specify a business group.

Use the Global Security Profile window.

See: Defining a Security Profile, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Identify the Reporting Organization *Required Steps*

HRMS Discoverer workbooks will not run unless the application can identify an organization to report on. If a workbook cannot identify the reporting organization, it will fail to generate certain report parameter lists and will produce the following error message when displaying the report parameter page:

"No values were found for the parameter Organization whilst attempting to build the list of values. To run this report your system must have the parameter Organization set up. Please contact your system administrator."

The application identifies an organization through a business group. You must therefore ensure that you assign a business group correctly to report users. How you assign the business group depends on which security model you implement and whether you are using a local or global security profile. See the options below.

Step 7: Set the Profile Option HR: Security *Optional Step*

If you have implemented the standard HRMS security model, with a local security profile, you must set the profile option HR: Security to the business group you want to report on.

The reports identify the business group through the profile option HR: Business Group. The application automatically sets this profile option to the value in the HR: Security profile option.

Set up the business group in the profile option HR: Security. Use the System Profile Values window.

See: System Profile Values Window , *Oracle Applications System Administrator's Guide*

Step 8: Set the Profile Option HR: Business Group *Optional Step*

If you have implemented the standard HRMS security model, with a global security profile, the HR: Business Group profile option is not set automatically.

Set the profile option HR: Business Group at responsibility level to the business group that you want to report on.

Use the System Profile Values window.

See: Defining Preferences with User Profile Options, *Oracle Applications System Administrator's Guide*

Step 9: Associate a Business Group with a Security Profile *Optional Step*

If you have implemented the Security Groups Enabled security model, the HR: Business Group profile option is not used. You associate a business group with a security profile.

Use the Assign Security Profile window.

See: Assigning Security Profiles, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Restrict Discoverer Workbook Access *Required Steps*

Step 10: Grant Access to Discoverer Business Areas *Required Step*

Grant access privileges to the Discoverer business area to determine which workbooks users can create or view.

Use Oracle Discoverer Administration Edition.

See: Grant Business Area Access, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Check the Vacancy Status Options *Required Steps*

Step 11: Check the Vacancy Status Options *Required Step*

To ensure the reports that cover vacancies return accurate results, you need to ensure that users close vacancies by using the status of CLOSED. You may have to obsolete an old vacancy status option that results in the status of C.

Use the Application Utilities Lookups window.

See: Check the Vacancy Status Options, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Populate Summary Tables *Required Steps*

To ensure your HRMS Discoverer workbooks run correctly and efficiently, you need to run concurrent programs to populate summary tables with your hierarchy data and workforce measurement values.

For a full discussion of these concurrent programs, and when you need to run them, see: Programs for Populating Workforce Intelligence Discoverer Reports, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 12: Populate the Organization Hierarchy Summary Table *Required Step*

All reports that use organization hierarchy gather information from the Organization Hierarchy Summary table. Populate this summary table with your organization hierarchy data. The table ensures that you are getting the best possible performance from your reports.

To populate the summary table, run the concurrent program HRI Load All Organization Hierarchy Versions.

Use the Submit Requests window.

See: Populating the Organization Hierarchy Summary Table, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 13: Populate the Supervisor Hierarchy History Table *Required Step*

All reports that use supervisor hierarchy gather information from the Supervisor Hierarchy Summary table. Populate this summary table with your supervisor hierarchy data. The table ensures that you are getting the best possible performance from your reports.

To populate the summary table, run the concurrent program HRI Load All Supervisor Hierarchies.

Use the Submit Requests window.

See: Populating the Supervisor Hierarchy History Table, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 14: Populate the Workforce Measurement Value History Table *Required Step*

Many reports use Workforce Measurement Values (WMVs). WMVs currently include headcount and full-time equivalent (FTE) assignment budget values.

Run the concurrent program HRI Load Workforce Measurement Value History to populate the Workforce Measurement Value History table with the WMVs used by your reports.

Use the Submit Requests window.

See: Populating the Workforce Measurement Value History Table, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 15: Populate the Generic Hierarchy Summary Table *Required Step*

Some US specific Discoverer workbooks use a 'Vets, EEO, AAP, OSHA, Multi Work Sites' hierarchy. They require information about the current generic hierarchy.

Run the concurrent program HRI Load All Generic Hierarchy Versions to calculate and collect the required data.

Use the Submit Requests window.

See: Populating the Generic Hierarchy Summary Table, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 16: Collect Organization Hierarchy Structures *Optional Step*

The Organization Rollup – Current folder in the Discoverer End User Layer uses organization hierarchies held in the HRI_ORG_PARAMS and HRI_ORG_PARAM_LIST tables. If you build Discoverer reports using the Organization Rollup – Current folder, you must populate these tables with your organization hierarchies.

To populate the tables, run the concurrent program BIS Load Organization Hierarchy Summary Table.

See: Collecting Organization Hierarchy Structures, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

HR Information Systems

Reports

Step 1: Use Standard Reports or Write New Reports *Optional Step*

A number of standard reports are supplied with Oracle HRMS. These reports have been written using Oracle Reports V.2 and registered as concurrent programs with the Standard Requests Submission (SRS) feature of Oracle Applications.

You can use these Standard Reports or write your own reports and register these as additional reports which users can request from the Submit a New Request window.

UK Payroll Implementation Only

In the UK, P45 and Pay Advice reports supplied with Oracle Payroll are designed for use with preprinted stationery. These reports use two special printer drivers to control the print format.

- **P45** paygbp45.prt
- **Pay Advice** paygbsoe.prt

If your printer does not accept the same control characters as the DEC LN03 printer, you may need to modify the special SRW driver files.

When you install Oracle Payroll the two sample files are stored in the $PAY_TOP/srw directory. You should copy the files to $FND_TOP/$APPLREP and then register them using the *Printer Drivers* window.

Step 2: Register Reports as Concurrent Programs *Optional Step*

After you have written your new reports and saved them in the correct subdirectory, you must register the report as a concurrent program. You also register the parameters which can be submitted with the report. For example, you may have written a report to display personal details and you want to submit employee name to limit the output to include one person at a time.

Use the Concurrent Programs window.

See: Concurrent Programs Window, *Oracle Applications System Administrator's Guide*

Step 3: Define Report Sets *Optional Step*

You can define sets of Reports:

- To restrict user access to specific reports.

  A set of reports can be linked to a Responsibility.

- To simplify requesting a report

  You can run a report set in one request, rather than a request for each report.

Use the Request Set window.

See: Defining Request Sets, *Oracle Applications System Administrator's Guide*

Standard Letter Generation

You can use standard letters in HRMS to help you to manage your enterprise's recruitment or enrollments, for example. You do this by issuing standard letters to applicants or students, triggered by changes in assignment or enrollment status.

Oracle HRMS provides you with three different methods to create standard letters:

- Method 1 Concurrent Processing using Word Processors, page 2-62
- Method 2: Concurrent Processing using Oracle Reports, page 2-64
- Method 3: Create Mail Merge Letters using Web ADI, page 2-64

Method 1 - Concurrent Processing using Word Processors

You can create standard letters using Multimate, WordPerfect or Microsoft Word.

Step 4: Plan Standard Letter Requirements *Optional Step*

You need to identify the database information to include in the letters.

See: Planning Standard Letter Requirements, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 5: Write a SQL*Plus Script *Optional Step*

Oracle HRMS supplies you with SQL*Plus scripts as templates for extracting database information for standard letters. You can copy the SQL*Plus script templates and modify them to create the standard letters you require.

See: Writing a SQL*Plus Script for MultiMate or WordPerfect, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

See: Writing a SQL*Plus Script for Microsoft Word, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 6: Register the SQL*Plus Script *Optional Step*

Register your SQL*Plus program with Oracle HRMS. You register your program so that you can run it as a concurrent program. Name the file PERWP*** (or OTAWP***). You must use this prefix for the system to recognize it as a type of letter.

Use the Concurrent Programs window.

See: Registering the SQL*Plus Script, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 7: Link the SQL*Plus Script to the Letter *Optional Step*

Link your SQL*Plus script with a letter and one or more statuses. In Oracle Human Resources, you can link one or more applicant assignment statuses with each recruitment letter. A request for the letter is then created automatically when an applicant is given an associated assignment status. For example, you can link your standard recruitment rejection letter to the status Rejected so that the letter is triggered when you set an applicant's assignment status to Rejected

Use the Letter window.

See: Linking the SQL*Plus Script with aLetter, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 8: Writing a Skeleton Letter *Optional Step*

Write a skeleton letter using your word processor. Include the appropriate merge codes from the data source for the word processor you are using.

See: Writing a Skeleton Letter, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 9: Requesting Letters *Optional Step*

When you, or other users, set the status for an applicant or enrollment that triggers your standard letters, Oracle HRMS creates a letter request automatically, with the status of Pending. It also adds the applicant's or student's name to the request. You can view the pending request and names through the Request Letter window.

Use the Request Letter window.

See: Requesting Letters/Running the Report, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 10: Merging the Data Files *Optional Step*

You now need to merge the data in the Data File with your skeleton letters.

See: Merging the Data File with the Standard Letter, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Method 2 - Concurrent Processing using Oracle Reports

You can create a report for each letter using Oracle Reports, or another tool of your choice. The report contains the skeleton letter text and Select statements specifying the data to be extracted from the Oracle database.

Step 11: Plan Standard Letter Requirements *Optional Step*

You need to identify the database information to include in the letters.

See: Planning Standard Letter Requirements, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 12: Write and Register the Report *Optional Step*

You now need to write and register the report.

See: Writing and Registering the Report, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 13: Link the Report with a Letter *Optional Step*

You need to link your report with a letter and one or more statuses. In Oracle Human Resources, you can link one or more applicant assignment statuses with each recruitment letter. A request for the letter is then created automatically when an applicant is given an associated assignment status. In Oracle Training Administration, you can link one or more enrollment statuses with each enrollment letter. A request for the letter is then created automatically when an enrollment is given an associated status.

Use the Letter window.

See: Linking the Report With a Letter, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 14: Run the Report *Optional Step*

When you, or other users, set the status for an applicant or enrollment that triggers your standard letters, Oracle HRMS creates a letter request automatically, with the status of Pending. It also adds the applicant's or student's name to the request. You can view the pending request and names through the Request Letter window.

Then, when you change the letter request from Pending to Requested, Oracle HRMS runs the report that you created.

Use the Request Letter window.

See: Registering Letters/Running the Report, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Method 3 Create Mail Merge Letters Using Web ADI

Step 15: Create Mail Merge Letters *Optional Step*

Define Web ADI integrators and layouts and set up template letters.

See Creating Mail Merge Letters Using Web ADI, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

People Management Templates

Step 16: Extend the Checklist Lookup Values *Optional Step*

You can add your own values to the supplied list of checklist items and statuses to be included in a template.

Define values for the CHECKLIST_ITEM and CHECKLIST_STATUS Lookup Types.

Define values for BUDGET_MEASUREMENT_TYPES

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 17: Write Formulas for Templates *Optional Step*

You can use formulas to configure the people management templates in the following ways:

- Template Validation Formula
- Template Information Formula
- People Management Message Formula for the Assignment Field
- People Management Message Formula for the Message Tokens

Use the Formulas window.

See: Writing Formulas for Templates, *Oracle HRMS FastFormula User Guide*

Step 18: Configure Templates *Optional Step*

You can use the People Management Configurator to create templates for your users to use. We recommend that you use one of the supplied templates as a basis for your configured version.

Use the People Management Configurator.

See: Using the People Management Configurator, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 19: Set up Notification Messages *Optional Step*

You can setup additional notification messages to be used with the people management templates.

Use Oracle Workflow

See: Notification Messages Issued from Templates Forms, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Oracle HRMS Configuration

Step 20: Define Elements and Distribution Sets *Optional Step*

Select element classifications or individual elements to define a set. There are three types of set:

- Customization set
- Run set
- Distribution set

Use the Element and Distribution Set window.

See: Defining an Element or Distribution Set, *Oracle HRMS Compensation and Benefits Management Guide*.

Step 21: Define Configured Version of a Window *Optional Step*

Form Customization lets you restrict the types of information a user can access in a specific window.

You can define your own window titles for any window configuration option. Remember that the user guides and the online help use the default window names to identify windows.

You can call the configured window in two ways:

- Define a customized node in a task flow
- Add the customization as an argument to the menu function which calls the window

Use the Form Customization window.

See: Configuring a Window With Customform, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 22: Add Configured Window to a Menu or a Task flow *Optional Step*

You must add your configured windows to a menu or task flow.

See: Adding Configured Windows to a Menu or a Task Flow, *Oracle HRMS Configuring, Reporting, and System Administration Guide*.

Step 23: Restrict Access to Query-Only Mode *Optional Step*

You can restrict access to query-only mode for an individual form.

See: Restricting Access to Query-Only Mode, *Oracle HRMS Configuring, Reporting, and System Administration Guide*.

Step 24: Change the Default National Address Style *Optional Step*

The different national address styles are held and configured in the Personal Address Information descriptive flexfield using the Descriptive Flexfield Segments window. You can change the national address style for any country.

See: Changing Default National Address Styles, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 25: Use Parameters for HRMS Form Functions to Hide Sensitive Data *Optional Step*

You can prevent sensitive data from appearing on the Enter a person window by using parameters for HRMS window functions.

See: Using Parameters for HRMS Form Functions, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Task Flows

A task flow defines the selection of windows you want to use when performing a specific task. These can be arranged in sequence or as branched groups of *Nodes*, and you can include configured windows as nodes in your task flow.

> **Warning:** Do not use apostrophes (') or percent (%) symbols in task flow names or task flow node names.

You can create task flows using:

- Forms, page 2-67
- Workflow, page 2-67

Create Task Flows Using Forms

Step 26: Define Task Flow Nodes *Optional Step*

All of the task flow windows provided with Oracle HRMS have nodes predefined for them. You can define new task flow nodes to provide different versions of these windows. For example, if you wanted to use CustomForm on a specific node in a task flow.

Use the Define Task Flow Nodes window.

See: Defining Task Flow Nodes, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 27: Define Task Flows *Optional Step*

Arrange the nodes of your task flows in sequential or branched groups

Use the Task Flow window.

See: Defining Task Flows, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Create Task Flows Using Workflow

Step 28: Create a Top Level Process *Optional Step*

You must define a top level process for each task flow. The top level process can contain sub processes, but not any other top level processes.

You use the Process Diagrammers within Oracle Workflow to create your task flows. You do this by adding and connecting the windows you want to appear.

You must create a top level process, sub processes are optional.

See: Creating a Top Level Process, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 29: Create Sub Processes *Optional Step*

You can group a logical set of task flow windows into a sub process, which can then be used by several top level processes. This simplifies process modelling. Each sub process can contain other sub processes. There are two rules to note regarding sub processes:

- A sub process cannot be defined as runnable.

- When you use a sub process in another process, you must connect the sub process to the Top Node window.

See: Creating Sub Processes, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 30: Create Button Labels *Optional Step*

You can enter the label you want to appear on the task flowed windows, such as Photo (for the Picture window), and such. Each task flow window activity has an attribute called Button Label. Use this attribute to override the default button label for a window and to define an access key (or keyboard shortcut).

See: Creating Button Labels, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 31: Position Button Display *Optional Step*

You can position the display order of buttons on the window. For example, you might want the first button to display the Picture window.

See: Positioning Button Display, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 32: Identify Windows or Blocks to Display *Optional Step*

If you are creating task flows using the combined People and Assignment window, complete this step, otherwise skip this step.

For most task flow windows, you must display the first block of the window on entry. However, when you use the Combined People and Assignment window in a task flow, you must specify whether to display the People window (or block) or the Assignment window on entry.

See: Identifying Windows or Blocks to Display, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 33: Identify Configured Forms to Include in the Task Flow *Optional Step*

If you have created a configured version of a window, you can use it in the task flow. If not, you can skip this step.

See: Identifying Configured Forms to Include in the Task Flow, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 34: Verify and Save the Workflow *Optional Step*

When you have completed the task flow definition within Oracle Workflow, use the Workflow Verify function to check that your workflow conforms to Oracle Workflow modeling rules. When you have successfully verified the Workflow, save it to the HRMS database.

See: Verifying and Saving the Workflow, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 35: Generate a Task Flow From Oracle Workflow *Optional Step*

After modelling a task flow in Oracle Workflow and saving it to the database, you must generate task flow definitions.

Use the Define Task Flow window.

See: Generating a Task Flow From Oracle Workflow, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Menus

Step 36: Define Menu Functions *Optional Step*

Menus are composed of submenus and functions and all Oracle Applications are supplied with default functions and menus to give you access to all of the available windows.

> **Warning:** You should not modify the default functions and menus supplied with the system. On upgrade, these defaults will be overwritten.
>
> If you want to add window configuration options or task flows you should define your own menus.

Use the Form Functions window.

See: Defining Menu Functions, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 37: Define Menus *Optional Step*

The supplied menus give you access to all of the available submenus. However, a number of seeded functions are not enabled on these menus. You need to add them for the responsibilities that should have access to these functions:

Use the Menus window.

See: Defining Menus, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 38: Disable the Multiple Windows Feature *Optional Step*

In most Oracle Applications, you can open multiple windows from the Navigator window without closing the window you already have open. HRMS, however, does not support Multiform functionality.

> **Important:** You must disable this feature on menu structures that access Oracle HRMS windows.

See: Disabling Multiple Windows, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

User Security

Any system that holds human resource and payroll information must be secured against unauthorized access. To reach employee information you need the correct security clearance.

The responsibility for defining and maintaining the internal security of your system is usually given to your system administrator.

Defining Security for HRMS Users

Defining the access limits of each user is a multi-stage process which defines which records a user can see and which forms and windows they can see and use.

There are two security models to enable you to set up the right type of security for your enterprise:

- Standard HRMS security model

  Set up standard security if your enterprise sets up a different responsibility for each business group.

- Security Groups Enabled security model

  Use Security Groups Enabled security if your enterprise wants to enable many business groups for one responsibility. This type of security is most commonly used by Service Centers.

See Defining Security for HRMS Users, page 2-70

Defining Security for Reporting Users

You can also create reporting users who have read only access to data. This can be useful if you want to permit access to the data from another system.

See: Defining Security for Reporting Users, page 2-73.

Defining Security for HRMS Users (Optional)

Step 39: Set up the Enable Security Groups option for your Security Model

- If you are using Standard HRMS security, ensure that the Enable Security Groups profile option is set to No at site and application level.
- If you are using Security Groups Enabled security, ensure that the Enable Security Groups profile option is set to Yes at the application level.

  Important: Once you have changed to Security Groups Enabled Security you *cannot* revert to the Standard Security model.

Use the System Profiles Value window

See: System Profile Values Window, *Oracle Applications System Administrator's Guide*

Step 40: (Security Groups Enabled Model only) Run the Enable Multiple Security Group Process

If you are using the Security Groups Enabled model, you must run the Enable Multiple Security Group process to set up Oracle HRMS to use security groups.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Step 41: Define a Security Profile

Use the Security Profile window (to give access to a single business group) or the Global Security Profile window (to allow users to access records from more than one business group).

See: Defining a Security Profile, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 42: Ensure Required Functions or Menus are Set Up

This is required for the responsibility. For menu functions calling configured forms or task flows, you must enter a parameter in the Parameter field of the Form Functions window.

See: Defining Menu Functions, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 43: Ensure Required Request Group is Set Up

You can define the groups of standard reports and processes that a user can run from the Submit a New Request window. Every responsibility can have access to one request group.

Use the Request Group window.

See: Defining Menu Functions, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

See: Request Groups Window, *Oracle Applications System Administrator's Guide*

Step 44: Define a Responsibility

You need to define a responsibility.

Use the Responsibilities window.

See: Responsibilities Window, *Oracle Applications System Administrator's Guide*

Step 45: Set the User Profile Option Values for Responsibility

Set the HR User Profile Options for the new responsibility.

You must set up the following:

- HR: User Type

  Use this profile option to limit field access on windows shared between Oracle Human Resources and Oracle Payroll.

- HR:Cross Business Group

  Set this profile option to Yes if you want users to be able to view some information across all business groups in your enterprise.

  For details of the information you can make available to users across business groups, see User Profiles, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

- HR: Security Profile

  - If you are using the *Standard Security model*, enter the security profile for the responsibility. This must be set up at responsibility level, otherwise the default view-all security profile is used. Using Standard HRMS security you can only set up one security profile for a responsibility.

  - If you are using the *Security Groups Enabled security model*, do *not* set up or amend the HR: Security Profile option using the System Profile Values window. To set up or change this profile option use the Assign Security Profile window.

  You can set also set up other User Profile Options.

Use the System Profile Values window.

See: System Profile Values Window, *Oracle Applications System Administrator's Guide*

Step 46: Associate a Responsibility With a Set of Help Files

Oracle Applications Help for HRMS defaults to Global help, but you can associate a responsibility with a set of help files for a localization, such as Canada, US or UK, or for a verticalization such as Oracle Federal HRMS. You do this by setting the user profile Help_Localization_Code.

See: User Profiles, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

In addition to associating a responsibility with a localization or a verticalization you can also specify that a particular responsibility should have access to a configured subset of the localized or verticalized help files.

See: Customizing Oracle Applications Help, *Oracle Applications System Administrator's Guide*

Step 47: Create Usernames and Passwords

- If you are using the *Standard Security model*, you need to create usernames and passwords and link responsibilities to users.
- If you are using the *Security Groups Enabled security model,* you need to create usernames and passwords. Do not link responsibilities and security groups (business groups) to users in the Users window for HRMS; instead, use the HRMS Assign Security Profile window.

  Important: If you do enter a responsibility and security group in this window when using Security Groups Enabled security, you still need to use the Assign Security Profile window, to link your user to a responsibility and security profile. If you do not use the Assign Security Profile window, the default view-all security profile is used and your user will be able to see all records in the business group.

Use the Users window.

See: Users Window, *Oracle Applications System Administrator's Guide*

Step 48: (Security Groups Enabled Model only) Assign Security Profiles

If you are using the Security Groups Enabled model, associate a security profile with a user, responsibility and business group.

Important: You cannot use the HRMS Assign Security Profile window to link responsibilities to users if you are setting up Standard Security.

Use the Assign Security Profile window.

See: Assigning Security Profiles, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 49: Run Security List Maintenance Process (PERSLM)

Oracle HRMS uses the Security List Maintenance process to generate the lists of organizations, positions, payrolls, employees, contingent workers, and applicants that each security profile can access.

Important: When you initiate the Security List Maintenance process you must enter the resubmission interval to run the process every night

You must do this so that the system will automatically update the lists with the data changes you make every day.

If a power or computer failure should disrupt this process, you can initiate it manually from the Submit a New Request window.

When this process has completed successfully you can sign on to the system using the new username and responsibility.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Defining Security for Reporting Users (Optional)

Step 50: Create a New Reporting User Oracle ID

If you want reporting users to have the same restricted access to records as your online users, ask your ORACLE Database Administrator to create a new ORACLE User ID.

Reporting Users have read only access to data. This can be useful if you want to permit access to the data from another system.

Note: You need to inform Reporting Users of their Reporting Username and Password.

Step 51: Register the New Oracle ID

Register the new ORACLE ID with Application Object Library.

Use the Register window.

Step 52: Define a Security Profile

Using a view-all responsibility, you can define security profiles in the Security Profile window.

Use the Security Profile window.

See: Defining a Security Profile, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Step 53: Run Generate Secure User Process (SECGEN)

The Generate Secure User process will grant permissions to the new Reporting User ORACLE ID. Until you run this process, reporting users cannot access Oracle HRMS data using this security profile.

1. Select *Generate Secure User.*
2. In the Parameters window, enter the security profile you created for the ORACLE ID.
3. Submit your request.

   A concurrent request ID appears in the ID field. You can check the progress of your request on the View Concurrent Requests window.

Use the Submit a New Request window

See: Submitting a Request, *Oracle Applications User's Guide*

Web Applications Desktop Integrator (Web ADI)

Step 54: Set Up Web ADI *Optional Step*

You can set up Web Applications Desktop Integrator (Web ADI) to export information from your Oracle HRMS database to desktop applications, for example, spreadsheets.

See Implementing Web ADI for Use with Oracle HRMS, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

See: Upgrade Information for Converting from ADE to Web ADI, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

Audit Requirements

Step 55: Turn on Auditing *Optional Step*

To turn on Auditing, set the AuditTrail:Activate profile option to Yes at Site or Application level.

Use the System Profile Values window.

See: System Profile Values Window, *Oracle Applications System Administrator's Guide*

Turning Audit on has no noticeable effect on the performance of the system and users will not be aware of any extra delay in committing their transactions.

Step 56: Estimate File Sizing and Management Needs *Optional Step*

Whenever you choose to audit the actions of users of the system you are deciding to keep the details of all the transactions which take place. This will include before and after details as well as the details of who made the change and when.

> **Warning:** In normal use the auditing of data can soon generate large volumes of audit data, which even when stored in a compressed format will continue to grow in size until you reach the limits imposed by your environment. If you reach the limits during active use then users will be unable to use the system until you remedy the problem.

You are strongly advised to consider the scope of your audit activities and how you will use the data you accumulate. Also you should consider how often you will report on the audit data, and when you will archive and purge your audit data.

If you need more advice on this you should contact your Oracle Support representative.

Step 57: Define Audit Installations *Optional Step*

If you have installed more than one Oracle Application you can audit across multiple installations. For Oracle HRMS you should enable auditing for the HR user and the APPLSYS user.

Use the Audit Installations window.

See: Audit Installations Window, *Oracle Applications System Administrator's Guide*

Step 58: Define Audit Tables and Columns *Optional Step*

With Oracle Applications you can define the level of detail you want to audit. You define the individual fields of each record that you want to audit.

- Query the Table you want to audit

- Enter the columns you want to audit for that table

Use the Audit Tables window.

See:Audit Tables Window, *Oracle Applications System Administrator's Guide*

Step 59: Define Audit Groups *Optional Step*

You can define one or more Audit Groups for your installation. You might find this useful if you have more than one Oracle Application installed.

Use the Audit Groups window.

See: Audit Groups Window, *Oracle Applications System Administrator's Guide*

Step 60: Run AuditTrail Update Tables Process and AuditTrail Update Datetracked Tables Process *Optional Step*

To start the AuditTrail activity you must submit the *AuditTrail Update Tables Process* for all tables, and the *AuditTrail Update Datetracked Tables Process* for all datetracked tables.

Use the Submit a New Request window.

See: Submitting a Request, *Oracle Applications User's Guide*

Modification of US Federal Workflow

Oracle Federal HR has predefined workflow attributes for several processes. You can customize the attributes for the following Item Types as follows:

- GHR Personnel Action

  If you route an action at least once or save it to your workflow worklist, the application does not send an FYI Notification when it successfully updates an action. (An FYI Notification is a message in your worklist that informs you of an action taken by the application.)

  You can customize the workflow process to have the application send an FYI Notification to the person who performs the Update to HR, or to the person who performs the update to HR and the Approver.

- GHR Within Grade Increase

  The application does not send an FYI Notification to the Personnel Office (POI) worklist when it successfully updates an Automatic Within Grade Increase (WGI) action. You can customize the workflow process to have the application send a Notification to the POI worklist.

  When an Automatic WGI is processed for an employee, the action is sent to the POI worklist for approval. You can customize the workflow process to require a Supervisor's approval before it is sent to the POI for approval.

See: Copying the Original Workflow Item Type, *Configuring, Reporting, and System Administration Guide*

Step 61: Configure GHR Personnel Action: Notify Update HR User

When a personnel action is successfully updated to the HR database, the application does not send a Notification to the person who last updated the action to the HR database. If you want this person to receive an FYI Notification, change the default attribute from No to Yes.

Use Workflow to change this attribute.

See: Changing a Workflow Attribute, *Configuring, Reporting, and System Administration Guide*

Step 62: Configure GHR Personnel Action: Notify Only Update HR User

When a personnel action is successfully updated to the database, the application sends a Notification to the last person who updated the action to the HR database. If you want the Approver to also receive the Notification, change the default attribute from Yes to No.

Use Workflow to change this attribute.

See: Changing a Workflow Attribute, *Configuring, Reporting, and System Administration Guide*

Step 63: Configure GHR Within Grade Increase: Use Personnel Office Only

When an Automatic WGI action is processed, the application sends a Notification to the Personnel Office (POI). If you want the supervisor to receive a Notification so that he or she can approve the action before it's sent to the Personnel Office, change the default attribute from Yes to No.

Use Workflow to change this attribute.

See: Changing a Workflow Attribute, *Configuring, Reporting, and System Administration Guide*

Step 64: Configure GHR Within Grade Increase: Notify Personnel Office (POI) of Update to HR Success

When an Automatic WGI action is successfully updated to the HR database, the application does not send a Notification to the POI. If you want the POI to receive a Notification, change the default attribute from No to Yes.

Use Workflow to change this attribute.

See: Changing a Workflow Attribute, *Configuring, Reporting, and System Administration Guide*

US Federal Workflow Routing

You can route forms such as the Request for Personnel Action and the Position Description to a variety of destinations including individuals, groupboxes, or routing lists. You use the Routing Groups and Groupboxes and Routing Lists maintenance forms to set up your routing groups, groupboxes, and routing lists.

Step 65: Define Your Routing Group

Use the Routing Group and Groupbox Details window or the Routing Group and Routing List Details window.

Arrange your Routing Groups so that users who need to exchange information are part of the same Routing Group. You can assign users to more than one Routing Group, but once an action is initiated within a Routing Group, you can only route it to other members of the same Routing Group, not to a different Routing Group.

See: Setting Up Routing Groups, *Configuring, Reporting, and System Administration Guide*

Step 66: Assign Users to Routing Groups and to Roles within that Routing Group

Roles are designations that describe each member's workflow activities within a Routing Group. You can assign multiple roles to each user in a Routing Group.

Use the People Extra Information window.

See: Adding a User to a Routing Group, *Configuring, Reporting, and System Administration Guide*

Step 67: Set Up Groupboxes and Add Users to Them

Groupboxes are a convenient way to pool work for multiple users, so that any user assigned to the Groupbox can process the action. Use the Routing Group and Groupbox Details window.

See: Setting up Groupboxes, *Configuring, Reporting, and System Administration Guide*

Step 68: Set Up Routing Lists and Add Users to Them

A routing list is a predefined list of routing destinations. The list defines the order in which a person or groupbox receives a workflow notification.

Use the Routing Group and Routing List Details window.

See: Setting up Routing Lists, *Configuring, Reporting, and System Administration Guide*

Define the Personnel Office ID Information *Required Steps*

Step 69: Define a Routing Groupbox

The Personnel Office ID (POI) groupbox is used as a standard groupbox, a central routing point for RPAs, a destination where the application sends actions that have errors (RPAs, Mass Action and Auto WGI processes).

To have the application route personnel actions to the Personnel Office, you need to define a Routing Group and Groupbox for each Personnel Office ID that your agency uses. You must also add one member who will be the Approving Official.

Use the Routing Group and Groupbox Details window.

See: Setting up Routing Groups, Groupboxes, and Routing Lists, *Configuring, Reporting, and System Administration Guide*

Step 70: Complete the Personnel Office Information

For each POI that your agency uses, you must complete the POI groupbox information on the Personnel Office ID Federal Maintenance Form by entering the Groupbox Name and the Approver's Full Name. The application enters the Approving Officer's name, working title, and approval date on the Notifications of Personnel Action for all mass actions. After you set up the groupbox for the Personnel Office ID, you enter the Approving Officer's name and enter the groupbox name that you set up for the Personnel Office.

Use the Personnel Office Identifiers window.

See: Maintaining Personnel Office ID Information, *Configuring, Reporting, and System Administration Guide*

Define a Groupbox *Required Steps*

Step 71: Define a Groupbox for the Workflow Administrator

If there's an error when routing a Within Grade Increase, Position Description, or other personnel action, and the application does not find a designated groupbox, such as a Personnel Office groupbox, it sends the notification to the Workflow Administrator's groupbox.

Use the Routing Group and Groupbox Details window.

See: Setting up a Workflow Administrator's Groupbox, *Configuring, Reporting, and System Administration Guide*

Agency Legal Authority Codes, Remarks, and Insertion Data

If your agency has agency-specific Legal Authority Codes (LACs), Remarks, and accompanying insertion data, you can add them and associate them to Nature of Action Codes (NOACs).

Step 72: Add Agency LACs

You can add new LACs or update existing ones.

Use the Lookup Values window, GHR_US_LEGAL_AUTH_CODE.

See: Defining Legal Authority Codes, *Configuring, Reporting, and System Administration Guide*

Step 73: Associate Agency LACs to NOACs

After adding a LAC, you can associate it to a NOAC. You can associate a single LAC to more than one NOAC, as well as several LACs to one NOAC.

Use the NOA Legal Authorities Federal Maintenance window.

See: Associating Legal Authority Codes to NOACs, *Configuring, Reporting, and System Administration Guide*

Step 74: Add Remark Codes and Descriptions

If your agency maintains agency-specific Remarks, you can add and update their Codes and Descriptions.

Use the Remarks Codes and Descriptions Federal Maintenance window.

See: Adding and Deleting Remarks, *Configuring, Reporting, and System Administration Guide*

Step 75: Associate Remarks and NOACs.

You can associate several Remarks to a single NOAC, as well as a single Remark to several NOACs.

Use the Remark Codes and Descriptions Federal Maintenance window.

See: Associating Remarks to NOACs, *Configuring, Reporting, and System Administration Guide*

Step 76: Enter Insertion Data for Remarks and Legal Authority Codes

The product includes descriptive flexfields for NOAC, Remark, and Legal Authority descriptions. These flexfields have five context-sensitive segments for insertion values that correspond to the underscores in the descriptions.

> **Note:** Underscores represent insertion data only. Make sure that your Remarks, Legal Authority Codes, or NOACs do not contain underscores unless they have corresponding insertion segments.

Adding insertion data is a task for the application administrator.

Use the Descriptive Flexfield Segments window.

See: Insertion Data, *Configuring, Reporting, and System Administration Guide*.

Productivity Event Categories and Codes

Step 77: Set up Event Categories

Define Categories in Lookup type GHR_US_EVENT_CATEGORIES. A category may contain multiple events. Entries include Code, Meaning, and Description.

Use the Lookup Types window.

See: Setting up Event Codes, *Configuring, Reporting, and System Administration Guide*

Step 78: Set up Event Codes

When you route an RPA, the application enters an RPA status in the Routing History. Some actions that you take to process an RPA are external to the routing process, such as obtaining confirmation from another organization. You can record these actions as events. For each event you can enter a Start Date Description, End Date Description, Category Code, Standard completion Time, From Date, and To Date.

Use the Enter and Maintain Event Codes window.

See: Setting up Event Codes, *Configuring, Reporting, and System Administration Guide*

> **Note:** If you need to capture more information related to a category, you can extend this Lookup type by defining other Attributes.

Schedule for US Federal Processes

Step 79: Set the Frequency for Producing Federal Reports

You can determine when and how often the system processes federal reports, such as the Notification of Personnel Action, or the Central Personnel Data File (CPDF) Dynamics and Status reports.

Use the Concurrent Manager Submit Requests window.

See: Reports and Processes in Oracle HR, *Configuring, Reporting, and System Administration Guide*

Step 80: Set the Frequency for Running the Within Grade Increases (WGI) process

The default WGI process automatically determines eligible employees, creates an RPA, and updates a WGI when the employee's WGI Pay Date is reached. Your system administrator can customize the workflow process to require a response from the Personnel Office or the Supervisor:

- Personnel Office receives a notification and no response is required
- Personnel Office receives a notification and a response is required
- Supervisor receives a notification and a response is required. The system then sends the notification to the Personnel Office.

Your system administrator can further customize the WGI eligibility criteria with user hooks. For example, the system administrator might customize the WGI eligibility to include a procedure that checks Rating of Record on the US Government Performance Appraisal flexfield.

The system identifies employees eligible for a WGI 90 days before the WGI Pay Date and generates a future effective WGI Request for Personnel Action. You can schedule the frequency with which the system processes automatic Within Grade Increases.

Use the Concurrent Manager Submit Requests window.

See: Scheduling the Automatic WGI Process, *Compensation and Benefits Management Guide*

Step 81: Set the Frequency for Processing Future Actions

You can process and update Requests for Personnel Action that have a future effective date. When the effective date is reached, the system performs a final update and generates a Notification of Personnel Action. You can set the frequency with which the system processes all RPAs that have come due.

Use the Concurrent Manager Submit Requests window.

See: Processing Future Actions, *Workforce Sourcing, Deployment, and Talent Management Guide*

Technical Essays

DateTrack

How DateTrack Works

DateTrack adds the dimension of time to an application's database. The value of a DateTracked record depends on the date from which you are viewing the data. For example, querying an employee's annual salary with an effective date of 12-JUL-1992 might give a different value than a query with an effective date of 01-DEC-1992. However, the application and the user see the employee's pay as a single record.

Behavior of DateTracked Forms

This section describes the behavior of forms that incorporate DateTracking.

When you begin to update or delete a record on a DateTracked form, you are prompted with a number of choices. This section describes the choices and their effect on the DateTracked table.

The term "today" refers to the effective date set by the user.

Update

When a user first alters a field in a DateTracked block in the current Commit unit, he or she sees a choice of Update prompts as follows:

- UPDATE - Updated values are written to the database as a new row, effective from today until 31-DEC-4712. The old values remain effective up to and including yesterday.
- CORRECTION - The updated values override the old record values and inherit the same effective dates.

If the user selects UPDATE, DateTrack checks whether the record being updated starts today. If it does, a message warns that the previous values will be lost (because DateTrack can only store information on a day by day basis). DateTrack then changes the mode for that record to CORRECTION.

Next, if UPDATE was selected, DateTrack checks whether the record being updated has already had future updates entered. If it has been updated in the future, the user is further prompted for the type of update, as follows:

- UPDATE_CHANGE_INSERT (Insert) - The changes that the user makes remain in effect until the effective end date of the current record. At that point the future scheduled changes take effect.
- UPDATE_OVERRIDE (Replace) - The user's changes take effect from now until the end date of the last record in the future. All future dated changes are deleted.

In most forms, users are prompted for the update mode for *each* record they update. In some forms, they are asked for the update mode for only the *first* record they update. Any other rows updated take the same update mode. Users are not prompted again, until they have committed or cleared any outstanding changes.

Delete

When deleting a record, the user is prompted for the type of delete. There are four options, as follows:

- DELETE (End Date) - This is the DateTracked delete. The record that the user is currently viewing has its effective end date set to today's date. The record disappears from the form although the user can requery it.
- ZAP (Purge) - This is the total delete. All records matching the key value, whatever their date stamps, are deleted.
- FUTURE CHANGE (All) - This choice causes any future dated changes to the current record, including a future DateTracked delete, to be removed. The current record has its effective end date set to 31-DEC-4712.

  The record can again be displayed by requerying.
- DELETE NEXT CHANGE (Next Change) - This choice causes the *next* change to the current DateTracked record to be removed.

  Where another future dated DateTracked row exists for this record, it is removed and the current row has its effective end date set to the effective end date of the deleted row.

  Where no future DateTracked row exists, but the current row has an end date other than 31-DEC-4712, then this option causes the effective end date to be set to 31-DEC-4712. This means that a date effective end is considered to be a change.

  Notice that this option again removes the current row from the form, though it can be displayed again by requerying.

Insert

The user is not prompted for any modes when inserting a record. The effective start date is always set to today (Effective Date). The effective end date is set as late as possible. Usually this is 31-DEC-4712, although it can be earlier especially when the record has a parent DateTracked record.

Table Structure for DateTracked Tables

A DateTracked (DT) record is what the application and the user see: a single DT record for each key value. However, this DT record may change over time, so it may correspond to one or more physical rows in the database. The history for the record is held by storing a row when the record is created, and an extra row every time the record changes. To control these rows, every DateTracked table must include these columns:

EFFECTIVE_START_DATE DATE NOT NULL

EFFECTIVE_END_DATE DATE NOT NULL

The effective start date indicates when the record was inserted. The effective end date indicates when the record was deleted or updated. A deleted record has the highest end date of all the rows with that key, but for an updated record there will be at least one row for this key with a higher effective end date.

As time support is not provided, the effective start date commences at 0000 hours and the effective end date finishes at 2359 hours. This means that a DT record can change at most once per day.

Example

Table Showing Example of DateTracked Table Contents

EMPID	EMPNAME	SALARY	EFFECTIVE_ START_DATE	EFFECTIVE_ END_DATE
3203	SMITH	17,000	12-MAR-1989	19-JUL-1989
3203	SMITH	18,200	20-JUL-1989	20-JUL-1989
3203	SMITH	18,400	21-JUL-1989	01-DEC-1989

The table above shows the physical table after the user has done the following:

- Set the effective date to 12-MAR-1989. Inserted record for SMITH.
- Set the effective date to 20-JUL-1989. Updated SMITH record with new salary.
- Set the effective date to 21-JUL-1989. Again updated SMITH record with new salary.
- Set the effective date to 1-DEC-1989. Deleted record for SMITH.

The table below shows what the user sees on querying the SMITH record at different effective dates.

Table of Example Query Results for a DateTracked Table

EFFECTIVE DATE	EMPID	EMPNAME	SALARY
11-MAR-1989	** no rows retrieved		
12-JUN-1989	3203	SMITH	17,000
21-JUL-1989	3203	SMITH	18,400
02-DEC-1989	** no rows retrieved		

Because the primary key column in the table is no longer unique, any indexes on the table that included the primary key column must now also include the EFFECTIVE_START_DATE and EFFECTIVE_END_DATE columns.

List of DateTracked Tables

To get a list of the DateTracked tables used in Oracle Human Resources, select from the data dictionary where the table name is like Application Short Name%F. Substitute in the HRMS application short code you are interested in (such as PER or BEN).

For each of the DateTracked tables there is a DateTracked view called <TABLE NAME> and a synonym pointing to the full table called <TABLE NAME_F>.

Creating a DateTracked Table and View

The previous section described the table structure of a DateTracked table. This section describes the steps to go through to create a DateTracked table and view.

You must use the following nomenclature for DateTracked tables:

Base table: <TABLE NAME_F>

DateTracked view: <TABLE NAME>

In addition to the DateTracked view, there is another view that shows the rows in the table as of SYSDATE. The name of this view is derived by replacing the _F at the end of the table name by _X.

Example

To incorporate DateTrack on to an existing table called EMPLOYEES, follow these steps:

1. Create a new table called EMPLOYEES_F that is identical to EMPLOYEES but with the columns EFFECTIVE_START_DATE and EFFECTIVE_END_DATE added. Normally you would set the EFFECTIVE_START_DATE and EFFECTIVE_END_DATE columns to the maximum range.

```
CREATE TABLE EMPLOYEES_F AS

SELECT EMPLOYEES.*,

TO_DATE('01-01-0001','DD-MON-YYYY') EFFECTIVE_START_DATE,

TO_DATE('31-12-4712','DD-MON-YYYY') EFFECTIVE_END_DATE

FROM EMPLOYEES;

ALTER TABLE EMPLOYEES_F

MODIFY (EFFECTIVE_START_DATE NOT NULL,

EFFECTIVE_END_DATE  NOT NULL);
```

   Remove the old table.

```
DROP TABLE EMPLOYEES
```

   If the old table already has the two new columns, just rename it.

```
RENAME EMPLOYEES TO EMPLOYEES_F;
```

2. Create the New Unique Indexes of the DateTracked Table by dropping the old indexes, creating the new unique indexes as old unique index + EFFECTIVE_START_DATE + EFFECTIVE_END_DATE, and creating the new non-unique indexes the same as the old non-unique indexes.

3. Create a DateTracked view called EMPLOYEES. This view uses the entry in FND_SESSIONS for the current user effective id for the effective date.

```
CREATE VIEW EMPLOYEES AS

SELECT *

FROM EMPLOYEES_F

WHERE EFFECTIVE_START_DATE <=

(SELECT EFFECTIVE_DATE

FROM   FND_SESSIONS

WHERE FND_SESSIONS.SESSION_ID = USERENV('SESSIONID'))

AND EFFECTIVE_END_DATE >=

(SELECT EFFECTIVE_DATE

FROM  FND_SESSIONS

WHERE FND_SESSIONS.SESSION_ID = USERENV('SESSIONID'))
```

4. To create the view EMPLOYEES_X based on the table EMPLOYEES_F, use the following SQL:

```
CREATE  VIEW EMPLOYEES_X AS

SELECT *

FROM EMPLOYEES_F

WHERE EFFECTIVE_START_DATE <= SYSDATE

AND EFFECTIVE_END_DATE  >= SYSDATE
```

Restricting Datetrack Options Available to Forms Users

When a user edits or deletes a datetracked record, the system displays a window asking the user what type of update or deletion to perfom. Before it displays this window, the system calls a custom library event (called DT_SELECT_MODE). It passes in the list of buttons that DateTrack would normally display (such as Update and Correction).

Your custom code can restrict the buttons displayed. If necessary, it can require that the user is given no update or delete options, and receives an error message instead. However, it cannot display buttons that DateTrack would not normally display for the entity, effective date, and operation the user is performing.

If the user chooses Update and future changes exist, the custom library event point may be executed a second time so your custom code can determine whether the user is given the two update options: Insert and Replace.

Global Variables

The following global variables can be used at the DT_SELECT_MODE event. They are not available at any other CUSTOM library event.

Table of Global Variables at DT_SELECT_MODE Event

Global Variable Name	Read/Write	Description
g_dt_update	Read and write	Set to TRUE when the product would normally display the Update button on the mode selection window. Otherwise set to FALSE.
g_dt_correction	Read and write	Set to TRUE when the product would normally display the Correction button on the mode selection window. Otherwise set to FALSE.
g_dt_update_change_insert	Read and write	Set to TRUE when the product would normally display the Insert button on the mode selection window. Otherwise set to FALSE.
g_dt_update_override	Read and write	Set to TRUE when the product would normally display the Replace button on the mode selection window. Otherwise set to FALSE.
g_dt_zap	Read and write	Set to TRUE when the product would normally display the Purge button on the mode selection window. Otherwise set to FALSE.
g_dt_delete	Read and write	Set to TRUE when the product would normally display the End Date button on the mode selection window. Otherwise set to FALSE.
g_dt_future_change	Read and write	Set to TRUE when the product would normally display the All button on the mode selection window. Otherwise set to FALSE.
g_dt_delete_next_change	Read and write	Set to TRUE when the product would normally display the Next button on the mode selection window. Otherwise set to FALSE.

Important: Custom code can change a TRUE value to FALSE. However, if it tries to change a FALSE value to TRUE, the system ignores this change.

Enabling the DT_SELECT_MODE Event

To enable the DT_SELECT_MODE event, add the following code to the STYLE procedure in the CUSTOM package, CUSTOM library:

```
if event_name = 'DT_SELECT_MODE' then

 return custom.after;

else

 return custom.standard;

end if;
```

Example Custom Code

Suppose you wanted to stop the Delete mode button from being displayed on the Mode Selection window when DateTrack would normally make it available. You could add the following code to the EVENT procedure in the CUSTOM package, CUSTOM library:

```
if (event_name = 'DT_SELECT_MODE') then

 if name_in('GLOBAL.G_DT_DELETE') = 'TRUE' then

  copy('FALSE', 'GLOBAL.G_DT_DELETE');

 end if;

end if;
```

Create and Modify DateTrack History Views

DateTrack History is available in most windows where you can enter date tracked information. DateTrack History enables you to track changes made to records and fields, and by whom. You can select the fields you want to focus on and view the changed values in those fields over time.

DateTrack History is available from a button on the toolbar.

What Can You Create and Modify?

You can create new views or modify existing views to customize the information displayed. You can:

- Create a view to join to other tables. This enables you to use a meaningful table name as a column header. By contrast, the base table can only display an ID of another table.
- Determine the fields to display, by modifying the views.
- Modify views to display column names aliases for the meaningful names you have defined for descriptive flexfield segments.
- Determine which view to use dependent on criteria of your choice, such as the Business Group ID.

What Happens When You Request DateTrack History?

When you request DateTrack History, Oracle HRMS extracts the information from one of three sources. The application looks first for the alternative view specified by the custom

library and if one exists, extracts the information from there. If there isn't an alternative view specified, it looks next for a default DateTrack History view from which to extract the information, and if that doesn't exist, it extracts the information from the base table. It then displays the information in the DateTrack History Change Field Summary window.

The name of the default DateTrack History view is the same as that of the base table, except that the suffix _F is replaced by _D. For example, if the base table is PER_ALL_PEOPLE_F, the application looks for a view called PER_ALL_PEOPLE_D.

> **Note:** It is possible to define more than one History view for each datetracked table, so there might be examples where the History view name does not follow this naming convention.

When a view exists, the application reads the information about the entity name and column prompts from the DateTrack tables:

- DT_TITLE_PROMPTS_TL
- DT_DATE_PROMPTS_TL
- DT_COLUMN_PROMPTS_TL

If the column information is not available in the DT_COLUMN_PROMPTS_TL table, the information is obtained from the view definition. The DateTrack History code modifies the column names of the table or view before presenting them. Underscores are replaced by spaces and the first letter of each word appears in upper case.

Rules for Creating or Modifying DateTrack History Views

DateTrack History views should have the same name as the corresponding base table, wherever possible, except that the suffix _F is replaced by _D. If you are using custom library to specify an alternative view, the view name is different, but you should still use the _D suffix.

All views must contain the following columns:

- The primary key of the base table
- The effective start date of the base table
- The effective end date of the base table
- The last updated date column
- The last updated by column (obtain the actual user name by an outer join to FND_USER_VIEW).

> **Note:** There is a limit of 35 columns in Date Track History views. The primary key, effective start date, and effective end date columns must be present in the view but cannot be seen in the DateTrack History windows.

Do not edit the supplied DateTrack History view creation scripts. If you want to customize the supplied DateTrack History views, copy the scripts and modify the copies instead. After an upgrade, you should check that your customizations are consistent with the new views supplied with the upgrade. If so, you can rerun your customized view creation scripts to recreate your customized views.

Update Folder Definitions When Adding Columns

Adding an additional column to DateTrack History views can affect the column order, and if you have previously saved folders, the data displayed and the prompts might no longer match up. This is because the Date Track History Change Field Summary window displays the column names in alphabetical order, but with the effective date values in the first two columns.

We recommend that you update any folder definitions straight after you apply the new view to the database, otherwise the data displayed and the prompts in folders might not match up in future.

Example of a DateTrack History View

In this example, the base table is PAY_GRADE_RULES_F.

```
create or replace view pay_grade_rules_d
 (grade_rule_id,
  effective_start_date,
  effective_end_date,
  maximum,
  mid_value,
  minimum,
  grade,
  rate_type,
  last_update_date,
  last_updated_by)
 AS
 select GRULE.grade_rule_id,
        GRULE.effective_start_date,
        GRULE.effective_end_date,
        GRULE.maximum,
        GRULE.mid_value,
        GRULE.minimum,
        GRADE.name,
        HR1.meaning,
        GRULE.last_update_date,
        FUSER.user_name
 from   pay_grade_rules_f         GRULE
 ,      per_grades                GRADE
 ,      hr_lookups                HR1
 ,      fnd_user_view             FUSER
 where  GRADE.grade_id          = GRULE.grade_or_spinal_point_id
and    HR1.lookup_code          (+)= GRULE.rate_type
and    HR1.lookup_type          (+)= 'RATE_TYPE'
and    FUSER.user_id            (+)= GRULE.last_updated_by
```

Using Alternative DateTrack History Views

Before the DateTrack History Change Field Summary window displays, the system calls a custom library event (called DT_ CALL_HISTORY). It passes in details of the current record and which DateTrack view the product normally uses. You can write custom code to change the name of the view DateTrack History should use. Your code can include IF statements that determine which view to use in different circumstances.

> **Note:** It is your responsibility to ensure that the alternative view exists in your database and the relevant users have select access to it.

For each additional view, you need to insert extra rows into the DT_TITLE_PROMPTS_TL and DT_COLUMN_PROMPTS_TL tables, based on the view name. Use SQL*Plus scripts to maintain the extra table contents and view definitions.

Global Variables

The following global variables can used at the DT_CALL_HISTORY event. They are not available at any other CUSTOM library event.

Table of Global Variables

Global Variable Name	Read/Write	Description
g_dt_basetable	Read only	Name of the database table where the data is held. For example: PER_ALL_PEO PLE_F
g_dt_uidfield	Read only	Name of the surrogate ID on the database table. For example: PERSON_ID
g_dt_uidvalue	Read only	The surrogate ID value for the current record.
g_dt_alternative_history_ view	Read and Write	Usually DateTrack History queries the history data from a database view that has the same name as the database table, except the _F suffix is changed to _D. In that case this global variable is null. For example when the database table is PER_ALL_PEOPLE_ F, the PER_ALL_PEOPLE_D view is used. If you want to use a different view, set this global variable to the actual view name (even if the variable is initially null).

Enabling the DT_ CALL_HISTORY Event

To enable the DT_CALL_HISTORY event add the following code to the STYLE procedure in the CUSTOM package, CUSTOM library:

```
if event_name = 'DT_CALL_HISTORY' then

  return custom.after;

else

  return custom.standard;

end if;
```

Example Custom Code

Suppose you want to use a different view whenever the standard product would normally use the PER_ALL_PEOPLE_D view. Add the following code to the EVENT procedure in the CUSTOM package, CUSTOM library:

```
if (event_name = 'DT_CALL_HISTORY') then

  if name_in('global.g_dt_basetable') = 'PER_ALL_PEOPLE_F' then

    copy

     ('NAME_OF_OTHER_VIEW'

     ,'global.g_dt_alternative_history_view'

     );

  end if;

end if;
```

List of DateTrack History Views

The supplied views and view creation scripts are as follows:

Table of DateTrack History Views

View Name	Based on (table)	View Creation Script
BEN_BENEFIT_CONTRIBUTIONS_D	BEN_BENEFIT_CONTRIBUTIONS_F	pedtbbcf.sql
HXT_ADD_ASSIGN_INFO_D	HXT_ADD_ASSIGN_INFO_F	hxtdtaas.sql
HXT_ADD_ELEM_INFO_D	HXT_ADD_ELEM_INFO_F	hxtdtael.sql
HXT_SUM_HOURS_WORKED_D	HXT_SUM_HOURS_WORKED_F	hxtdtsum.sql
HXT_TIMECARDS_D	HXT_TIMECARDS_F	hxtdttim.sql
PAY_ALL_PAYROLLS_D	PAY_ALL_PAYROLLS_F	pydtpayr.sql
PAY_BALANCE_FEEDS_D	PAY_BALANCE_FEEDS_F	pydtbalf.sql
PAY_CA_EMP_FED_TAX_INFO_D	PAY_CA_EMP_FED_TAX_INFO_F	pycadtfd.sql
PAY_CA_EMP_PROV_TAX_INFO_D	PAY_CA_EMP_PROV_TAX_INFO_F	pycadtpv.sql
PAY_COST_ALLOCATIONS_D	PAY_COST_ALLOCATIONS_F	pydtpcst.sql
PAY_ELEMENT_LINKS_D	PAY_ELEMENT_LINKS_F	pydtelin.sql

View Name	Based on (table)	View Creation Script
PAY_ELEMENT_TYPES_D	PAY_ELEMENT_TYPES_F	pydtetyp.sql
PAY_FORMULA_RESULT_ RULES_D	PAY_FORMULA_RESULT_ RULES_F	pydtfmrr.sql
PAY_GRADE_RULES_D	PAY_GRADE_RULES_F	pydtgrdt.sql
PAY_INPUT_VALUES_D	PAY_INPUT_VALUES_F	pydtinpv.sql
PAY_LINK_INPUT_VALUES_ D	PAY_LINK_INPUT_VALIES_ F	pydtliiv.sql
PAY_ORG_PAYMENT_ METHODS_D	PAY_ORG_PAYMENT_ METHODS_F	pydtpaym.sql
PAY_PERSONAL_ PAYMENT_METHODS_D	PAY_PERSONAL_ PAYMENT_METHODS_F	pydtppym.sql
PAY_STATUS_PROCESSING_ RULES_D	PAY_STATUS_PROCESSING_ RULES_F	pydtstpr.sql
PAY_USER_COLUMN_ INSTANCES_D	PAY_USER_COLUMN_ INSTANCES_F	pydtucin.sql
PAY_USER_ROWS_D	PAY_USER_ROWS_F	pydtussrr.sql
PER_ALL_ASSIGNMENTS_D	PER_ALL_ASSIGNMENTS_F	pedtasgn.sql
PER_ALL_PEOPLE_D	PER_ALL_PEOPLE_F	pedtpepl.sq
PER_ASSIGNMENT_ BUDGET_VALUES_D	PER_ASSIGNMENT_ BUDGET_VALUES_F	pedtabv.sql
PER_COBRA_COVERAGE_ BENEFITS_D	PER_COBRA_COVERAGE_ BENEFITS_F	pedtccbf.sql
PER_GRADE_SPINES_D	PER_GRADE_SPINES_F	pedtgrsp.sql
PER_SPINAL_POINT_ PLACEMENTS_D	PER_SPINAL_POINT_ PLACEMENTS_F	pedtsppp.sql
PER_SPINAL_POINT_STE PS_D	PER_SPINAL_POINT_STE PS_F	pedtspst.sql
PER_PERSON_TYPE_ USAGES_D	PER_PERSON_TYPE_ USAGES_F	pedtptu.sql
PER_CONTRACTS_D	PER_CONTRACTS_F	pedtctc.sql

Batch Element Entry

Creating Control Totals for the Batch Element Entry Process

Batch control totals provide a mechanism for customizing the validation of batch contents to meet particular user requirements. This validation may be done for example, by doing *total*, or *average* operations on the batch lines and matching the values with values entered by the user.

Batches can be entered and viewed using the Batch Header window, and other windows available from it.

Setting Up Control Totals

A control total type is predefined for checking the number of lines in a batch (control type = Total Lines).

You can create control totals to sum numerical element input values by defining a lookup for the lookup type CONTROL_TYPE. See: Setting Up BEE Validation Using Control Totals, *Oracle HRMS Configuring, Reporting, and System Administration Guide*

If you need other kinds of control totals, you can define lookups for them, but you must also write a validation procedure for checking the batch against the total. The next section explains how to write this validation procedure.

Creating the SQL Code

The following procedure is delivered with a null statement in it. Replace the null statement with your customized control total validation code.

- Procedure: check_control
- Package: user_check
- File: pyusrchk.pkb

Parameters

The check_control procedure is executed during the batch validation phase of the BEE process. The parameters passed to this procedure are:

- p_batch_id The batch ID.
- p_control_type The name of the control total.
- p_control_total The user entered value to match.

Two other parameters (p_status, p_message) are used in this procedure to return an error code and message to the system if the batch control total validation fails.

Batch Lines

Each line of batch data is stored as a record in the pay_batch_lines table. The data is stored in the fields value_1 - value_15. The number of the field corresponds to the column in the Batch Lines window.

For example, if you want to validate a check digit, you could use the following PL/SQL code as a basis:

```
PROCEDURE check_control

  (
```

```
    p_batch_id          IN      NUMBER,

    p_control_type      IN      VARCHAR2,

    p_control_total     IN      VARCHAR2,

    p_status            IN OUT  VARCHAR2,

    p_message           OUT     VARCHAR2

  ) IS

  total NUMBER;

BEGIN

-- Check the control type is the one we're expecting

IF p_control_type = 'CHECK_DIGIT' THEN

-- Calculate the MOD 10 of total values in value_1

    SELECT MOD(NVL(SUM(value_1),0),10) INTO total FROM pay_batch_
lines

    WHERE batch_id = p_batch_id;

-- Compare with the user entered value

    IF total <> p_control_total THEN

-- Create the error message to return and set the status to E(rro
r)

      p_message := 'Control total TOT1 (' || p_control_total ||

                   'does not match calculated value (' || total |
|

                   ')';

      p_status := 'E';

    ENDIF;

  ENDIF;

END check_control;
```

This, however, is a very simplistic example. If batch lines within the same batch are entered for more than one element then the value columns may vary between elements. Here is a more complex example to validate the check digit on the input value 'Identification':

```
PROCEDURE check_control

  (
```

```
    p_batch_id          IN      NUMBER,
    p_control_type      IN      VARCHAR2,
    p_control_total     IN      VARCHAR2,
    p_status            IN OUT  VARCHAR2,
    p_message           OUT     VARCHAR2
  ) IS
    CURSOR c1 IS
      SELECT DISTINCT element_type_id
        FROM pay_batch_lines
        WHERE batch_id = p_batch_id;
--
    r1 c1%ROWTYPE;
    total NUMBER;
    value_num NUMBER;
    sqlstr VARCHAR2(200);
    c2 INTEGER;
    ret INTEGER;
  BEGIN
--
-- Check the control type is the one we're expecting
    IF p_control_type = 'CHECK_DIGIT2' THEN
      total := 0;
--
-- Loop through each element in the batch lines
      FOR r1 IN c1 LOOP
--
-- Find out the value number that 'Identification' is in
        SELECT display_sequence
          INTO value_num
```

```
          FROM pay_input_values iv,
               pay_batch_headers bh,
               pay_element_types et
          WHERE bh.batch_id = p_batch_id AND
                iv.business_group_id = bh.business_group_id AND
                et.element_type_id = r1.element_type_id AND
                iv.element_type_id = et.element_type_id AND
                iv.name = 'Identification';
--
-- Create an SQL string to add the values
        sqlstr := 'SELECT MOD(NVL(SUM(value_' || value_num ||
                  '),0),10) ' ||
                    'FROM pay_batch_lines ' ||
                    'WHERE batch_id = ' || p_batch_id || ' AND '
                     ||'element_type_id = ''' ||
                     r1.element_type_id || '''';
--
-- Call the string using dynamic SQL and put the value in 'total'
        c2 := dbms_sql.open_cursor;
        dbms_sql.parse (c2,sqlstr,dbms_sql.v7);
        dbms_sql.define_column (c2,1,total);
        ret := dbms_sql.execute (c2);
        ret := dbms_sql.fetch_rows (c2);
--
-- Check we got some values back
        if ret > 0 then
          dbms_sql.column_value (c2,1,total);
        else
          total := 0;
```

```
        end if;
--
        dbms_sql.close_cursor (c2);
--
-- Check the total matches the user entered value and create an
-- error message if it doesn't
         IF total <> p_control_total THEN
          p_message := 'Check digit expected '||p_control_total||
          ' but got '||to_char(total);
          p_status := 'E';
         END IF;
      END LOOP;
    END IF;
  END check_control;
```

Payroll Processes

Overview

Oracle Payroll provides you with the flexibility you require to run your regular pay cycle in the best way to meet your business needs. To do this, we provide you with a modular batch process called PYUGEN.

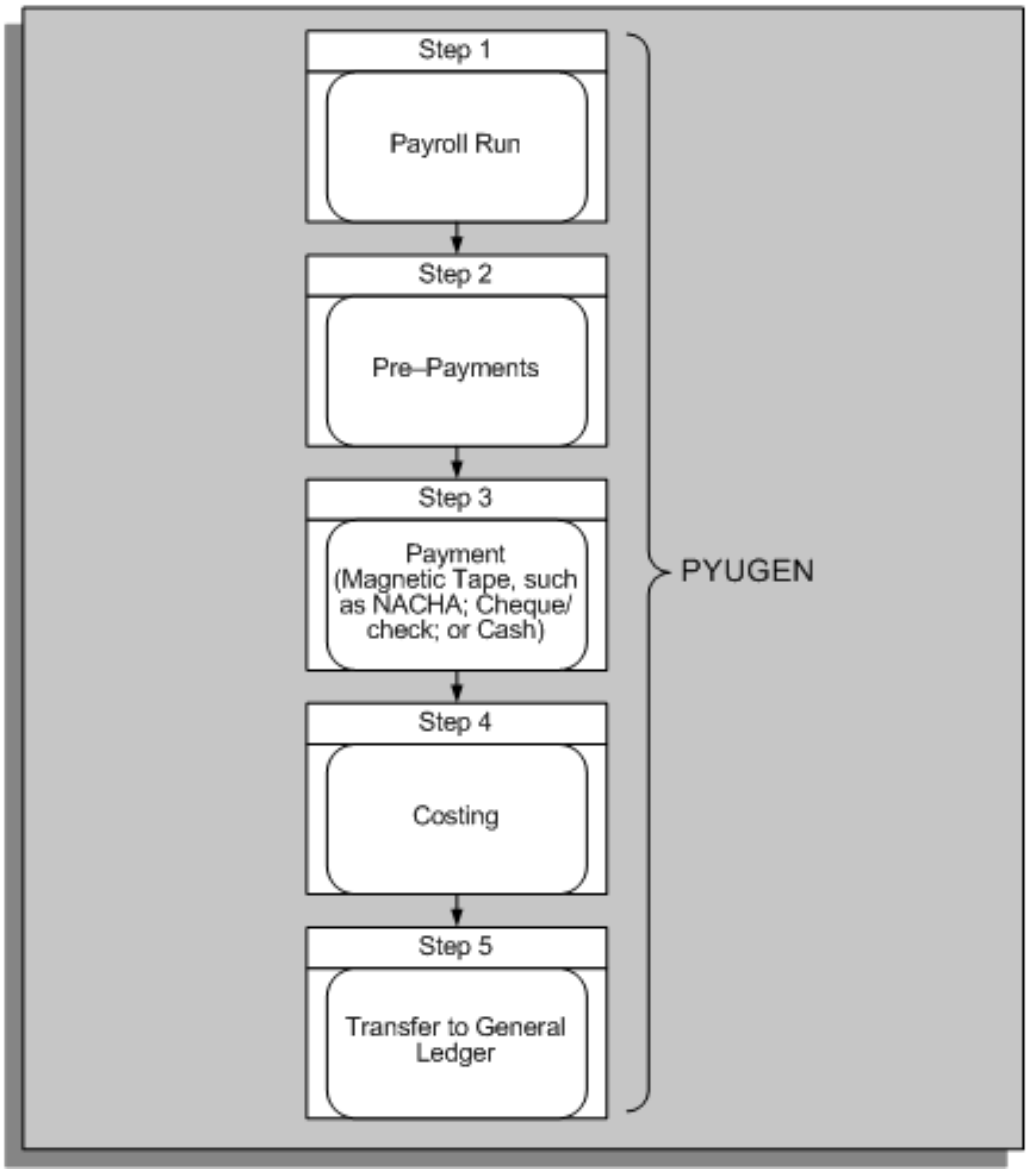
PYUGEN

PYUGEN is a generic process that can perform several actions. The Oracle Payroll system administrator specifies which actions it can perform by registering it with certain parameter sets and defaults.

The parameter identifies the specific payroll process to execute. These are predefined in Oracle Payroll; the values are not visible to the user.

The following figure illustrates the payroll processes executed by PYUGEN, and the typical sequence in which they are performed. Each process performs specific actions required to calculate and generate your employees' pay.

Pay Cycle Sequence



hr_400.gif

Checking Registration Details

You can check the registration details for each payroll process using the Concurrent Programs window. These details are predefined and are protected from change. During implementation you can add your own versions of these payroll processes to simplify the running of a pay cycle for your users. For example, you might want to define a separate payroll run process for each payroll, with different:

- Names
- Security
- Default values for different users

Consult your *Oracle Applications System Administrator's Guide* for more information on registering concurrent programs.

Payroll Action Parameters

Payroll action parameters are system-level parameters that control aspects of the Oracle Payroll batch processes. It is important to recognize that the effects of setting values for specific parameters may be system wide.

See: Payroll Action Parameters, page 2-132

Overview of the Payroll Processes

The first process you run in your pay cycle is the Payroll Run process. This process calculates the gross to net payment for your employees. After the successful completion of the Payroll Run, you start the Pre-Payments process. This process distributes employees' pay over the payment methods employees have requested. It also allocates payments to third parties.

The next step is to start one of the payment processes to produce payments for employees:

- MAGTAPE (for example BACS in the UK or NACHA in the US)
- CHEQUE (Cheque Writer or Check Writer)
- CASH (Cash) - for UK only

The payment processes take the unpaid prepayment values allocated to each payment type and produce the required payment file. It is these processes that actually produce payments for employees.

The Costing process allocates payroll run results to cost segments. The Transfer to the General Ledger process transfers cost information to Oracle General Ledger interface tables.

See Also

Payroll Run Process, page 2-100

Pre-Payments Process, page 2-108

Payment Processes, page 2-112

- Magnetic Tape Process, page 2-112
- Cheque Writer/Check Writer Process, page 2-126
- Cash Process, page 2-132

Costing Process, page 2-146

Transfer to General Ledger Process, page 2-146

Supporting Processes

In addition to this regular cycle of activities there are other processes that support the correction and completion of each cycle. These include:

- Mark for Retry
- Retry
- Rollback
- QuickPay

- RetroPay
- Advance Pay
- Archive

See the guide *Running Your Payroll Using Oracle HRMS* for more information about these supporting processes. See: The Payroll Archive Reporter (PAR) Process, page 2-148 for information about the Archive process.

Assignment Level Interlocks

The sequence in which the PYUGEN calculates payment is critical to the success of processing. This is because each process builds upon the results of the previous process in the sequence. The sequence of the processing is also determined by issues of data integrity. For example, the Pre-Payments process (which prepares the payments according to the payment methods) uses the results of the Payroll Run process (which calculates the gross to net payment).

To ensure correct payments, you cannot change Payroll Run results without also changing the prepayment results. Oracle Payroll uses assignment level interlock rules to enforce this.

See: Assignment Level Interlocks, page 2-141.

Payroll Run Process

The Payroll Run process calculates the gross to net payment for your employees.

This process uses *payroll actions* to represent each payroll run. It identifies which assignments have payroll actions performed on them - that action is an assignment action of the type *payroll*.

The results from processing each element for an assignment are the *run result values.* These individual results are accumulated into balances that summarize gross to net, and in particular the payment balances. Payment balances are taken forward by Pre-Payments, which is the next process in the regular pay cycle.

Determine Assignments and Elements

The first phase of the Payroll Run process is to determine the assignments and elements to be included in the current batch. The user specifies these by selecting an assignment set and element set when initiating the run. The default is All.

The Payroll Run accesses a number of specific entities for processing. It identifies whether they are used for select, update, delete or insert. Where an entity is datetracked, the Payroll Run process also identifies any datetracked information that has changed, and actions it accordingly. For example, an update of a datetracked entity may require an actual insert into the table.

The following list indicates the main entities for processing:

Key: S = Select, U = Update, D = Delete, I = Insert.

Entity Name	Datetracked?	Processing
Payroll Action	No	S, U, I
Assignment Action	No	S, U, I
Element Entry	Yes	S, U
Element Entry Value	Yes	S, U
Person Latest Balance	No	S, U, I
Assignment Latest Balance	No	S, U, I
Balance Context	No	S, U, I
Action Context	No	S, I
Run Result	No	S, U, I
Run Result Value	No	S, U, I

Process Each Assignment

The Payroll Run applies the appropriate processing to each assignment. For a specific payroll run, this is identified by an assignment action. The following 'pseudo code' represents the processing that occurs:

```
get assignment status();

if assignment status is 'Process' then

     load element entries and values ();

     load latest balances ();

     while(entries to process)

          create run results if necessary ();

          set up User Defined Context Area ();

          /* third party hook */

          get processing mode for entry ();

          if(we are not skipping) then

               look for formula to run ();

               if(there is formula to execute) then

                    execute formula ();

                    if(error detected) then
```

```
                    handle error ();

                end if;

            end if;

            post run results and feed balances ();

        end if ;

    end while ;

    flush run results and values ();

    write / update latest balances ();

end if ;
```

Element Entry Processing

Element entries hold the entry values that are input to the gross to net calculations. The result of processing each entry value is a run result value. Before processing each assignment, Payroll Run loads all entries for that assignment into memory. This includes any pre-inserted run results and values.

By default, nonrecurring entries are only fetched if they are unprocessed in the current pay period. Recurring entries are always fetched and processed when you submit a payroll run. You must use frequency rules, element skip formulas, or element sets to limit the inclusion of recurring entries.

If you make an additional entry of a recurring element, the Payroll Run processes the additional entry as a nonrecurring entry. (Additional entries are not used by Oracle Payroll in the US.)

Processing Priority

The sequence of processing entries for each assignment is determined by the processing priority of the element, and the subpriority order of each entry. When the subpriority is null, entries are ordered by:

1. processing priority
2. element_type_id
3. entry type

Payroll Run checks for Overrides and Replacement entries before calculating normal entries and additional entries for non-US legislations.

If subpriority is specified, the in-memory list is reordered to reflect this. Adjustments and target entries are kept together.

Termination Processing

Payroll Run implements the entry processing rules for a terminated assignment.

For the US legislation, this means that if the date earned of Payroll Run is between the *actual date* of termination and the *final process date* for an assignment, the assignment is processed only when there exists an unprocessed nonrecurring entry for the assignment.

For non-US legislations, a user can also enter a *last standard process* date. This means that if the date earned of Payroll Run is between the last standard process date and the

final process date for an assignment, the assignment is processed only when there exists an unprocessed nonrecurring entry for the assignment.

An additional entry counts as nonrecurring for termination purposes.

Create Run Results and Values

For every entry that is processed there must be a run result; for each entry value there must be a run result value. If these do not already exist, by pre-insertion, then the appropriate run results and values are created in memory and are inserted into the database, ready for Payroll Run to process.

For example, a nonrecurring entry may have pre-inserted run results and values if you have entered the Pay Value.

Pre-inserted values are automatically deleted by a rollback or mark for retry operation, and Payroll Run re-establishes them. However on the rollback of a reversal, nonrecurring pre-inserted values are re-established.

At the same time, Payroll Run uses the current exchange rate for the payroll to perform any currency conversions. This happens if the input and output currency codes of the element are different. You can define an element with any input currency.

If the element contributes to a payment balance for the employee the output currency must be the base currency of the Business Group. Payment balances can be converted into other currencies as part of the PrePayments process linked to payment methods.

Set Up Contexts

Before an entry is processed, Payroll Run sets up the contexts that are needed by FastFormula for Payroll and Element Skip formulas. This may include legislative specific contexts. The values of all the contexts are held in a special data structure, known as the User Defined Context Area (UDCA). The generic contexts that are always created provide additional route information for the formula. These are:

- ORIGINAL_ENTRY_ID
- ELEMENT_ENTRY_ID
- BUSINESS_GROUP_ID
- PAYROLL_ACTION_ID
- PAYROLL_ID, ASSIGNMENT_ID
- ASSIGNMENT_ACTION_ID
- DATE_EARNED
- ELEMENT_TYPE_ID
- TAX_UNIT
- JURISDICTION
- SOURCE_ID

A special third party interface is called so that the value of legislative specific contexts can be set. This has been used extensively for US legislations.

Run Element Skip Rules

Element Skip Rules enable you to define specific formula criteria to determine whether an entry is processed or not. A skip rule formula must return a skip_flag value of Y or N.

Where appropriate, a skip formula is fired and any input values are taken from the in memory run result values (to allow for any currency conversion). When looking at the skipping of an adjustment, the formula inputs are taken from the entry values of the normal target entry, not the adjustment entry itself.

There may also be legislative-specific skip rules predefined for specific elements. This additional third party skip hook is called at the same time that the internal function looks for a normal skip formula. This legislative specific skip rule is defined in 'C' code.

Element Entry Processing Modes

Payroll Run uses processing modes to control whether entries of an element are processed. At first, the mode is set to indicate that it should process. Then, depending on the entry type and whether a skip rule has fired, a different mode may be set. This controls the processing of the current entry and (possibly) other entries of the same element. For example, when processing an Override entry, the mode is set to Override. This mode persists throughout the processing of this element, so no other entries are processed.

Create and Maintain Balances

Payroll Run needs to be able to access and maintain balances and latest balances. In summary, the Payroll Run:

- Loads any existing assignment- or person-level latest balances into memory
- Checks all loaded balances for expiry, and sets them to zero if they have expired
- Creates new in memory latest balances, where required
- Adds the appropriate run results to the current value of balances in memory
- Writes the new balances to the database (for some balance dimensions types only)

For more information about latest balances, see: Balances in Oracle Payroll, page 2-160.

Loading Balances Into Memory

Any existing assignment-level or person-level latest balances (and any associated balance contexts) are loaded into memory before any entries are processed. The basic data structure for this is a doubly linked list, kept ordered by balance_type_id. The balance values themselves are held and manipulated as Oracle Numbers. The fetch is a union, in this case because the two types of balances are held in separate tables.

Expiry Checking of Latest Balances

Latest balances should expire (that is, return to zero) at a time determined by their dimension. For example, a YTD (Year to Date) balance expires at the end of the year.

All loaded balances are checked for expiry. If they have expired, they are set to zero. The expiry step is entirely separate from the loading step, due to the need to deal with balance context values.

To process expiry checking, the Payroll Run calls Expiry Checking code that is held in a PL/SQL package. To prevent performance from being degraded, the number of accesses required is cut down by making certain assumptions about the different expiry checking levels. The assumptions made are determined by the balance's expiry checking type. See: Expiry Checking Type, page 2-165.

Creation of In Memory Latest Balances

Not all balances are loaded from the database, some have to be created. Once they have been created, they have to be maintained.

For some dimension types, the newly created or updated balances must be written to the tables.

A balance's dimension type determines how it is treated by the payroll run. For example, balances with the dimension type F are fed but not stored, so the Payroll Run creates a balance in memory. For a description of the dimension types, see: Dimension Type, page 2-164.

There are three places in the code where *in memory balances* are created. One place is for dimension types A, P and F, and two places are for type R.

- An in memory balance is created when a formula has just accessed a defined balance with the dimension type A, P or F and which is not already held as an in memory balance. The in memory balance is created using the value accessed by the formula.
- An in memory balance with a value of zero is created before the execution of a formula, if the formula accesses a defined balance with the run level balance dimension type (R). (A run level balance must be zero, by definition.)
- In memory balances with a value of zero are created before balance feeding time if the code is attempting to feed defined balances with run level dimension types (R).

The corollary of the above rules is that, except for the Run Level dimension type, a latest balances can only be created for a particular defined balance when that balance is accessed by an executed formula.

Run Results Added to In Memory Balances

Next, the appropriate run results are added to the current value of the balance.

A summary of the algorithm that is used is:

1. For each processed run result, look at the balance feeds, which identify the balance types that are potentially fed by each run result value.
2. Scan the in memory balances to see if there are any potential feeds.
3. If there are, perform feed checking.

   The feed checking strategy is determined by the feed checking type on the appropriate balance dimension. See: Feed Checking Type, page 2-165.
4. If the result of feed checking is that the run result should feed the balance, then: balance value = balance value + (result value * scale).

In the case of run result values that might feed run level balances, Payroll Run might need to create them in memory, before feed checking occurs. Since Payroll Run cannot identify which balances might be required at this point, it has to create all those it might need.

In practice, this means it creates balances for each of the run level defined balances that might potentially be fed by the run result being examined.

> **Note:** If the dimension type is R and the feed checking type is set to S, this represents a special case for United States legislation. A different algorithm is used in this case.

Writing of In Memory Balances

The contents of the in memory balances (and any associated contexts) need to be written to the database as appropriate, that is, where the replace flag on the in memory balance is set. Only balances with a dimension type of A or P are written. This occurs after all entries have been processed for the current assignment action.

After all element entries have been processed for the assignment, the in memory balance list is scanned, data is moved to an array buffer and then array inserted or updated on the database.

Run Formulas

Payroll Run calls FastFormula to enable it to perform its complex calculations.

> **Note:** Even if a formula has been defined against an element using a formula processing rule, it does not fire if the Pay Value is not null.

The FastFormula Interface

The interface used by Payroll Run to access FastFormula is made up of two sections, which are:

- The common part of the interface (available to any product)

  This sets up pointers from Formula's internal data structures to the data to be input to the formula (contexts and inputs) and output from the formula (formula results).

- A special interface

  This is designed especially for Payroll Run, and allows access to Formula's database item cache.

Execution of FastFormula by Payroll Run

Payroll Run goes through the following steps:

1. Declares that a new formula is executed.
2. Formula tells the run code what formula contexts, inputs and outputs are required.
3. The in memory balance chain is scanned.

   If the formula might access any of the defined balances held as latest balances, it writes the current value of the balance to the FastFormula database item cache.
4. Any formula contexts are satisfied. All the values are taken from the User Defined Context Area (UDCA).
5. Values that are passed to the formula as 'inputs are' variables are satisfied. This is done by looking for a run result value that has an associated Input Value name matching the input variable name.
6. The outputs that FastFormula has told the run code about are directed to a buffer area.

Execute the Formula

The third party post formula hook is called. This enables special legislative dependent functions to manipulate the formula results before they are processed by Payroll Run. For instance, it enables certain run results to be suppressed.

The formula results are processed.

Processing the Formula Results

Following the execution of a formula, Payroll Run loops through any returned results, processing them as required by the formula result rules. It looks for a formula result rule name that matches the formula result that has been returned. There are several types of result rule, and they are summarized below, from an internal processing point of view.

Message Rule

If the severity level of the message is fatal, it causes an assignment level error. Otherwise, the message is written to the messages table. Note that the length of a message is restricted to the size that can be held in the run result values table (currently 60 characters).

Direct Rule

If the Unit Of Measure is Money, the value is rounded as necessary. Then the run result value chain is searched for the entry holding the Pay Value and is updated. The replace flag is set to indicate this.

Indirect and Order Indirect Rule

These two types are grouped together, because they cause very similar processing. During the processing of the current element entry, all indirects are held on a temporary chain, and merged into the main entry chain later.

First of all the temporary chain is searched. If there is no existing entry for the element, a new one is created and added to the chain. Then, in the indirect rule case only, the appropriate entry value is located and updated with the new value. In the Order Indirect case, the subpriority of the indirect entry is set to the formula result value.

> **Note:** If two formula result rules target the same input value, the second result to be processed takes precedence.

Following the processing of all formula results, the chain of indirects is merged into the main element entry chain at the appropriate point. What is appropriate depends on the main processing priority and the subpriority (which can be set using the Order Indirect rule).

Payroll Run prevents the processing priority of an indirect element from being the same as the element that gives rise to the indirect. However, the form continues to disallow this. Same priority indirects was provided specifically for United States legislative requirements.

Same priority indirects can cause problems, however, because they create an endless loop.

Update Recurring Rule

Payroll Run calls a PL/SQL procedure to find the appropriate element entry to update. This procedure then performs the date effective update. If this entry happens to exist further down the entry chain, its value is updated to reflect the change.

Stop Recurring Rule

Payroll Run calls a PL/SQL procedure to find the appropriate element entry to stop. This procedure then performs the date effective delete.

Run Result Processing

The run result and their associated run result values form the corollary of element entries and element entry values. The entries express eligibility to certain elements, whilst the results and values contain the after effect of processing those entries.

During processing, run results and values are held in memory, hung off the in memory element entry chain. This reflects their close connection in database terms.

Creation of Run Results and Run Result Values

Results and values are created internally in one of three ways:

- Loaded when entries and entry values are loaded - as pre-inserted results, arising from nonrecurring element entries.
- Created by Payroll Run before processing the appropriate element entry if there are any missing results and values.
- Created via indirect results.

Defaulting of Run Result Values

Payroll Run handles Hot and Cold defaulting while it checks that results and values exist. If results and values do already exist, and are null, Payroll Run attempts to default them.

If currency conversion is required, it is performed at the same time. Internally, it uses Oracle Numbers for the calculation. Following this, if it is processing an input value with a 'Money' Unit of Measure, it performs rounding on the result as necessary.

Writing Results and Values to the Database (Flushing)

The process moves the results and values to a special buffer and then writes the run results and values to the database (update or insert). It uses array processing techniques (similar to the technique used by latest balances).

This process is usually referred to as *flushing the results* and there are two circumstances that may trigger it:

- If the process is about to execute a formula that accesses a database item not held in memory. The route for that database item might need to access run results that have been generated so far in Payroll Run itself. This assumption is made because there is no way of finding out for sure.
- When all the element entries for the assignment action have been processed, any remaining results and values are flushed.

Payroll Data Cache

During processing, Payroll Run has to access attributes of certain entities that represent static definition data. For instance, it may need to know the element name or the balance feeds for a particular input value. Furthermore, the same data typically requires access many times over. If this data were selected from the database every time it was needed, it would cause severe performance degradation.

To resolve this problem, a special static payroll data cache was introduced. All the appropriate data for the entity is loaded into memory the first time it is accessed. From then on, any subsequent accesses to the data can go straight to memory.

Pre-Payments Process

The Pre-Payments process prepares the payments generated by the Payroll Run for payment. It prepares payments for each assignment and inserts the results into PAY_PRE_PAYMENTS for each payment method for an assignment.

The Pre-Payments process also:

- Calculates the amount of money to pay through each payment method for an assignment, and converts any currency if the payment method is in a foreign currency.
- Handles the preparation of third party payments.

For example, garnishments, court orders and child maintenance. Third party payments are managed through the definition of special payment methods for the employee.

Setting Up Payment Methods

During implementation, you set up your own specific payment methods with source account details. When you hire an employee, you can record one or more payment methods for the employee, and apportion payment by percentage or amount. You can also record payment methods in different currencies.

The Pre-Payments process prepares payments following the payment methods for each assignment. There are three predefined payment types that Oracle Payroll processes:

- Cheque/Check
- Magnetic Tape (such as NACHA/BACS)
- Cash (UK only)

You can set up as many payment methods as you require (based on the three predefined payment types) to support your business needs.

Every payroll has a default payment method. Pre-payments uses the default method when there is no personal payment method entered for a specific assignment.

> **Note:** You cannot have a default method of type Magnetic Tape. This is because Magnetic Tape payment methods require knowledge of the employee's bank account details, including prenotification details in the US.
>
> See Prenotification, page 2-110

Payment methods are processed in order of their priority for an assignment. For example, an employee may want:

1. 50% of the salary to be paid directly into their bank account by Magnetic Tape payment
2. 100 dollars paid by Cheque/Check
3. 100 dollars paid in Cash

Pre-Payments prepares the payments in priority order, provided that the amount to be paid covers the payments. If there is less to be paid than the payment methods specify, the system pays up to 100% and stops. If there is more to be paid than the payment methods specify, the system adds the excess to the last payment method.

Preparing Cash Payments (UK Only)

If you are using Oracle Payroll to prepare cash payments, you can calculate the banknote and coinage requirements for each employee. Pre-Payments breaks down the amount into the individual monetary units for payment and insert the results into the PAY_COIN_ANAL_ELEMENTS table.

You can define the monetary units for each currency you pay for cash payments administered through Oracle Payroll. You can also define cash analysis rules to specify minimum numbers of each denomination of the currency.

Setting Up a Cash Rule

The are two steps to setting up a cash rule:

1. Alter the package body hr_cash_rules

   The alteration should test for the name of the cash rule you want to set up and then perform the payment. For example, if the rule name is 'TENS AND FIVES' then enter the following:

   ```
   if cash_rule = 'TENS AND FIVES' then

   --

   hr_pre_pay.pay_coin(6, 10)

   hr_pre_pay.pay_coin(3, 5)

   --

   -- number to pay ---^  ^--- unit value of currency

   --

   end if;\
   ```

   Using this cash rule with a currency of dollar results in a minimum of 6 ten dollars and 3 five dollars being paid (given sufficient funds).

2. Register the rule.
3. Enter the Lookup Values window and query the Lookup type of CASH ANALYSIS.
4. Add the new Cash rule with the meaning and description fields set to TENS AND FIVES.
5. Use the cash rule when setting up an organization payment method.

Prenotification (US Only)

Prenotification validation (also known as prenoting) applies to payment methods of the type Magnetic Tape. This validation is performed when bank details require checking before a payment can be made. For example, when an employee has changed banks or changed bank details, a payment value of zero is made to the employee's bank account. The payment is then made by subsequent methods, or by the default method.

Consolidation Sets

Pre-Payments is run for a consolidation set. A consolidation set is a tag that ties groups of actions together. You can use a consolidation set to prepay all assignment actions in the set that have not yet been prepaid. These assignment actions can be for different payrolls and different time periods. For example, you could use a consolidation set to force the magnetic tape process to pay both of a company's payrolls where one is monthly and one is weekly.

Third Party Payments

Third party payments are post tax deductions from an employee's salary, that are paid to organizations or individuals. For example, court orders are payable to a municipal court whereas child support orders may be directly payable to a spouse, or other individual.

These payments are processed in a slightly different way. The element entry that produces the run result value for the payment holds details of which payment method to use. This enables you to make more than one entry of a third party payment element to an

assignment, with each entry representing a payment to a different party. For example, an employee can pay a third party element of Child Support to two different people.

Third party payments can only be made by magnetic tape or cheque/check. Cash payments are not allowed. In addition, these methods pay the full amount of the payments, so only one method is used. There is no default method for these payments, so a payment method must always be specified. US: If the magnetic tape prenote validation fails, the process creates an error for that assignment.

Exchange Rates

Pre-Payments calculates the currency conversion if the payment is in a different currency to that of the remuneration balance (the element output currency in the case of third party payments). If the process cannot find the exchange rate for the two currencies, it creates an error for the assignment.

Overriding Payment Method

You can specify an overriding payment method when making a prepayments run. This method overrides the personal payment methods, so the full amount of the payment is made by the overriding method. The only exceptions are the third party payments; these are paid by the method specified in the element entry.

The overriding payment method can be either:

- Cash
- Cheque/check

You cannot specify magnetic tape payments as an override method, as this type of payment requires prior knowledge of bank account details.

The Process

The Pre-Payments process creates payroll actions and assignment actions. The assignment actions are based on assignment actions of the payroll/consolidation set specified that do not have interlocks to a prepayment process. The interlocks guarantee that Payroll Run cannot be rolled back until Pre-Payments is rolled back. Thus, the new assignment actions are created with interlocks to the run's assignment actions.

See: Assignment Level Interlocks, page 2-141

Chunking

The assignment actions are split into groups called chunks, the size of which are denoted by the CHUNK_SIZE action parameter in the PAY_ACTION_PARAMETERS table. The process could spawn several threads (child processes), depending on the THREADS action parameter. Each thread then picks a chunk to process, processes the assignment actions and then picks another chunk until all the chunks are processed. The number of threads can be used to enhance performance on multiprocessor machines.

PL/SQL Procedures

The main part of the C process (the section that performs the payment), is a harness for PL/SQL procedures. The PL/SQL procedures create the entries in the Pre-Payment table.

The threads process the assignment actions by:

- Retrieving the third party details and recording third party payments as defined by the personal payment methods
- Retrieving the value for the assignment's remuneration balance using the PL/SQL balance functions

- Recording payment of this value as defined by the payment methods

Error Handling

Errors encountered while processing can be at two levels:

- Payroll action level

  These errors are fatal.

- Assignment level

  These errors occur while processing assignment actions. If an error is encountered at this level, it marks the assignment action's status as in Error, and continues processing. If the process then completes, it marks the payroll action status as Complete.

Using the MAX_ERRORS_ALLOWED action parameter you can set the number of assignment errors that can be processed before an error should be raised at payroll action level. If MAX_ERRORS_ALLOWED is not found then the chunk size is used as a default.

All the error messages are written to the PAY_MESSAGE_LINES table with a more detailed explanation in the log file.

This method of handling errors enables Pre-Payments to continue processing if minor errors are encountered. For example, if Pre-Payments has thousands of assignments to process and a few are paid by cash but the currency details have not been loaded, the process creates an error for the assignments with cash payments ("Process unable to perform the cash breakdown"). Most assignment actions complete, only the assignments with errors have to be rerun.

Payment Processes

After running the Pre-Payments process to prepare the results for payment (according to the payment methods), you produce payments for your employees.

With Oracle Payroll, you can run the following types of payment process:

- The Magnetic Tape process - MAGTAPE

  See: Magnetic Tape Process, page 2-112

- The Cheque process - CHEQUE

  See: Cheque Writer/Check Writer Process, page 2-126

- The Cash Payments process - CASH (UK only)

  See: Cash Process, page 2-132

The payment processes take the unpaid prepayment values allocated to each payment type and produce the required payment file.

You can also record any manual payments you make to a specific employee. These payments are not handled by the Payments processes. Recording a manual payment has the effect of marking the prepayment as paid.

Magnetic Tape Process

The Magnetic Tape process generates the payment due and writes the data to a file on magnetic tape. It is this tape that is taken to the bank for payment.

There are two types of magnetic tape file, which are created differently:

- Payments

- End of year tax reporting

The actual format of these tapes is legislation specific.

The tape process is a simple 'C' harness which calls Oracle stored procedures and FastFormula formulas to produce the required tape file. The routine is generic: you can use it for any task that requires magnetic tape reporting. The actual structure and content of the tape is defined entirely by the stored procedure and a series of formulas.

Some examples that use the routine are:

- BACS
- NACHA
- W2
- P35 submissions (and equivalent in other countries)

> **Note:** The order of the entries in the magnetic file is critical. Therefore the Magnetic Tape process cannot run with multiple threads (unlike the PrePayments or Cheque/Check Writer processes).

See also:

The Payroll Archive Reporter (PAR) Process, page 2-148

Running the Magnetic Tape Payments Process

The payroll assignment action creation code is the entry point to the Magnetic Tape Payments process. Employee magnetic tape payments are recorded in Oracle HRMS as payroll and assignment actions with interlocks to the relevant pre-payment assignment actions. The interlocks prevent the pre-payments actions being rolled back while the magnetic tape actions exist.

Third party payments (such as the company's health plan contributions) do not result in payroll and assignment actions, and therefore would use the magnetic tape report interface.

Batch Process Parameters

You run PYUGEN with the following parameters:

- **consolidation_set_id - mandatory**

  Defines which set of unpaid pre-payments are paid.

- **payment_type_id - mandatory**

  Defines the driving PL/SQL procedure.

- **effective_date - optional**

  Identifies the effective date for processing.

- **payroll_id - optional**

  Restricts the assignments processed to those on the specified payroll on the effective date

- **start_date - optional**

  Specifies how far back the process searches for target prepayments. If this parameter is not specified, then the process scans back to the beginning of time.

- **organisation_payment_method_id - optional**

Creates assignment actions interlocking to unpaid prepayments for that payment.

- **legislative - optional**

  Free-format parameters, available to all payroll actions. Your localization team may use these to pass in a number of legislation-specific parameters, made accessible to the payroll action through the entity horizon.

PL/SQL Procedure for the Payment Type

The system uses the PL/SQL driving procedure specified for the payment type on the database (for example, <package name>.<procedure name>). The PL/SQL procedure for the Magnetic Tape Writer process must drive off the assignment actions and not further restrict the assignments processed. Further restricting the assignments presents the danger of leaving some magnetic tape assignment actions never processed. When the process first runs the PL/SQL, one of the parameters passed is the payroll action id (PAYROLL_ACTION_ID).

The Magnetic Tape process actions prepayments with an effective date on or before the effective date of the magnetic tape action. The magnetic tape effective date defaults to session date in an AOL environment, and sysdate outside AOL.

Output Filenames

The magnetic tape file generated is named as per the normal file-naming standards:

p<trunc(conc_request_id, 5)>.mf

The file name is padded with zeros if the length of the request id is shorter than five characters, (for example, p03451.mf).

It is written to the $APPLCSF/$APPLOUT directory, if $APPLCSF is defined, and otherwise to $PAY_TOP/$APPLOUT.

Several other files can be produced by this process. You can use these files to audit the assignments that are being processed. The audit files are created in the same way, except that the file extension .a<file_number>. So if a formula returns a value for audit file 6 then a file with the extension .a6 is created in the correct directory using the concurrent request id as described above.

Running Magnetic Tape Reports

Magnetic Tape reports are not recorded as payroll and assignment actions. The entry point is the specific Magnetic Tape code, PYUMAG. The PL/SQL determines which assignments to process.

Mandatory Parameters

- Driving PL/SQL procedure (<package name>.<procedure name>)
- Output file (full pathname included)

Optional Parameters

- Audit file prefix (the prefix to the extension, plus the full path)
- Effective date (the parameters to the driving PL/SQL procedure)

The optional parameters to the PL/SQL must be tokenised, so that the generic tape writer process can populate the PL/SQL tables for parameter name and parameter value. These tables constitute the interface between the generic writer process and the driving PL/SQL procedure.

See: The PL/SQL Driving Procedure, page 2-117

The magnetic tape action only processes formulas with an effective date on or before the effective date of the magnetic tape action. The magnetic tape effective date defaults to session date, in an AOL environment, and sysdate outside AOL.

Output Filenames

The magnetic tape filename is generated if it is not supplied to the process. The filename is in the format:

o<trunc(conc_request_id, 5)>.mf

When an audit file prefix is not set but the process tries to write to an audit, the concurrent request id is used as the prefix and .out used as the extension. In these circumstances all audit returns are written to this file.

SRS Definitions

Using SRS, the generic tape writer process is defined once as an executable. You can then define any number of concurrent programs that invoke that executable. Each concurrent program can have its own set of parameters, its own hidden parameters, defaults and so on. For example, we can define two concurrent programs:

- W2 report
- Illinois Quarterly State Tax report

They would both use the magnetic tape writer executable PYUMAG, each with a hidden parameter specifying the appropriate PL/SQL procedure, and possibly, each with specific parameters. They appear as completely distinct reports to the user. This would be set up in the SRS process interface.

Similarly, magnetic payments can be made to appear as distinct processes to the user - the only difference is that the payment type is the hidden parameter, and the generic code determines the driving PL/SQL procedure from that.

How the Magnetic Tape Process Works

Magnetic tapes are usually broken down into:

- Records
- Fields

The sequence in which the process writes the records to tape follows strictly defined rules. As a result, you can write a piece of code to return the name of the next record to write to tape.

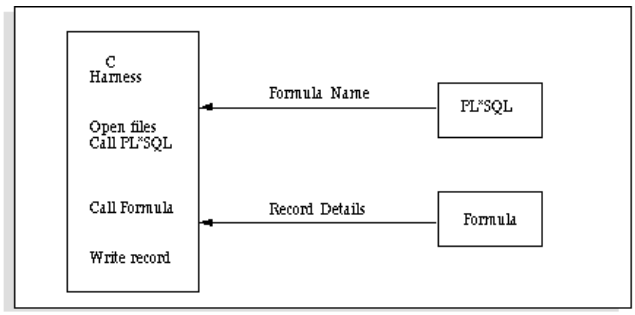
Similarly, the actual records have strict field place and length requirements. For example:

Record	Fields
Tape Header	Batch Id, Company Name, Batch Record Length, and so on
Employee	Employee Id, Salary, Age, Job, and so on
Tape Footer	No. of Records Processed, Salary Total, and so on

C Harness, PL/SQL, and Formulas

The following figure illustrates the Magnetic Tape process.

The Magnetic Tape Process



A C code harness performs the file handling (opening, closing and writing to files), and enables the PL/SQL and the formulas to interface.

The driving PL/SQL code sequences records by returning the name of a formula.

Each formula writes one type of record, such as the Tape Header, to tape. It defines the contents of the record.

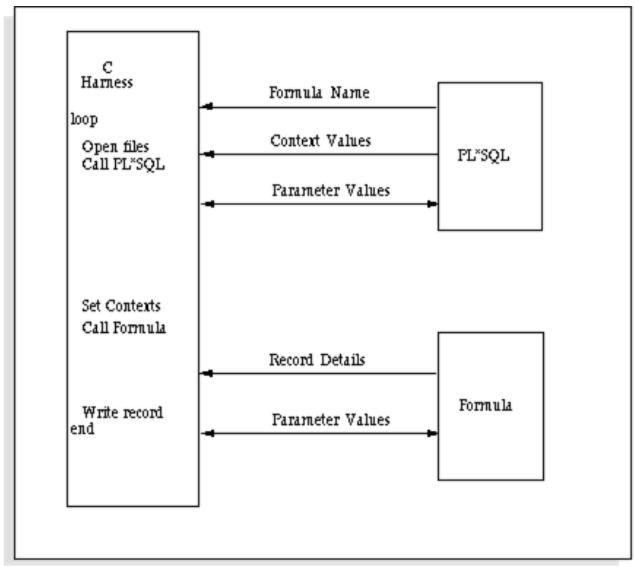
The process of getting the formula and record name, then writing the record to tape is repeated until all the records are processed.

Context and Parameter Values

The driving PL/SQL determines which type of record is required at any stage of the processing, and uses context and parameter values to communicate with the formula.

The following figure illustrates how the C code acts as an interface between the PL/SQL and formula, and how the data is passed as context values.

C Code Interface



Context Values

Formulas use database items to reference variable values. For example, the employee and assignment number could be different for each run of the formula and record.

The database item is held within the database, which consists of components to make up a SQL statement. As the value could be different for each run of the formula, the 'where' clause of the statement is slightly different. This is done by substituting key values into the 'where' clause that uniquely select the required value. These substitution values are known as context values.

Context values are set by the driving PL/SQL procedure that places the values into a PL/SQL table. The PL/SQL table is passed back to the C code, which in turn places it in the formula structure.

Parameter Values

Parameter values are used to store the variable data to be transferred between the formula and the PL/SQL. For example, the running totals are passed to the formula in this way.

The parameters can be:

- Passed into the C process from the command line
- Created by the driving PL/SQL procedure
- Created by the formula

Only the driving PL/SQL procedure and the formula can update the values.

The PL/SQL Driving Procedure

The PL/SQL driving procedure determines the format of the magnetic tape file. You can write this procedure from scratch by opening cursors processing a particular formula for each fetch of the cursor, or you can use the generic PL/SQL. The generic PL/SQL drives off the magnetic tape batch tables.

The interface between the 'C' process and the stored procedure makes extensive use of PL/SQL tables. PL/SQL tables are single column tables that are accessed by an integer index value. Items in the tables use indexes beginning with 1 and increasing contiguously to the number of elements. The index number is used to match items in the name and value tables.

The names of the tables used to interface with the PL/SQL procedure are:

- pay_mag_tape.internal_prm_names
- pay_mag_tape.internal_prm_values
- pay_mag_tape.internal_cxt_names
- pay_mag_tape.internal_cxt_values

The first two tables (pay_mag_tape.internal_prm_names and pay_mag_tape.internal_prm_values) are used to pass parameter details to the PL/SQL and formula. These are reserved for the number of entries in the parameter tables and the formula ID that is to be executed. The second two tables (pay_mag_tape.internal_cxt_names and pay_mag_tape.internal_cxt_values) are used to set the context rules for the database items in the formula. These are reserved for the number of entries in the context tables.

The Generic PL/SQL

The Magnetic Tape process uses generic PL/SQL that drives off several tables that contain cursor names. These cursors and tables control the format of the magnetic tape.

These cursors retrieve three types of data:

- Data that is used in subsequent cursors

- Data that is to be used as context value data
- Data to be held as parameter/variable data

Example

Here are two select statements as examples:

```
cursor business is

select business_group_id,

'DATE_EFFECTIVE=C', effective_start_date

from per_business_groups

cursor assignment is

select 'ASSIGN_NO=P', assignment_id

from pay_assignments
```

In the above example, the first select (DATE_EFFECTIVE) is a context value that is passed to a subsequent formula. The business_group_id column is retrieved for use in subsequent cursors. It is accessed by using a function described later.

The second select (ASSIGN_NO=P) is used as a parameter.

When the cursor is opened, it assigns rows in a retrieval table that it can select into (the number of rows depends on the number of columns retrieved by the cursor). For example, if the above cursors were used, and the previous example was run, the retrieval table would look like this:

After First Run	After Second Run
50000	50000
DATE_EFFECTIVE=	DATE_EFFECTIVE=C
16-MAR-1997	16-MAR-1997
	ASSIGN_NO=P
	50367

Functions to Access Data

Some cursors require access to data previously selected. This can be achieved in two ways:

- If the column was selected as a context or an individual column (like business group in the previous example), use the get_cursor_return function. It returns the value, given the cursor name and the column position in the select statement. For example, to get the business group in the above select statement use the following command:

  ```
  pay_magtape_generic.get_cursor_return('business', 1)
  ```

- Or, select the value as a parameter and access a function that retrieves that value given the parameter name. For example to get the ASSIGN_NO parameter value use the following command:

  ```
  pay_magtape_generic.get_parameter_value('ASSIGN_NO')
  ```

Context and Parameter Data

The formula requires two types of data:

- Context
- Parameter

The context data is held in PL/SQL tables, which are filled by the PL/SQL with data retrieved by the cursors, as described above. The context rules are inherited to lower levels unless the lower level cursor retrieves a different value for that context name. The PL/SQL always uses the lowest level context value for a particular context. For example, if the second cursor above retrieved a context value for DATE_EFFECTIVE, this value would be used for the formula until the cursor is closed. It is at a lower level in the retrieval table than the previous DATE_EFFECTIVE. When the cursor is closed, the rows in the retrieval table are reclaimed and the DATE_EFFECTIVE context reverts to the first one.

The Parameter data is also held in tables, but unlike context values the values are not level dependent. The formula can access these values by selecting the parameter on the input line. If the formula returns a value for that parameter, it overwrites the entry in the table. If the formula returns a parameter that does not exist, the parameter is entered in the table.

Cursor/Block Table

The driving structure for the package procedure is held in two database tables:

- PAY_MAGNETIC_BLOCKS
- PAY_MAGNETIC_RECORDS (the Formula/Record table, see below)

The PAY_MAGNETIC_BLOCKS table is as follows:

Name	Null?	Type
MAGNETIC_BLOCK_ID	NOT NULL	NUMBER (9)
BLOCK_NAME	NOT NULL	VARCHAR2 (80)
MAIN_BLOCK_FLAG	NOT NULL	VARCHAR2 (30)
REPORT_FORMAT	NOT NULL	VARCHAR2 (30)
CURSOR_NAME		VARCHAR2 (80)
NO_COLUMN_RETURNED		NUMBER (5)

Example

block_id	cursor_ name	block_name	no_of_ select_ values	main_block	type
1	company_ curs	companies	2	Y	CA
2	employee_ curs	employees	2	N	CA
3	assignment_ curs	assignments	1	N	CA

- Block_id is system generated.
- No_of_select_values is the number of columns retrieved by the select statement specified by cursor_name.
- Main_block signifies the starting block to use. Only one of these can be set to Y for a given report.
- Type refers to the type of report that the select statement represents.

Formula/Record Table

The PAY_MAGNETIC_RECORDS table is as follows:

Name	Null?	Type
FORMULA_ID	NOT NULL	NUMBER (9)
MAGNETIC_BLOCK_ID	NOT NULL	NUMBER (9)
NEXT_BLOCK_ID		NUMBER (9)
LAST_RUN_EXECUTED_ MODE	NOT NULL	VARCHAR2 (30)
OVERFLOW_MODE	NOT NULL	VARCHAR2 (30)
SEQUENCE	NOT NULL	NUMBER (5)
FREQUENCY		NUMBER (5)

Example

formula_ name	block_id	seq	next_block	frequency	O/F	exec.last
formula 1	1	1	-	-	N	N
formula 2	1	2	2	-	N	N
formula 3	2	1	-	-	N	N
formula 4	2	2	3	-	N	N
formula 5	3	1	-	-	N	N
formula 6	2	3	-	-	N	N
formula 7	1	3	-	-	N	N

Formulas/records can be of three general types:

- Standard formulas executed for every row returned from cursor
- Intermediate formulas executed once every x number of rows
- Formula executed depending on the result of the previous formula (overflow formula)

The table columns are as follows:

- Block id refers to the block that this formula is part of.
- Seq refers to the sequence in the block.
- Next_block column signifies that after this formula has run, the cursor defined by next_block should be opened and that block's formula should be run until there are no more rows for that cursor.
- Frequency is used by the intermediate formula to specify the number of rows to be skipped before the formula is run.
- O/F (overflow) specifies whether the formula is an overflow. If it is (set to Y), and if the last formula returned the TRANSFER_RUN_OVERFLOW flag set to Y, then the formula runs.

  Similarly, if the formula is a Repeated overflow (set to R), and the TRANSFER_RUN_OVERFLOW flag is set to Y then that formula is continually repeated until the formula does not return TRANSFER_RUN_OVERFLOW set to Y.
- Exec.last can apply to all the types of formula but most commonly the intermediate formulas. This column specifies that the formula can run one extra time after the last row has been retrieved from the cursor.

  For intermediate formula this column can be set to 4 different values:

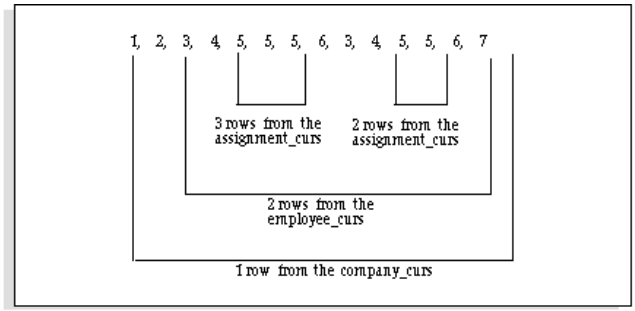
  - N - Never run after last row returned
  - A - Always run after last row returned
  - R - Run only if the intermediate formula has run for this cursor

- F - Run only if this is the first run of the formula for this cursor

Note: For overflow and standard formula only N and A are valid.

Using the above specification the formulas could be retrieved in the following sequence:

Formula Sequencing



The generic PL/SQL procedure identifies which type of report to process. It does this by passing the parameter MAGTAPE_REPORT_ID when calling the process. The previous figure illustrates how MAGTAPE_REPORT_ID=CA is passed when calling the process.

The Formula Interface

Typically, a magnetic tape consists of a number of record types. Oracle suggests having a formula associated with (generating) each record type. The formulas do the following:

- Define the field positions in the records
- Perform calculations
- Report on the details written to tape (auditing)
- Raise different levels of error messages

A PL/SQL stored procedure provides the main control flow and determines the order in which the formulas are called.

The routine uses FastFormula to prepare records The records are written to an ASCII file in preparation for transfer to magnetic tape. To implement the required actions, there are more formula result rule types. These are listed below:

Result Rule Types	Purpose
TRANSFER	This transfers the output parameter to the input of the stored procedure. The parameter may or may not be modified by the stored procedure before being used in the next execution of the formula.
WRITE TO TAPE	This instructs the process to write the result to the magnetic tape file. This is always a character string that represents the desired record. The writes are performed in the order in which they are returned from the formula.
REPORT FILE	This writes the string result to an "audit" file.
ERROR	This instructs the process that an ERROR/ WARNING has been detected within the formula. Thus the process should handle the error appropriately.

Naming Convention

These are not implemented in the traditional manner using the formula result rules table. They use the naming convention:

WRITE TO TAPE results are named WRITE_<result_name>.

TRANSFER results follow a similar convention, but the result_name part must be the name of the parameter. For example, a result company_total_income would be named transfer_company_total_income.

The REPORT result must identify which file is to be written to. The file number is embedded in the formula return name For example: REPORT1_<result_name> - this writes to report/audit file 1.

Reports

Reports can be written during the production of the magnetic tape file. These reports could be used to check the details that are produced. A number of reports can be created in the same run. The number can be limited by using the ADD_MAG_REP_FILES action parameter in the PAY_ACTION_PARAMETERS table.

Each report is accessed by using a prefix that denotes the file, for example, REPORT1_ to denote report number 1, REPORT2_to denote report number 2, and so on. If a report number is outside the range of the ADD_MAG_REP_FILES value, an invalid return error is reported. The report files are opened as and when needed with the names of the files previously described.

FastFormula Errors

Errors can be of three types:

- Payroll errors

  These are identified by a return of ERROR_PAY_<error_name>.

- Assignment errors

  These are denoted by ERROR_ASS_<error_name>.

- Warning errors

These are denoted by ERROR_WARN_<error_name>.

The actual messages themselves have to be prefixed with the assignment action id or payroll action id. This is done to insert the messages into the PAY_MESSAGE_LINES table. Warning messages are regarded as being at the assignment action level and require the assignment action id. If no id is supplied, the message is only written to the log file. No id must be supplied when running a magnetic tape report, since no actions exist for reports. Only payments have actions.

Example

Here are some examples of the format to use:

Error	Message	Meaning
ERROR_PAY_TEXT1	= '50122: Unexpected value'	- Payroll action id 50122 with message 'Unexpected Value'
ERROR_PAY_TEXT1	= ':Unexpected value'	- No payroll action id just a message
ERROR_ASS_TEXT1	= '56988: Unexpected value'	
ERROR_ASS_TEXT1	= 'Unexpected value'	
ERROR_WARN_TEXT1	= '56988: Unexpected value'	
ERROR_WARN_TEXT1	= ':Unexpected value'	

Error Handling

Magnetic tape either fully completes the process, or marks the whole run with a status of error.

Within this there are two types of errors:

- Payroll action level errors, which are fatal

  If this form of error is encountered, the error is reported and the process terminates.

- Assignment action level

  These can be set up in formulas and result in the error message being reported and the process continuing to run. This can be used to report on as many errors as possible during the processing so that they can be resolved before the next run.

The payroll action errors at the end of the run if assignment action level errors are encountered.

A description of the error message is written to the Log file. Also an entry is placed in the PAY_MESSAGE_LINES table if the action id is known.

Example PL/SQL

The following piece of PL/SQL code could be used to format a magnetic tape payment (drives off assignment actions). An alternative to writing a PL/SQL procedure would be to use the generic procedure and populate the batch magnetic tape tables.

> **Note:** This example only works for a business group of 'MAG Test GB' (the legislative formula is for GB only).

```
create or replace package body pytstm1
as
CURSOR get_assignments( p_payroll_action_id NUMBER)
 IS
 SELECT ppp.org_payment_method_id,ppp.personal_payment_method_id,
ppp.value, paa.assignment_id
 FROM pay_assignment_actions paa, pay_pre_payments ppp
 WHERE paa.payroll_action_id = p_payroll_action_id
 AND ppp.pre_payment_id = paa.pre_payment_id
 ORDER BY ppp.org_payment_method_id;
Also need to:
Test that the assignment are date effective?
Order by name or person_number or other ?
p_business_grp NUMBER;
--
--
PROCEDURE new_formula
IS
--
p_payroll_action_id NUMBER;
assignment      NUMBER;
p_org_payment_method_id NUMBER;
p_personal_payment_method_id NUMBER;
p_value NUMBER;
--
--
FUNCTION get_formula_id ( p_formula_name IN VARCHAR2)
   RETURN NUMBER IS
p_formula_id NUMBER;
BEGIN
 SELECT formula_id
 INTO p_formula_id
 FROM ff_formulas_f
 WHERE formula_name = p_formula_name
  AND (business_group_id = p_business_grp
   OR (business_group_id IS NULL
    AND legislation_code = 'GB')
  OR (business_group_id IS NULL AND legislation_code IS NULL)
);
--   RETURN p_formula_id;
--
END get_formula_id;
--
BEGIN
--
pay_mag_tape.internal_prm_names(1) :=
'NO_OF_PARAMETERS'; -- Reserved positions
pay_mag_tape.internal_prm_names(2) := 'NEW_FORMULA_ID';-- --
Number of parameters may be greater than 2 because formulas
may be -- keeping running totals.--
pay_mag_tape.internal_cxt_names(1)  := 'Number_of_contexts';
pay_mag_tape.internal_cxt_values(1) := 1;            --
Initial value---- IF NOT get_assignments%ISOPEN  THEN
-- New file--   pay_mag_tape.internal_prm_values(1) := 2;
pay_mag_tape.internal_prm_values(2) := get_formula_id
('REPORT_HEADER_1');--   if
pay_mag_tape.internal_prm_names(3) = 'PAYROLL_ACTION_ID'
  then p_payroll_action_id :=
```

```
to_number(pay_mag_tape.internal_prm_values(3));   end if;--
OPEN get_assignments (p_payroll_action_id);-- ELSE----
FETCH get_assignments INTO
p_org_payment_method_id,
p_personal_payment_method_id,           p_value,
assignment;--   IF get_assignments%FOUND THEN
-- New company
pay_mag_tape.internal_prm_values(1) := 2;
pay_mag_tape.internal_cxt_names(2)  := 'ASSIGNMENT_ID';
pay_mag_tape.internal_cxt_values(2) := assignment;
pay_mag_tape.internal_cxt_names(3)  := 'DATE_EARNED';
pay_mag_tape.internal_cxt_values(3) := to_char (sysdate,'DD-MON-YY
YY');
pay_mag_tape.internal_cxt_values(1) := 3;
pay_mag_tape.internal_prm_values(2) := get_formula_id
('ENTRY _DETAIL');
ELSE--       pay_mag_tape.internal_prm_values(1) := 2;
pay_mag_tape.internal_prm_values(2) := get_formula_id
('REPORT_CONTROL_1');
CLOSE get_assignments;
--  END IF;
--END IF;--
END new_formula;
BEGIN
-- 'MAG test BG' used as an example. The business group could be
-- retrieved using the payroll action id.
 select business_group_id
 into p_business_grp
 from per_business_groups
 where name = 'MAG test BG';
--END pytstm1;
```

Cheque Writer/Check Writer Process

Note: For ease, we refer to the Cheque Writer/Check Writer process as Cheque Writer throughout this technical essay.

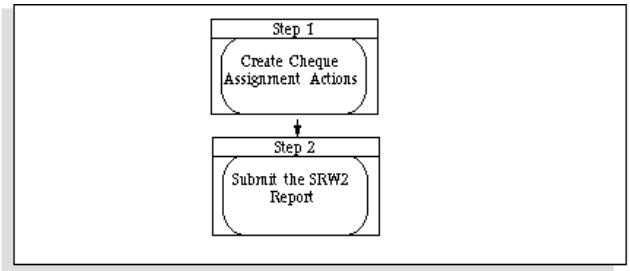
You run the Cheque Writer process to produce cheque payments for unpaid pre-payment actions. Before you run the process, you need to set up certain things, for example, the SRW2 report and the 'order by' option to sequence cheques (if required).

You run Cheque Writer through Standard Reports Submission (SRS). Unlike the Magnetic Tape process, you can have multiple threads in Cheque Writer.

The Process

The Cheque Writer process has two distinct steps:

Cheque Writer Steps



Step 1 - Create Cheque Assignment Actions

Cheque Writer creates cheque assignment actions for each of the target pre-payments, subject to the restrictions of the parameters specified. The target pre-payments must be unpaid-that is, never been paid-or if they have been paid, then voided.

Cheque Writer creates assignment actions in two stages:

1. Multiple threads insert ranges of assignment actions, which interlock back to previous actions.

   This happens in the same way as Pre-Payments and Magnetic Tape create assignment actions.

   See: The Process, page 2-111 (Pre-Payments)

   See: Running the Magnetic Tape Payments Process, page 2-113

2. A single thread runs through all the assignment actions in a specific order to update the chunk and cheque number.

   The order is specified by a PL/SQL procedure that you can customize. The thread divides the assignment actions equally into chunks, one chunk per thread. It assigns each action a cheque number.

   See: Using or Changing the PL/SQL Procedure, page 2-132

At this stage, the status of the assignment actions is 'Unprocessed'.

> **Note:** Cheque Writer creates an assignment action and cheque for each target pre-payment of the assignment. Consequently, a single Cheque Writer run can produce more than one cheque for a single assignment.

Step 2 - Submit SRW2 Report

When Cheque Writer has created the assignment actions and interlocks, each thread submits the specified SRW2 report as a synchronously spawned concurrent process. The reports produce files in a specific cheque format.

If the spawned concurrent process is successful, the status of the assignment actions is changed to 'Complete'. If the process fails, the status of the assignment actions is changed to 'In Error'. So, if you resubmit Cheque Writer, it can start at the point of submitting the report.

In this respect, Cheque Writer is similar to the magnetic tape process: the whole process must be successful before the payroll action is Complete. But, while the Magnetic Tape process can mark *individual* assignment actions In Error, Cheque Writer marks *all* assignment actions In Error.

Batch Process Parameters

The batch process has a number of parameters users can enter. The definition of the printer type (for example, laser or line printer for the report output) is not a parameter. The default for this is specified as part of the registration of the concurrent process for the report. Consult your *Oracle Applications System Administrators Guide* for more information on printers and concurrent programs.

- **payroll_id - optional**

  This parameter restricts the cheques generated according to the current payroll of the assignment. It is a standard parameter to most payroll processes.

- **consolidation_set_id - mandatory**

  This parameter restricts the target pre-payments for Cheque Writer to those which are for runs of that consolidation set.

- **start_date - optional**

  This parameter specifies how far back, date effectively, Cheque Writer searches for target pre-payments. If this parameter is not specified, Cheque Writer scans back to the beginning of time.

- **effective_date - optional**

  This parameter specifies the effective date for the execution of Cheque Writer. If it is null, the effective date is taken to be the effective date held in FND_SESSIONS. If there is no such row, then it is defaulted to SYSDATE.

- **payment_type_id - mandatory**

  This parameter specifies which payment type is being paid. For UK legislation, it must be a payment type which is of payment category Cheque. For US legislation, it must be a payment type which is of payment category Check.

- **org_payment_method_id - optional**

  This parameter restricts the target prepayments to those which are for that organization payment method. It would be used where different cheque styles are required by organization payment method.

- **order_by_option - mandatory**

  This parameter specifies which *order by* option is called to create and order the cheque assignment actions. By providing this as a parameter, the user can specify what ordering they want to take effect for the generated cheques.

- **report_name - mandatory**

  This parameter is the name of the SRW2 report that is synchronously spawned by Cheque Writer to generate the print file of cheques and any attached pay advices, and such.

  A user-extensible lookup is provided.

- **start_cheque_number - mandatory**

  This parameter specifies the contiguous range of numbers to be assigned to cheques generated.

- **end_cheque_number - optional**

  This parameter specifies the contiguous range of numbers to be assigned to cheques generated. If this parameter is specified, this range constrains how many cheque assignment actions are created. Cheque Writer is the only payroll action that does not necessarily process, what would otherwise be, all of its target actions.

  If the end number is not specified, Cheque Writer assigns numbers sequentially from the start number onwards for all generated cheque assignment actions.

  If cheques must be printed for different contiguous ranges (as may occur when using up the remnants of one box of cheque stationery, before opening another box), then the Cheque Writer process must be invoked separately for each contiguous range.

Cheque Numbering

The cheque stationery onto which the details are printed is typically authorized, and has the cheque number preprinted on it. It is common in the UK for there to be a further cheque number box which is populated when the cheque is finally printed. It is this number that the generating payroll system uses.

Usually, these two numbers are the same. It is not known whether any clearing system invalidates the cheque if they are not. However, it seems likely that if you need to trace the path of a cheque through a clearing system, the preprinted cheque number would prove most useful, and hence, it should be the number recorded for the cheque payment on the payroll system.

It is a user's responsibility to ensure that the cheque numbers used by Cheque Writer (and recorded on the system) are identical to those on the preprinted stationery. In certain circumstances, you might want to use numbers that are not the same. In this case, the cheque number recorded by the payroll system is simply a different cheque identifier from the preprinted cheque number.

> **Note:** Preprinted stationery usually comes in batches, for example, boxes of 10000. Therefore, you may want to use different ranges of cheque numbers when printing off cheques at the end of the pay period. For example, you may have to print off 2500 cheques using the remains of one box (numbered 9500 - 10000) and then an unopened box (numbered 20001 - 30000). Cheque Writer uses the start and end cheque number parameters to enforce these ranges.

Voiding and Reissuing Cheques

Under some circumstances, users might need to void a cheque and optionally issue a replacement. For example, an employee loses their cheque and requests a replacement, or you discover that the employee has previously left employment and should not have been paid. In both cases the first step is to void the cheque. This activity may also involve contacting the bank that holds the source account and cancelling the cheque.

> **Note:** Voiding a cheque does not prevent the payment from being made again.

Voiding and reissuing a cheque is different from rolling back and reprinting a cheque. You void a cheque when it has actually been issued and you need to keep a record of the voided cheque. You rollback when a cheque has not yet been issued. For example, during a print run your printer might jam on a single cheque and think it has printed more than one. These cheques have not been issued and the batch process should be rolled back and restarted for those actions.

Depending on the reason for voiding, a user may want to issue another cheque. This is known as 'reissuing'. This requires no extra functionality. The user has the choice of issuing a manual cheque and recording the details online, or of resubmitting the batch process for automatic printing.

You cannot reprocess actions that have already been paid. The process only creates payments for those actions that have never been paid, or have been voided.

Mark for Retry

Cheque Writer actions can be marked for retry. As with the rollback process, when marking a Cheque Writer payroll action for retry, the user can determine which assignment actions are to be marked by specifying an assignment set parameter.

Marking cheque assignment actions for retry does not remove the assignment actions, but simply updates their status to 'Marked For Retry' (standard behavior for all action types). The assigned cheque numbers are left unaltered. Hence, on retry, Cheque Writer generates a new print file.

The reason for this is that we cannot reassign cheque numbers for assignment actions of a cheque payroll action. The payroll action stores the start and end cheque numbers specified. If different ranges of numbers could be used on several retries of the payroll action, then some of its assignment actions could be assigned numbers outside the range held on the payroll action.

Rolling Back the Payments

If a user wants to assign new cheque numbers, they must rollback the Cheque Writer payroll and assignment actions, and submit a separate batch request.

> **Note:** It usually makes sense to roll back all of the cheques. If you mark individual cheques for retry, their cheque numbers are unlikely to be contiguous and it would be difficult to print these on the correct preprinted cheque stationery.

SRW2 Report

You may need to set up the format for the cheque stationery. The SRW2 report, invoked by Cheque Writer is passed in two parameters:

- payroll_action_id (of the cheque action)
- chunk number (to be processed)

For this purpose, the report must take the parameters named PACTID and CHNKNO.

By the time the report is run, the appropriate assignment actions have been created and cheque numbers assigned according to the order specified in the order by parameter.

The report must drive off the assignment actions for the cheque payroll action and chunk number specified. It must generate one cheque for each assignment action. The cheque number is held directly on the assignment action, while the amount to be paid is retrieved from the associated pre-payment.

The report must maintain the order of the cheques when printed out, the report must process the assignment actions in order of cheque number.

Example SELECT statement

The following select statement illustrates how to drive a report:

```
select to_number(ass.serial_number),

ass.assignment_action_id,

round(ppa.value,2),

ppf.last_name,
```

```
ppf.first_name

from per_people_f ppf,

per_assignments_f paf,

pay_assignment_actions ass,

pay_pre_payments ppa

where ass.payroll_action_id =:PACTID

and ass.chunk_number =:CHNKNO

and     ppa.pre_payment_id = ass.pre_payment_id

and     ass.assignment_id = paf.assignment_id

and     ass.status <>'C'

and     paf.person_id = ppf.person_id

order by  to_number(ass.serial_number)
```

Registering the Report

Once the SRW2 report is written, you must register it as a Cheque Writer report. This is similar to registering 'Cash Analysis Rules' for the Pre-Payments process.

You must also define a new Lookup Value for the Type of 'CHEQUE_REPORT'. Enter the report name and description.

In a similar way to the Magnetic Tape process, the file generated by the report is named:

p<trunc(conc_request_id,5)>.c<chunk_number>

The file name is padded with zeros if the length of the request id is shorter than five characters, for example, p03451.cl.

It is written to the $APPLCSF/$APPLOUT directory, if $APPLCSF is defined, and otherwise to $PAY_TOP/$APPLOUT.

If Cheque Writer is run with multiple threads, it produces several files. This is because Cheque Writer assignment actions are split into several chunks, one chunk per thread. So, each thread can pick a chunk and process it. This is done to improve performance on machines with multiple processors. For example, if there are four threads processing, there would be four files produced:

- p03451.c1
- p03451.c2
- p03451.c3
- p03451.c4

Cheque Writer creates a fifth file (by the process that concatenates the four files into one). The name of this file is p03451.ch.

Using or Changing the PL/SQL Procedure

Cheque Writer updates the assignment actions with the cheque and chunk number in the sequence determined by a PL/SQL procedure, called anonymously from the process. A default PL/SQL procedure is provided with the generic product - pay_chqwrt_pkg.chqsql.

The default sort order is:

1. Organization
2. Department
3. Surname
4. First name

You can change this procedure to set up several different sorting orders by criteria, denoted by a flag passed to the procedure. You should copy the core select statement, and alter the subquery to order according to your own business needs.

The advantage of giving access to the whole SQL statement is that the cheques can be ordered by any criteria. If we had only allowed specification of an ORDER BY clause, then the ordering would have been restricted to attributes on those tables already in the FROM clause of the core SQL statement.

To set up new order by requirements, change the pay_chqwrt_pkg.chqsql package procedure. You could add the following IF statement when checking the procname variable:

```
else if procname = 'NEW ORDER BY' then

   sqlstr := 'select ....'
```

The select statement could be a copy of the existing select statement but with the order by clause changed. The select statement must return the assignment action's rowid.

Based on this information the assignment action can be given a serial/cheque number and assigned to a chunk.

Similarly, as with the SRW2 report the new order by option has to be registered before it can be used. This is done in a similar manner except that the Lookup Type is CHEQUE PROCEDURE. Enter a meaningful description in the Meaning field and the name of the option, for example NEW ORDER BY, in the Description field.

Cash Process

The Cash process indicates to the system that payment has been made, and prevents pre-payments from being rolled back.

Note: This is a UK-only process.

Payroll Action Parameters

Payroll action parameters are system-level parameters that control aspects of the Oracle Payroll batch processes. It is important to recognize that the effects of setting values for specific parameters may be system wide. The text indicates where parameters are related to specific processes. For some parameters you should also understand the concept of array processing and how this affects performance.

Action Parameter Values

Predefined values for each parameter are supplied with the system, but you can override these values as part of your initial implementation and for performance tuning.

Action parameter values are specified by inserting the appropriate rows into the following table: **PAY_ACTION_PARAMETERS**, which has two columns:

```
PARAMETER_NAME   NOT NULL VARCHAR2(30)

PARAMETER_VALUE  NOT NULL VARCHAR2(80)
```

The payroll batch processes read values from this table on startup, or provide appropriate defaults, if specific parameter values are not specified.

Summary of Action Parameters

The following list shows user enterable action parameters and values with any predefined default value.

Note: Case is significant for these parameters.

Parameter	Value	Default
ADD_MAG_ REP_FILES	1 or more	4
BAL BUFFER SIZE	1 or more	30
CHUNK SHUFFLE	Y or N	N
CHUNK_SIZE	1 - 16000	20
EE BUFFER SIZE	1 or more	40
LOG_AREA	See later	
LOG_ASS IGN_END	See later	
LOG_ASS IGN_START	See later	
LOGGING	See later	
MAX_ ERRORS_ ALLOWED	1 or more	CHUNK_SIZE or 20 (if no chunk size)
MAX_S INGLE_UNDO	1 or more	50
RR BUFFER SIZE	1 or more	20
RRV BUFFER SIZE	1 or more	30
COST BUFFER	1 or more	20
THREADS	1 or more	1
TRACE	Y or N	N
USER_ MESSAGING	Y or N	N

Note: All parameter names without underscores also have an alias with underscores (except CHUNK SHUFFLE).

Parallel Processing Parameters

THREADS

Parameter Name: THREADS

Parameter Value: 1 or more

Default Value:1

Oracle Payroll is designed to take advantage of multiprocessor machines. This means that you can improve performance of your batch processes by splitting the processing into a number of 'threads'. These threads, or sub-processes, will run in parallel.

When you submit a batch process to a concurrent manager the THREADS parameter determines the total number of sub-processes that will run under the concurrent manager. The master process will submit (THREADS - 1) sub-processes.

Set this parameter to the value that provides optimal performance on your server. The default value, 1, is set for a single processor machine. Benchmark tests on multiprocessor machines show that the optimal value is around two processes per processor. So, for example, if the server has 6 processors, you should set the initial value to 12 and test the impact on performance of variations on this value.

> **Important:** The concurrent manager must be defined to allow the required number of sub-processes to run in parallel. This is a task for your Applications System Administrator.

CHUNK_SIZE

Parameter Name: CHUNK_SIZE

Parameter Value: 1 - 16000

Default Value: 20

Size of each commit unit for the batch process. This parameter determines the number of assignment actions that are inserted during the initial phase of processing and the number of assignment actions that are processed at one time during the main processing phase.

> **Note:** This does not apply to the Cheque Writer/Check Writer, Magnetic Tape or RetroPay processes.

During the initial phase of processing this parameter defines the array size for insert. Large chunk size values are not desirable and the default value has been set as a result of benchmark tests.

Each thread processes one chunk at a time.

Array Select, Update and Insert Buffer Size Parameters

The following parameters control the buffer size used for 'in-memory' array processing. The value determines the number of rows the buffer can hold.

> **Note:** These parameters apply to the *Payroll Run* process only.

When you set values for these parameters you should note that there is a trade-off between the array size, performance and memory requirements. In general, the greater the number of rows fetched, updated or inserted at one time, the better the performance. However, this advantage declines at around 20.

Therefore, the improvement between values 1 and 20 is large, while between 20 and 100 it is small. Note also that a higher value means greater memory usage. For this reason, it is unlikely that you will gain any advantage from altering the default values.

CHUNK_SIZE

Parameter Name: CHUNK_SIZE

Parameter Value: 1 - 16000

Default Value: 20

Size of each commit unit for the batch process. As before.

RR BUFFER SIZE

Parameter Name: RR BUFFER SIZE

Parameter Value: 1 or more

Default Value: 20

Size of the *Run Result* buffer used for array inserts and updates: one row per Run Result.

RRV BUFFER SIZE

Parameter Name: RRV BUFFER SIZE

Parameter Value: 1 or more

Default Value: 30

Size of the *Run Result Value* buffer used for array inserts and updates: one row per Run Result Value. Typically this will be set to (RR BUFFER SIZE * 1.5).

BAL BUFFER SIZE

Parameter Name: BAL BUFFER SIZE

Parameter Value: 1 or more

Default Value: 30

Size of the *Latest Balance* buffer used for array inserts and updates: 1 row per Latest Balance.

EE BUFFER SIZE

Parameter Name: EE BUFFER SIZE

Parameter Value: 1 or more

Default Value: 40

Size of the buffer used in the initial array selects of Element Entries, Element Entry Values, Run Results and Run Result Values per assignment.

Costing Specific Parameters

COST BUFFER SIZE

Parameter Name: COST BUFFER SIZE

Parameter Value: 1 or more

Default Value: 20

Size of the buffer used in the array inserts and selects within the Costing process.

Magnetic Tape Specific Parameters

ADD_MAG_REP_FILES

Parameter Name: ADD_MAG_REP_FILES

Parameter Value: 1 or more

Default Value: 4

The maximum number of additional audit or report files the magnetic tape process can produce.

Error Reporting Parameters

In every pay cycle you would expect some errors to occur in processing individual assignments, especially in the Payroll Run. These errors are usually caused by incorrect or missing data in the employee record. For practical reasons, you would not want the entire run to fail on a single assignment failure. However, if many assignments generate error conditions one after the other, this will usually indicate a serious problem, and you will want to stop the entire process to investigate the cause. For processes that support assignment level errors you can use the MAX_ERRORS_ALLOWED parameter to control the point at which you want to stop the entire process to investigate these errors.

The processes that use this feature are:

- Payroll Run
- Pre-Payments
- Costing
- Rollback

MAX_ERRORS_ALLOWED

Parameter Name: MAX_ERRORS_ALLOWED

Parameter Value: 1 or more

Default Value: CHUNK_SIZE or 20 (if no chunk size)

The number of consecutive actions that may have an error before the entire process is given a status of 'Error'.

Rollback Specific Parameters

Rollback of specific payroll processes can be executed in two ways. A batch process can be submitted from the Submit Requests window. Alternatively, you can roll back a specific process by deleting it from the Payroll Process Results window or the Assignment Process Results window. When you roll back from a window this parameter controls the commit unit size.

MAX_SINGLE_UNDO

Parameter Name: MAX_SINGLE_UNDO

Parameter Value: 1 or more

Default Value: 50

The maximum number of assignment actions that can be rolled back in a single commit unit when rollback is executed from a form. Although you can change the default limit, you would usually use the Rollback process from the SRS screen if it is likely to be breached.

Payroll Process Logging

During installation and testing of your Oracle Payroll system you may need to turn on the detailed logging options provided with the product. Use the **LOGGING** parameter to provide a large volume of detailed information that is useful for investigating problems.

Detailed logging options should only be switched on when you need to investigate problems that are not easily identified in other ways. The logging activities will have an impact on the overall performance of the process you are logging. Usually, this feature is needed during your initial implementation and testing before you go live. In normal operation you should switch off detailed logging.

> **Important:** If you need to contact Oracle Support for assistance in identifying or resolving problems in running your payroll processes, you should prepare your log file first. Define the Logging Category, Area and range of Assignments and then resubmit the problem process.

Logging Categories

Logging categories define the type of information included in the log. This lets you focus attention on specific areas that you consider may be causing a problem. You can set any number of these by specifying multiple values:

- **G** General (no specific category) logging information

  Output messages from the PY_LOG macro for general information. This option does not sort the output and you should normally choose a list of specific categories.

- **M** Entry or exit routing information

  Output information to show when any function is entered and exited, with messages such as 'In: pyippee', 'Out : pyippee'. The information is indented to show the call level, and can be used to trace the path taken through the code at function call level. Often, this would be useful when attempting to track down a problem such as a core dump.

- **P** Performance information

  Output information to show the number of times certain operations take place at the assignment and run levels and why the operation took place. For example, balance buffer array writes.

- **E** Element entries information

  Output information to show the state of the in-memory element entry structure, after the entries for an assignment have been fetched, and when any item of the structure changes; for example, addition of indirects or updates. This also shows the processing of the entry.

- **L** Balance fetching information

  Output information to show the latest balance fetch and subsequent expiry stage.

- **B** Balance maintenance information

Output information to show the creation and maintenance of in-memory balances

- **I** Balance output information

  Output information to show details of values written to the database from the balance buffers.

- **R** Run results information

  Output information to show details of run results and run result values written to the database from the Run Results or Values buffer.

- **F** Formula information

  Output information to show details of formula execution. This includes formula contexts, inputs and outputs.

- **C** C cache structures information.

  Output information to show details of the payroll cache structures and changes to the entries within the structure.

- **Q** C cache query information

  Output information to show the queries being performed on the payroll cache structures.

- **S** C Cache ending status information

  Output information to show the state of the payroll cache before the process exits, whether ending with success or error. Since much of the logging information includes id values, this can be used to give a cross reference where access to the local database is not possible.

- **V** Vertex (available to US and Canadian customers only)

  Output information to show the values being passed in and out of the Vertex tax engine.

  This option also creates a separate file in the Out directory showing the internal settings of the engine.

Logging Parameters

LOGGING

Parameter Name: LOGGING

Parameter Value: G, M, P, E, L, B, I, R, F, C, Q, S, V

Default Value: No logging

LOG_AREA

Parameter Name: LOG_AREA

Parameter Value: Function to start logging

Default Value: No default

LOG_ASSIGN_START

Parameter Name: LOG_ASSIGN_START

Parameter Value: Assignment to start logging

Default Value: All assignments

LOG_ASSIGN_END
Parameter Name: LOG_ASSIGN_END

Parameter Value: Assignment to end logging, including this one

Default Value: All assignments

Output Log File

When you enable the logging option the output is automatically included in the log file created by the concurrent manager. You can review or print the contents of this log file.

Except for the General category, the log file will contain information in a concise format using id values. This keeps the size of the log file to a minimum while providing all the technical detail you need.

To help you understand the output for each logging category, other than 'G' and 'M', the log file contains a header indicating the exact format.

Miscellaneous Parameters

USER_MESSAGING
Parameter Name: USER_MESSAGING

Parameter Value: Y/N

Default Value: N

Set this to parameter to 'Y' to enable detailed logging of user readable information to the pay_message_lines table. This information includes details about the elements and overrides that are processed during the Payroll Run.

> **Note:** This information is useful when you are investigating problems, but you may find that it is too detailed for normal working.

TRACE
Parameter Name: TRACE

Parameter Value: Y/N

Default Value: N

Set this parameter to 'Y' to enable the database trace facility. Oracle trace files will be generated and saved in the standard output directory for your platform.

> **Warning:** Use the trace facility only to help with the investigation of problems. Setting the value to 'Y' causes a significant deterioration in database performance. If you experience a significant problem with the performance of your payroll processes, check that you have reset this parameter to the default value - 'N'.

System Management of QuickPay Processing

When users initiate a QuickPay run or a QuickPay prepayments process, the screen freezes until the process finishes. QuickPay is set up to manage any cases in which the concurrent manager fails to start the process within a specified time period, or starts it but fails to complete it within the specified period. This situation can sometimes arise when, for example, many high priority processes hit the concurrent manager at the same time.

The system's management of the screen freeze occurring when a user initiates a QuickPay process involves:

- Checking the concurrent manager every few seconds for the process completion.
- Unfreezing the screen and sending an error message to the user when the process has not completed within a maximum wait time.

  The error message includes the AOL concurrent request ID of the process. The user must requery the process to see its current status.

System administrators can improve the speed of QuickPay processing at their installation by:

- Changing the default for the interval at which checks for process completion occur.

  By default, the check of the concurrent manager occurs at 2 second intervals. The parameter row QUICKPAY_INTERVAL_WAIT_SEC in the table PAY_ACTION_PARAMETERS sets this default.
- Changing the default for the maximum wait time.

  The maximum wait time allowed for a QuickPay process to complete defaults to 300 seconds (5 minutes), after which the system issues an error message. The parameter row QUICKPAY_MAX_WAIT_SEC in the PAY_ACTION_PARAMETERS table sets this default.
- Defining a new concurrent manager exclusively for the QuickPay run and prepayments processes.

To change the defaults for the interval at which checks occur or for the maximum wait time:

- Insert new rows (or update existing rows) in the table PAY_ACTION_PARAMETERS.

Notice that QUICKPAY_INTERVAL_WAIT_SEC and QUICKPAY_MAX_WAIT_SEC are codes for the Lookup type ACTION_PARAMETER_TYPE.

To define a new concurrent manager exclusively for the two QuickPay processes:

1. Exclude the two QuickPay processes from the specialization rules for the standard concurrent manager.
2. Include them in the specialization rules for the new QuickPay concurrent manager to be fewer than those of the standard concurrent manager. Doing so reduce the time it takes to start requests for the QuickPay processes.

Assignment Level Interlocks

When you process a payroll, you run a sequence of processes that each perform an action on the assignments.

The sequence in which you run the processes is critical to the success of processing, as each process uses, and builds upon, the results of the previous process in the

sequence. The sequence of the processing is also determined by issues of data integrity. For example, the Pre-Payments process (which prepares the payments according to the payment methods) uses the results of the Payroll Run process (which calculates the gross to net payment).

It is essential for correct payments that the results cannot be changed without also changing the prepayment results. To prevent this from occurring (and for data integrity), Oracle Payroll uses assignment level interlock rules.

Action Classifications

The payroll processes (such as Payroll Run and Costing) and action types (such as QuickPay) are classified as Sequenced or Unsequenced. The action classification determines how interlock processing rules are applied.

Processes and Action Types	Classification	Insert Interlock Rows?
Payroll Run	Sequenced	No
QuickPay	Sequenced	No
Reversal	Sequenced	Yes
Balance Adjustment	Sequenced	No
Balance Initialization	Sequenced	No
Pre-Payments	Unsequenced	Yes
QP PrePayments	Unsequenced	Yes
Ext/Manual Payments	Unsequenced	Yes
Magnetic Tape Transfer	Unsequenced	Yes
Advance Pay	Sequenced	No
Cheque Writer	Unsequenced	Yes
Cash	Unsequenced	Yes
Costing	Unsequenced	Yes
Transfer to GL	Unsequenced	Yes
Retropay by Action	Sequenced	No
Retropay by Aggregate	Sequenced	No

Sequenced Actions

These actions exist at the same level and must be processed in strict sequence, for example, Payroll Run before QuickPay. The general rule is that you cannot insert a sequenced action for an assignment if there is another sequenced action in the future, or if there is an incomplete sequenced action in the past.

There are exceptions for Process Reversal and Balance Adjustment. And, there may be specific legislative requirements that have implications for this rule. For more information, see Pay Period Dependent Legislation, page 2-143.

The sequence rule uses the effective date of the payroll action. If there is more than one action with the same effective date, the action sequence number determines the sequence of processing.

Unsequenced Actions

You can insert unsequenced actions for an assignment even when there are other assignment actions for that assignment in the future or in the past. For example, you can run the Costing process before or after you run the PrePayments process.

Pay Period Dependent Legislation

The rules that govern the calculation of tax for employees with multiple assignments vary between legislations, and this determines how the rules for interlocking are applied.

For example, in the UK when you calculate tax, you must take account of all earnings for all assignments in a pay period. For this type of legislation, the interlock rules check the sequence of actions for all assignments and a failure on one assignment in a pay period may be caused by an action that applies to another assignment.

For example, if you process an employee who is on both a monthly and a weekly payroll, you cannot roll back the monthly pay run for that employee if you have subsequently processed and paid them on the weekly payroll. You would have to roll back the payments process for the weekly assignment before you could roll back their monthly payroll action.

In other legislations, for example in the US, each assignment is considered separately and interlock failure for one assignment does not cause failure for any others.

Action Interlock Rows

When interlocks are inserted for an assignment action, they lock the action that is being processed. For example, a pre-payment interlock points to the payroll run action to be paid, thus locking the run from being deleted. The existence of a sequenced action prevents the insertion of sequenced actions prior to that action. That is, sequenced actions have to happen in order.

Checking for Marked For Retry Actions

There is one special rule for assignment actions that are marked for retry. If you attempt to retry a Payroll Run or QuickPay action, the system checks there are no sequenced assignment actions marked for retry existing in the past for any assignments (or people, in some legislations) that you are attempting to process.

Specific Rules for Sequenced Actions

An assignment action is not inserted if any of the following situations exist:

- There is an incomplete sequenced action for the assignment with a date on or before the insertion date
- There is a sequenced action for the assignment with any action status, at a date after the insertion date
- There is a non removable action at a date after the insertion date

There are two exceptions:

- Reversal
- Balance Adjustment.

When a reversal or balance adjustment is inserted, the system maintains the action sequence by changing the action sequence numbers for any assignment actions that exist later in the pay period.

Specific Rules for Unsequenced Actions

An unsequenced assignment action is not inserted if there is an interlock for the assignment action currently being processed from another unsequenced assignment action.

For example, if we had performed a QuickPay followed by a QuickPay Pre-Payment, a subsequent Pre-Payments process would not insert an assignment action/interlock to the QuickPay. This is because the QuickPay Pre-Payment would have inserted an action and an interlock, and Pre-Payments has the same action classification.

Rules For Rolling Back and Marking for Retry

This table summarizes the rules for retry and rollback of payroll and assignment actions. For some processes, you cannot roll back actions only for an individual assignment. For example you cannot roll back an individual from the Magnetic Transfer process. This process actually produces the magnetic tape file so you must roll back the whole process, and then redo it.

Action Type Name	Payroll Action - Retry	Payroll Action - Rollback	Assignment Action - Retry	Assignment Action - Rollback
Payroll Run	Yes	Yes	Yes	Yes
QuickPay	Yes	Yes	Yes	No
Reversal	No	Yes	No	No
Balance Adjustment	No	Yes	No	No
Balance Initialization	No	Yes	No	No
Purge	Yes	No	No	No
Pre-Payments	Yes	Yes	Yes	Yes
QP PrePayments	Yes	Yes	Yes	No
Ext/Manual Payment	No	Yes	No	No
Magnetic Tape Transfer	Yes	Yes	No	Yes
Cheque Writer	Yes	Yes	Yes	Yes
Cash	No	Yes	No	Yes
Costing	Yes	Yes	Yes	Yes
Transfer to GL	Yes	Yes	No	No
Advance Pay	Yes	Yes	Yes	Yes
Retropay by Aggregate	Yes	Yes	Yes	Yes
Retropay by Action	Yes	Yes	Yes	Yes

Rolling Back Sequenced Actions

You cannot roll back a sequenced action if there is a later sequenced action for the assignment, except for Balance Adjustments or Reversals. For example, you cannot roll back a payroll run in one period, if you have already processed another payroll run in the next pay period.

Marking Actions For Retry

You cannot mark a sequenced action for retry if there is a later sequenced action for the assignment, except for Balance Adjustments or Reversals. However, you can do this if the future action causing the lock is itself marked for retry.

You can retry an unsequenced action if the locking action is itself marked for retry.

Transfer to the General Ledger Process

After you have run the post-run process *Costing* (which accumulates costing results), you are ready to transfer the results to the General Ledger or other systems.

This process can be submitted using multiple threads, in the same way as the Payroll Run.

Costing Process

After running the payroll processes, you start the post-run process, *Costing*. The Costing process accumulates results for transfer to the General Ledger and other applications. This process sorts the run results in accordance with the information you have selected from the Cost Allocation flexfield at all levels, by the following:

- Company
- Set of Books
- Cost Center
- General Ledger
- Labour Distribution Accounts

Examples of the cost allocation of payroll results and of the distribution of employer charges over selected employee earnings appear in the following table.

If your installation also includes Oracle General Ledger, run the Transfer to the General Ledger process after you have run the Costing process. This transfers the results from the Costing process to Oracle General Ledger.

Example of Payroll Costs Allocation

The following table displays payroll run results for four employees, using accounts and work structures identified using the Cost Allocation key flexfield. The example *Costing Process Results* table illustrates how the Costing process allocates these payroll results to:

- Accounts and cost centers for the General Ledger
- Accounts for cost centers and product lines within cost centers, for labour distribution purposes

Sample Payroll Results

Employee	Cost Center	Product Line	Salary	Wages	Overtime	Union Dues
Employee 1	Production	H201 100%		1,000	400	20
Employee 2	Sales	H305 100%	1,500			
Employee 3	Production	H201 50% H202 50%		2,000	600	30
Employee 4	Sales	H305 20% H310 40%	1,000			

The following table illustrates the allocation of costs from these sample run results.

Example Costing Process Results

Account Code	Production	Sales	H201	H202	H305	H307	H310
Salaries		2,500			1,700	400	E400
Wages	3,000		2,000	1,000			
Overtime	1,000		700	300			
Union Dues Liability	50						

Example Costing Process Results (continued)

Account Code	Results
Clearing	Account contains balancing credits for earnings Salary, Wages and Overtime, and balancing debits for deduction Union Dues

Example of Employer Charge Distribution

When you give links for elements representing employer charges and the costable type Distributed, the Costing process distributes the employer charges as overhead for each employee over a set of employees' earnings. This example shows how employer payments totalling 100 dollars are distributed over a set of earnings including wages and overtime, for the cost center Production and the product lines H201 and H202.

Overhead Distribution for the Production Cost Center

```
Total paid to Production Cost Center as Wages run result: $3,000.
00

Total paid to Production Cost Center as Overtime run result: $1,0
00.00

Total for Earnings types specified for Distribution: $4,000.00

Ratio for Wages distribution, Production Cost Center = 3000/4000
= .75

Wages overhead = Pension Charge 100 x .75 = 75.00

Ratio for Overtime distribution, Production Cost Center = 1000/40
00 = .25

Overtime overhead = Pension Charge 100 x .25 = 25.00
```

Overhead Distribution for the Product Lines H210 and H202

```
Total paid for Product Line H201 as Wages run result: $2,000.00

Total paid for Product Line H202 as Wages run result: $1,000.00
```

```
Total paid for Product Lines H201 and H202 as Wages: $3,000.00

Ratio for Wages distribution, Product Line H201 = 2000/3000 = 0.6
667

Product Line H201 overhead = Total Wages overhead $75 x .6667 = $
50.00

Ratio for Wages distribution, Product Line H202 = 1000/3000 = 0.3
334

Product Line H202 overhead = Total Wages overhead $75 x .3334 = $
25.00

Total paid for Product Line H201 as Overtime run result: $700.00

Total paid for Product Line H202 as Overtime run result: $300.00

Total paid for Product Lines H201 and H202 as Overtime: $1,000.00

Ratio for Overhead distribution, Product Line H201 = 700/1000 = .
7

Product Line H201 overhead = Total Overtime overhead $25 x .7 = $
17.50

Ratio for Overhead distribution, Product Line H202 = 300/1000 = 0
.3

Product Line H202 overhead = Total Overtime overhead $25 x .3 = $
7.50
```

Table: Distribution of Overhead Over Cost Center and Production Line Totals

Account Code	Cost Center - Production	Product Line H201	Product Line H202
Wages	3,000	2,000	1,000
Employer Liability Distribution	75	50	25
Overtime	1,000	700	300
Employer Liability Distribution	25	17.50	7.50

The Payroll Archive Reporter (PAR) Process

Using the Payroll Archive Reporting (PAR) process, you can produce complex payroll reports on employee assignments on a periodic basis, for example at the end of the tax year, or for each tax quarter. You can submit these reports to a tax authority or other governmental body using magnetic tape.

If necessary, you can archive the data reported on exactly as it appears in the reports. This covers the possibility that the payroll department, or external authorities receiving the reports, may need to review the data at some future time.

If archiving is not required, you can still retain a record of the production of the reports and which employee assignments were included in them.

The primary use of the PAR process is for magnetic tape reporting, but you can also use it (in Archive mode) for reports delivered using Oracle Report Writer.

The generic PAR process described here may not meet the payroll reporting requirements of all HRMS payroll localizations. Therefore your localization team may have made changes such as extending the data reported on to include payroll actions, payrolls, or organizations.

PAR Modes

To support flexibility in its use, PAR can be run in three different modes:

- Magnetic Tape with Archive

  In this mode, PAR archives the values needed for reporting in the FastFormula archive tables (FF_ARCHIVE_ITEMS and FF_ARCHIVE_ITEM_CONTEXTS). It then produces a report on magnetic tape based on the archived values.

- Archive

  In this mode, PAR only archives values needed for reporting in the FastFormula archive tables.

  Having run the PAR process in Archive mode, you can extract data from the FastFormula archive tables using either Oracle Report Writer or a magnetic tape process.

- Magnetic Tape without Archive

  In this mode, PAR produces a report on magnetic tape and maintains a record of the report production (in the table PAY_PAYROLL_ACTIONS) and/or records of the individual assignments reported on (in the table PAY_ASSIGNMENT_ACTIONS).

  > **Note:** When you produce magnetic tape reports using the alternative process PYUMAG, there is no record of the report production.

Notice that running PAR in Archive mode and then in Magnetic Tape without Archive mode is convenient if you need to produce a number of reports by magnetic tape, each of which requires a subset of a large set of data. All the data can be archived at once in Archive mode, and then the individual reports can be produced for magnetic tape delivery in Magnetic Tape without Archive mode.

Overview of the PAR Process

The PAR process operates as follows:

1. It creates a payroll action with associated assignment actions. In these actions, PAR code evaluates live database items (that is, items that point to live tables) representing the data needed for a payroll report. The PAR code uses contexts for the database items as necessary.
2. When run in the Archiver or Magnetic Tape with Archiver modes, PAR then stores the results of the database evaluations in the FastFormula archive tables (FF_ARCHIVE_ITEMS and FF_ARCHIVE_ITEM_CONTEXTS).

3. When run in the Magnetic Tape with Archiver or Magnetic Tape without Archiver modes, PAR code retrieves values from the archive tables by evaluating archive database items, and includes the values in reports delivered by magnetic tape.

Overview of the Setup Steps

To set up the PAR process

1. Decide on the employee data to report on and to archive, and the formatting of the reports.
2. Create the archive and live database items that are needed to produce the data in the reports, setting contexts for them as necessary.

   See: Create Database Items for Archiving, page 2-150
3. For Archive mode or Magnetic Tape with Archive mode, write formulas that determine which database items are to be archived. For Magnetic Tape with Archiver and Magnetic Tape without Archiver modes, write formulas that format strings as required by tape formats, and provide error and warning messages to users.

   See: Write Formulas, page 2-153
4. Write package procedures that determine the assignments and assignment actions for PAR to process for the reports.

   See: Write Package Procedures for Assignments and Assignment Actions, page 2-153
5. Provide an SRS (Standard Report Submission) definition from which users can launch the PAR process.

   See: Provide an SRS Definition for the PAR Process, page 2-154
6. Identify your custom reports, formulas and package procedures to the system by making the appropriate entries in the table PAY_REPORT_FORMAT_MAPPINGS_F.

   See: Populate Rows in the PAY_REPORT_FORMAT_MAPPINGS_F Table, page 2-155

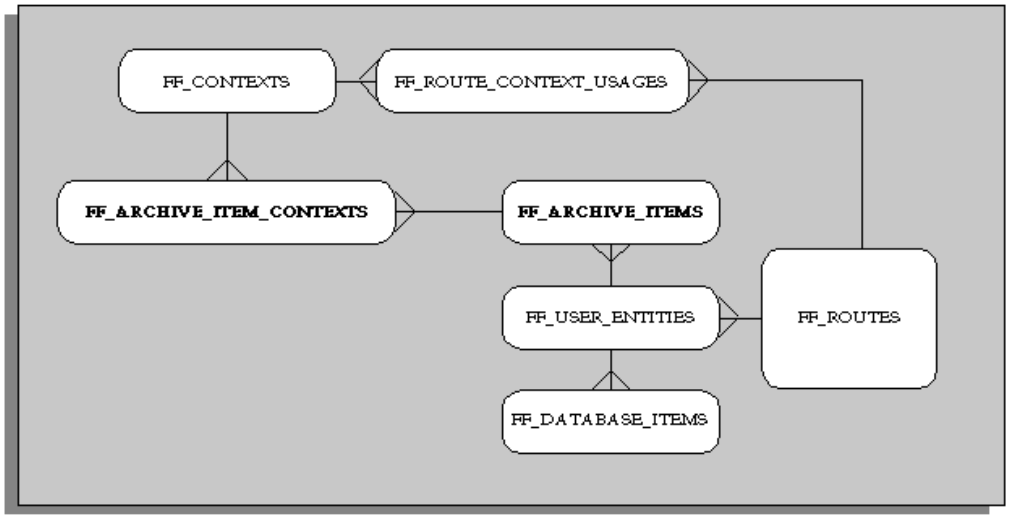
Create Database Items for Archiving

For its archiving function, PAR uses both live database items (which point at live tables), and archive database items (which point at the archive tables to retrieve archived data). For each archive database item, there must be a corresponding live database item. You are responsible for creating the archive database items, and for any live database items you need that do not already exist.

For example, for the archive database item A_INCOME_TAX_YTD referenced in a formula, there must be a live database item INCOME_TAX_YTD. PAR runs this live database item and places the value in the archive table FF_ARCHIVE_ITEMS.

Archive Database Item Creation: Background

The entity relationship diagram below shows the relationship of the PAR tables to other tables in generic HRMS:



The FF_ARCHIVE_ITEMS table records a snapshot of what particular database items evaluate to on a run of PAR.

The creation of archive database items includes the creation of archive routes. You define these in FF_ROUTES, with definition texts that are simple select statements from the two tables FF_ARCHIVE_ITEM_CONTEXTS and FF_ARCHIVE_ITEMS. Notice however that you must define these based on the number of contexts being passed into the routes, and the data type of the contexts. There are however, seeded Archive Routes, which you may be able to make use of rather than defining your own; these are detailed in the next section.

You define the route context usages in the table FF_ROUTE_CONTEXT_USAGES. The recommended way to do this is to retrieve from FF_CONTEXTS the context IDs that the live and archive routes require, and then define new route context usages based on the new archive routes. The route parameter is always defined based on the new archive route and a parameter name of User Entity ID.

Here is an example of a more complex archive route:

```
l_text := 'ff_archive_items target,
 ff_archive_item_contexts fac,
 ff_archive_item_contexts fac1
where target.user_entity_id = &U1
and target.context1 = &B1 /* context assignment action id */
and fac.archive_item_id = target.archive_item_id
and fac.context = to_char(&B2) /* 2nd context of source_id */
and fac1.archive_item_id = target.archive_item_id
```

The simple structure underlying this relatively complex route is still evident. Each context added just represents a further join to FF_ARCHIVE_ITEM_CONTEXTS.

Seeded Generic Archive Routes

The seeded generic archive routes fall into two categories: routes that have only one context (using ASSIGNMENT_ACTION_ID) and routes that have two contexts.

Routes with One Context

For the generic archive routes with one context, three datatypes are supported for that context, and therefore three such routes are automatically created when you run the automatic database item generator:

- A Character Context route, mapping onto a FF_CONTEXT of datatype 'T' (Text). This is named ARCHIVE_SINGLE_CHAR_ROUTE.
- A Numeric Context route, mapping onto a FF_CONTEXT of datatype 'N' (Number). This is named ARCHIVE_SINGLE_NUMBER_ROUTE.
- A Date Context route, mapping onto a FF_CONTEXT of datatype 'D' (Date). This is named ARCHIVE_SINGLE_DATE_ROUTE.

Here is the text for ARCHIVE_SINGLE_CHAR_ROUTE:

```
ff_archive_items target
    where target.user_entity_id = &U1
    and target.context1 = &B1
```

Routes with Two Contexts

For the generic archive routes that have two contexts, the automatic database item generator references the table FF_ARCHIVE_ITEM_CONTEXTS, whose column CONTEXT is stored as a Varchar2(30). It makes the assumption that the first context stored in FF_ARCHIVE_ITEMS is a number, and is an assignment action ID. It can seed only one such 'two-context archive route' by decoding the where clause of the generic archive route as follows:

```
ff_archive_items target,
ff_archive_item_contexts context
ff_contexts ffc
where target.user_entity_id = &U1
and target.context1 = &B1
and target.archive_item_id = context.archive_item_id
and ffc.context_id = context.context_id
and context.context = decode(ffc.data_type,'T', &B2, 'D', fnd_date
.date_to_canonical(&B2),
to_char(&B2));
```

Running the Archive Database Item Generator

You make several calls to the procedure for running the interface to the archive database item generator, one for each of the database items that you want to archive. The procedure is as follows:

```
procedure pay_archive_utils.create_archive_dbi(
  p_live_dbi_name  IN VARCHAR2(30),
  p_archive_route_name IN VARCHAR2(30) DEFAULT NULL,
  p_secondary_context_name IN VARCHAR2(30));
```

Contexts for Database Items

Using the standard set_context procedure, you set global contexts or assignment level contexts for those database items that require contexts. INITIALIZATION_CODE sets the global contexts for formulas, for example, PAYROLL_ID. ARCHIVE_CODE sets the context for the assignment level contexts, such as ASSIGNMENT_ID.

See: Examples: INITIALIZATION_CODE and ARCHIVE_CODE, page 2-157.

Write Formulas

To run PAR in Archive or Magnetic Tape with Archive mode, you write formulas that identify the database items used in the archiving process. To run PAR in Magnetic Tape with Archive or Magnetic Tape without Archive modes, you must write formulas to format strings as required, and to provide warnings and errors.

The PAR process uses the entry existing for a report in the column REPORT_FORMAT of the table PAY_REPORT_FORMAT_MAPPING_F to find the formulas associated with the appropriate magnetic tape format in the table PAY_MAGNETIC_BLOCKS.

See also: Populate Rows in the PAY_REPORT_FORMAT_MAPPINGS_F Table, page 2-155.

Write Package Procedures For Assignments And Assignment Actions

You must code two package procedures as follows:

- The RANGE_CODE procedure, to specify ranges of assignments to be processed in the archive.
- The ASSIGNMENT_ACTION_CODE procedure, to create the assignment actions to be processed.

RANGE_CODE Example

This package procedure returns a select statement. This select statement returns the person_id that has the assignment for which PAR must create an assignment action.

```
--
procedure range_cursor (pactid in number,
sqlstr out varchar2) is
begin
--
sqlstr := 'select distinct person_id
           from per_people_f ppf,
           pay_payroll_actions ppa
           where ppa.payroll_action_id = :payroll_action_id
           and ppa.business_group_id = ppf.business_group_id
           order by ppf.person_id';
--
end range_cursor;
```

> **Note:** There must be one and only one entry of :payroll_action_id in the string, and the statement must be, order by person_id.

ASSIGNMENT_ACTION_CODE Example

This package procedure further restricts and creates the assignment action.

```
--
procedure action_creation(pactid in number,
                          stperson in number,
                          endperson in number,
                          chunk in number) is
--
CURSOR c_state IS
   SELECT ASG.assignment_id assignment_id
     FROM per_assignments_f ASG,
          pay_payroll_actions PPA
    WHERE PPA.payroll_action_id = pactid
      AND ASG.business_group_id = PPA.business_group_id
      AND ASG.person_id between stperson and endperson
      AND PPA.effective_date between ASG.effective_start_date
                                 and ASG.effective_end_date
 ORDER BY ASG.assignment_id;
--
lockingactid number;
begin
  for asgrec in c_state loop
     --
     -- Create the assignment action to represent the person / tax
 unit
     -- combination.
     --
     select pay_assignment_actions_s.nextval
       into lockingactid
       from dual;
      --
      -- insert into pay_assignment_actions.
      hr_nonrun_asact.insact(lockingactid,asgrec.assignment_id, pa
ctid,chunk, NULL);
   end loop;
end action_creation;
--
```

Note: Four values are passed into the procedure. Start and End person MUST be used to restrict the creation here, as these are used for multithreading. Similarly, chunk must also be used and passed to the insact procedure. This actually creates the action.

Provide an SRS Definition for the PAR Process

The PAR process is a batch process that users start from the Submit Requests window. You need to set up the SRS definition for your process. The parameters for this definition are as follows:

Table of Parameters for the PAR Process

Parameter Name	Mandatory?
report_type	Yes
report_qualifier	Yes
start_date	No *
effective_date	No *
report_category	Yes
business_group_id	Yes
magnetic_file_name	No
report_file_name	No
legislative_parameters	No *

* The PAR process requires the start_date and effective_date. However, these can be set either by entries to the standard parameters or by using special legislative parameters START_DATE and END_DATE. These special parameters are passed to the parameter legislative_parameters in the form START_DATE=<date> and END_DATE=<date>.

Populate Rows in the PAY_REPORT_FORMAT_MAPPINGS_F Table

You control PAR processing by entries you make in the table PAY_REPORT_FORMAT_MAPPINGS_F. The columns for this table are as shown in the following table:

Column Name	Type	Comments
REPORT_TYPE	NOT NULL VARCHAR2(30)	A short name of the report. Example: SQWL (for State Quarterly Wage Listing)
REPORT_QUALIFIER	NOT NULL VARCHAR2(30)	A qualifying name for the report. Example: for SQWL it could be the state name (such as Texas or California).
REPORT_FORMAT	NOT NULL VARCHAR2(30)	A foreign key to the PAY_MAGNETIC_BLOCKS table. Needed when running in ALL modes.
EFFECTIVE_START_DATE	NOT NULL DATE	
EFFECTIVE_END_DATE	NOT NULL DATE	

Column Name	Type	Comments
RANGE_CODE	VARCHAR2(60)	The name of a package procedure that you code to specify ranges of assignments to be processed in the archive. For example code, see: Write Package Procedure for Assignments and Assignment Actions, page 2-153.
ASSIGNMENT_ACTION_CODE	VARCHAR2(60)	The name of a package procedure that you code to create the assignment actions to be processed. For example code, see: Write Package Procedure for Assignments and Assignment Actions, page 2-153.
INITIALIZATION_CODE	VARCHAR2(60)	A package procedure that sets any global contexts needed for the lifetime of the archiving. Will likely be used infrequently, but you must create the procedure (see: Contexts for Database Items, page 2-152 and Examples: INITIALIZATION_CODE and ARCHIVE_CODE, page 2-157. If no value is entered in this column, PAR performs no archiving.
ARCHIVE_CODE	VARCHAR2(60)	Sets contexts at the assignment action level to be used during the archive. Will likely be used instead of INITIALIZATION_CODE. See: Contexts for Database Items, page 2-152 and Examples: INITIALIZATION_CODE and ARCHIVE_CODE, page 2-157.
MAGNETIC_CODE	VARCHAR2(60)	The standard generic magnetic tape driving PL/SQL procedure (see: Magnetic Tape Process, page 2-112). To produce the magnetic tape, PAR uses REPORT_FORMAT as a foreign key to the table PAY_MAGNETIC_BLOCKS. If no value is entered for MAGNETIC_CODE, PAR does not produce a magnetic tape.

Column Name	Type	Comments
REPORT_CATEGORY	NOT NULL VARCHAR2(30)	Indicator of the media type. Naming standards are: RT - Reel to Reel Tape SD - Floppy Disk REPORT - Paper Report ARCHIVE - Archive
REPORT_NAME	VARCHAR2(60)	This remains null for runs in the Magnetic Tape with Archive, Archive, and Magnetic Tape without Archive modes. Available for future use with other possible modes.
SORT_CODE	VARCHAR2(60)	Entered only when processing a report for which the delivery vehicle is Oracle Report Writer. Enter the name of a package procedure, which you have coded, that returns the assignment actions in the order they should be processed in.

The key to this table is REPORT_TYPE, REPORT_QUALIFIER, REPORT_CATEGORY, EFFECTIVE_START_DATE and EFFECTIVE_END_DATE.

Examples: INITIALIZATION_CODE and ARCHIVE_CODE

INITIALIZATION_CODE

```
/* Name  : archinit
 Purpose  : This performs the US specific initialization
      section.
 */
 procedure archinit(p_payroll_action_id in number) is
  jurisdiction_code   pay_state_rules.jurisdiction_code%TYPE;
  l_state             VARCHAR2(30);
begin
null;
end archinit;
```

ARCHIVE_CODE

Note: This code sets the contexts by assignment action. There are two ways of setting contexts, one using the set_context function, the other using the PL/SQL context table. The context table is used only when contexts can have multiple values, as in this example for SOURCE_ID and SOURCE_TEXT.

```
/* Name  : archive_data
   Purpose  : This performs the ZA specific employee
       context setting.
```

```
   */
procedure archive_data(p_assactid in number, p_effective_date in d
ate) is
    asgid          pay_assignment_actions.assignment_id%type;
    l_count        number;
    l_context_no   number;
    aaseq          number;
    aaid           number;
    paid           number;
  cursor cursars is
    select distinct code
      from pay_za_irp5_bal_codes
     where code in (4001, 4002, 4003, 4004, 4005, 4006, 4007);
  cursor curclr is
    select distinct nvl(pet.element_information1, '&&&')
                                        element_information1
      from pay_element_types_f pet,
           pay_element_classifications pec,
           pay_assignment_actions paa,
           pay_payroll_actions    ppa
     where paa.assignment_action_id = p_assactid
       and pec.classification_name = 'Deductions'
       and pec.classification_id = pet.classification_id
       and ppa.payroll_action_id = paa.payroll_action_id
       and exists (select ''
                   from pay_assignment_actions paa2,
                        pay_payroll_actions    ppa2,
                        pay_run_results        prr
                  where paa2.assignment_id = paa.assignment_id
                    and paa2.payroll_action_id =
                                        ppa2.payroll_action_id
                    and paa2.assignment_action_id =
                                        prr.assignment_action_id

                    and prr.element_type_id = pet.element_type_id
                    and ppa2.effective_date between ppa.start_date
                                        and ppa.effective_date
                );
  begin
      SELECT aa.assignment_id
             into asgid
        FROM pay_assignment_actions aa
       WHERE aa.assignment_action_id = p_assactid;

      l_context_no :=  pay_archive.g_context_values.sz;

      for i in 1..l_context_no loop
          pay_archive.g_context_values.name(i) := NULL;
          pay_archive.g_context_values.value(i) := NULL;
      end loop;
      pay_archive.g_context_values.sz := 0;
      l_count := 0;

/* Set up the assignment id, date earned and tax unit id contexts
*/

      l_count := l_count + 1;
      pay_archive.g_context_values.name(l_count) :=
```

```
                                        'ASSIGNMENT_ID';
      pay_archive.g_context_values.value(l_count) := asgid;

      SELECT MAX(paa.action_sequence)
        INTO aaseq
        FROM pay_assignment_actions paa,
             pay_payroll_actions ppa,
             pay_action_classifications pac,
             pay_payroll_actions        ppa_arch,
             pay_assignment_actions     paa_arch
        WHERE
          paa_arch.assignment_action_id = p_assactid
          and paa_arch.payroll_action_id =
                                      ppa_arch.payroll_action_id

          and paa.assignment_id = paa_arch.assignment_id
          AND paa.payroll_action_id = ppa.payroll_action_id
          AND ppa.action_type = pac.action_type
          AND pac.classification_name = 'SEQUENCED'
          AND ppa.effective_date between ppa_arch.start_date
                                     and ppa_arch.effective_date
          and exists (select ''
                        from pay_payroll_actions ppa2,
                             pay_assignment_actions paa2,
                             pay_run_results prr,
                             pay_element_types_f pet
                       where ppa2.time_period_id =
                                      ppa.time_period_id
                         and ppa2.payroll_action_id =
                                      paa2.payroll_action_id
                         and paa2.assignment_action_id =
                                      prr.assignment_action_id

                         and prr.element_type_id =
                                      pet.element_type_id
                         and ppa2.effective_date between
                                       pet.effective_start_date and
                                      pet.effective_end_date
                         and paa2.assignment_id = paa.assignment_i
d
                         and pet.element_name =
                                      'ZA_Tax_On_Lump_Sums')
          and not exists (select ''
                            from pay_assignment_actions paa3,
                                 ff_archive_items fai,
                                 ff_user_entities fue
                           where paa3.assignment_id =
                                      paa_arch.assignment_id
                             and paa_arch.payroll_action_id =
                                      paa3.payroll_action_id
                             and paa3.assignment_action_id =
                                      fai.context1
                             and fai.user_entity_id =
                                      fue.user_entity_id
                             and fue.user_entity_name =
                                      'A_PAY_PROC_PERIOD_ID'
                             and fai.value = ppa.time_period_id);
      if aaseq is null then
```

```
        SELECT MAX(paa.action_sequence)
          INTO aaseq
          FROM pay_assignment_actions paa,
               pay_payroll_actions ppa,
               pay_action_classifications pac
          WHERE
            paa.assignment_id = asgid
            AND paa.payroll_action_id = ppa.payroll_action_id
            AND ppa.action_type = pac.action_type
            AND pac.classification_name = 'SEQUENCED'
            AND ppa.effective_date <= p_effective_date;
      end if;
      SELECT assignment_action_id, payroll_action_id
        INTO aaid, paid
        FROM pay_assignment_actions
       WHERE
         assignment_id = asgid
         AND action_sequence = aaseq;

      l_count := l_count + 1;
      pay_archive.g_context_values.name(l_count) :=
                                        'ASSIGNMENT_ACTION_ID';
      pay_archive.g_context_values.value(l_count) :=aaid ;
      pay_archive.balance_aa := aaid;

      l_count := l_count + 1;
      pay_archive.g_context_values.name(l_count) :=
                                        'PAYROLL_ACTION_ID';
      pay_archive.g_context_values.value(l_count) :=paid ;
      for clrrev in curclr loop
        l_count := l_count + 1;
        pay_archive.g_context_values.name(l_count) :=
                                         'SOURCE_TEXT';
        pay_archive.g_context_values.value(l_count) :=
                                  clrrev.element_information1;
      end loop;
      for sarrec in cursars loop
        l_count := l_count + 1;
        pay_archive.g_context_values.name(l_count) := 'SOURCE_ID';

        pay_archive.g_context_values.value(l_count) := sarrec.code
;
      end loop;
-
      pay_archive.g_context_values.sz := l_count;
-
  end archive_data;
```

Balances in Oracle Payroll

This essay deals with the definition and use of balances and balance dimensions in Oracle Payroll. It also explains how to deal with the issue of loading initial balances. This essay does not provide any detail on how to add balance dimensions to the system.

Terms

This essay assumes that you are already familiar with the database design diagrams and tables contained in the Oracle HRMS *Technical Reference Manual*.

If you are not already familiar with the setup and use of balances, or the concepts of employee assignment, assignment actions, database items, or payroll processing in Oracle FastFormula you should refer to your Oracle HRMS user guides for more information.

For additional information on how the Payroll Run processes balances, see also: Payroll Run Process - Create and Maintain Balances, page 2-100.

Overview of Balances

In Oracle Payroll a balance is defined as the accumulation of the results of a payroll calculation. The balance has a name, feeds and dimensions.

For example, the balance GROSS PAY is the accumulation of the results of processing all 'Earnings'. However, the idea of a dimension is unique to Oracle Payroll. Dimensions enable you to view the value of a balance using a combination of different criteria. So, you might want to view the value of Gross Pay for one employee for the current pay period, or for the year to date. The actual balance and dimension you would use in a formula or a report would be the GROSS_PAY_ASG_PTD or the GROSS_PAY_ASG_YTD.

In general, balances in Oracle Payroll can be thought of as the 'calculation rules' for obtaining the balance value. Most values are not held explicitly in the database. This approach has many advantages: New balances can be defined and used at any time with any feeds and dimensions; balance values do not need to be stored explicitly in the database, taking up valuable storage space and causing problems with data archiving and purging.

Balance Types

These are the balance names, for example Gross Pay and Net Pay. Balance types always have a numeric Unit Of Measure, and in some instances a currency code.

Balance Feeds

Balance feeds define the input values that contribute to a balance. For example the pay values of all earnings types contribute to the Gross Pay balance. Feeds can add to (+) or subtract from (-) a balance

Balance Dimensions

The balance dimension is identified by the database item suffix for the balance. For example, '_YTD' indicates the balance value is for the year to date. Balance dimensions are predefined in Oracle Payroll.

Defined Balances

The defined balance is the name used to identify the combination of Balance Type and Balance Dimension. For example, GROSS_PAY_ASG_YTD. When you use the Balance window to define a new balance, Oracle Payroll automatically generates database items for every balance dimension you select. You can then access the value directly within any formula. In any detailed calculation or report on balances you always refer to the 'defined balance' to return a value.

Latest Balances

To optimize the performance of payroll processing, some balance values are held explicitly in the database and these are referred to as **Latest Balance Values**. The payroll process accesses and updates latest balance values as it runs. In some cases it clears and then resets values, for example when you do a rollback. All of this is invisible to the user and is managed by the payroll process.

Note: If you need to return the value of a balance in a report you should use the balance function pay_balance_pkg.get_value. See: Including Balance Values in Reports, page 2-177.

Expiry

An important concept for latest balances is that of 'expiry'. For example, consider the GROSS_PAY_YTD balance. When you cross the tax year boundary you would expect the value to return to zero. This 'expiry' of a balance is maintained internally by Oracle Payroll and there is code to work out if we have crossed such a boundary.

Important: Even if a defined balance has expired in theory for a payroll run, it is not actually zeroed on the database unless it is subsequently updated by the same payroll run. Thus, following a Payroll Run, you may well see balances that you would have expected to have expired, but have their old values.

Balance Contexts

There is occasionally a requirement to report balances where the combination of ASSIGNMENT_ACTION_ID and BALANCE_TYPE_ID does not uniquely identify the individual balance values that should be reported. For example in the US legislation you need to maintain balance dimensions for particular states, while in the UK legislation you need to maintain balance dimensions for distinct tax offices.

Both of these requirements are met by the definition of special balance contexts. These are legislative specific 'C' code and appear to you as part of the balance dimensions.

User definition of additional balance contexts is not yet supported because of the major impact these may have on the overall performance of the payroll process. Bad code in the definition of these contexts can run exceptionally slowly, especially when you accumulate a large number of run results.

Context Balances - a UK Example

To report on context balances, we must define the relevant balances with the ELEMENT_PTD and ELEMENT_ITD dimensions. The further context that is required to identify the values is taken from the PAY_RUN_RESULTS.SOURCE_ID. This is obtained from the balance feed joining to the PAY_RUN_RESULT_VALUES table, then to PAY_RUN_RESULTS.

Using this value, we can select via the PAY_ASSIGNMENT_LATEST_BALANCES -> PAY_BALANCE_CONTEXT_VALUES method. Or, if there is no latest balance, by the route code call, which in the UK can be done with a function call:

```
hr_gbbal.calc_element_ptd_bal(ASSIGNMENT_ACTION_ID,
                              BALANCE_TYPE_ID,
                              SOURCE_ID);
(or calc_element_itd_bal with the same parameters).
```

Balance Dimensions

This essay describes what a balance dimension is and what it does, and how the various parts interact with formulas and the Payroll Run.

A balance dimension defines how the value of a specific balance should be calculated. The balance dimension is also an entity with its own attributes that are associated with balance calculations.

Database Item Suffix

The database item suffix identifies the specific dimension for any named balance. The 'defined balance' name is the combination of the balance and the suffix. For example, the suffix '_ASG_YTD' in 'GROSS_SALARY_ASG_YTD' identifies that the value for the gross salary balance is calculated for one assignment, for the year to date.

Routes

The balance dimension route is a foreign key to the FF_ROUTES table. A route is a fragment of SQL code that defines the value to be returned when you access a balance. As with other database items, the text is held in the DEFINITION_TEXT column of the FF_DATABASE_ITEMS table.

The select clause of the statement is always:

```
select nvl(sum(fnd_number.canonical_to_number(TARGET.result_value
) * FEED.scale), 0)
```

Thus, a balance could be defined as the sum of those run result values that feed the balance type ('Gross Salary' in our example), across a certain span of time (in our example, this is since the start of the current tax year).

The SQL statement itself must follow a number of rules, and an example appears below:

```
        pay_balance_feeds_f      FEED
       ,pay_run_result_values    TARGET
       ,pay_run_results          RR
       ,pay_payroll_actions      PACT
       ,pay_assignment_actions   ASSACT
       ,pay_payroll_actions      BACT
       ,pay_assignment_actions   BAL_ASSACT
where  BAL_ASSACT.assignment_action_id = \&B1
and    BAL_ASSACT.payroll_action_id    = BACT.payroll_action_id
and    FEED.balance_type_id            = \&U1
and    FEED.input_value_id             = TARGET.input_value_id
and    TARGET.run_result_id            = RR.run_result_id
and    RR.assignment_action_id         = ASSACT.assign_action_id
and    ASSACT.payroll_action_id        = PACT.payroll_action_id
and    PACT.effective_date between
          FEED.effective_start_date and FEED.effective_end_date
and    RR.status in ('P','PA')
and    PACT.effective_date >=
          (select to_date('06-04-' || to_char( to_number(
                  to_char( BACT.effective_date,'YYYY'))
           +  decode(sign( BACT.effective_date - to_date('06-04-'
               || to_char(BACT.effective_date,'YYYY'),'DD-MM-YYYY'
)),-1,-1,0)),'DD-MM-YYYY')
           from dual)
and    ASSACT.action_sequence <= BAL_ASSACT.action_sequence
and    ASSACT.assignment_id = BAL_ASSACT.assignment_id');
```

This example is the route for a UK based assignment level year to date balance that uses the 6th of April as the start of the tax year.

Comments

The route is made up of the following parts:

1. Return all possible actions for the assignment
2. Identify the possible feeds to the balance
3. - feed checking
4. Restrict the period for which you sum the balance

   - expiry checking

   Note: The expiry and feed checking parts have a special significance that will become obvious later.

Specific table aliases should be used as they have a particular meaning.

- The BAL_ASSACT table is the 'source' assignment action, that is, the current action for this assignment.
- The ASSACT table is the 'target' assignment action, that is, the action for those results that feed the balance.
- The PACT table is the 'target' payroll action, that is, used to define the date of the ASSACT assignment actions.
- We join to the BACT table, getting all the Payroll Actions in which the assignment appears.
- We join to the FEED table for the balance type and get all the TARGET input values that could possibly feed this balance.
- The run results that feed must be processed ('P' or 'PA').
- The complicated looking sub-query returns the start of the current tax year, which is from when we are summing the balance. That is, the results that feed the balance will be between the start of the current tax year and the current action sequence.

Dimension Type

Dimension type determines how a balance is treated by the Payroll Run, and for predefined dimensions this is optimized for performance of the payroll run.

The dimension type can take one of the following values:

- **N** - Not fed and not stored. This dimension type does not create a latest balance at any time. A balance with this dimension will always have its SQL re-executed whenever that balance is executed.
- **F** - Fed but not stored. This dimension type creates a balance 'in memory' during the Payroll Run. This balance is fed by the run code but it does not store a latest balance on the database.
- **R** - Run Level balance. This dimension type is used specifically for those balances that total for the current run and must be used with the appropriate route. No latest balance value is stored on the database.
- **A** - Fed and stored at assignment level. This dimension type creates an assignment level latest balance and stores it in the PAY_ASSIGNMENT_LATEST_BALANCES table.
- **P** - Fed and stored at person level. This dimension type creates a person level latest balance and stores it in the PAY_PERSON_LATEST_BALANCES table.

Feed Checking Type

The feed checking type controls the feed checking strategy used during the payroll run. This type is used to keep the in memory balance up to date by deciding whether a run result should feed the balance. It can have the following values:

- **Null** This is the default value, and means that all the run result values included by the existing balance feeds will feed the balance.
- **P** Payroll Run executes the package procedure defined in the expiry_checking_code column on the dimension. An expiry flag parameter indicates whether feeding should occur or not.
- **E** Equality feed checking is done. That is, feeding occurs if there is a match between the in memory balance context values and the contexts held in the UDCA (User Defined Context Area).

The following additional types are for US and Canadian legislative balances only:

- **J** Jurisdiction checking is done.
- **S** Subject Feed Checking is done.
- **T** A combination of 'E' and 'S' feed checking types.
- **M** A combination of feed checking types 'S', 'J' and 'E'.

Expiry Checking Type

Latest balances should expire (that is, return to zero) at a time determined by their dimension. For example, a YTD (Year to Date) balance expires at the end of the year.

All loaded balances are checked for expiry by the Payroll Run, according to their expiry checking type:

- N - Never expires: balances are never set to zero.
- P - Payroll Action Level: for these types, a list of the expiry check results for each owning action/balance dimension are kept.

  Once expiry checking code has been called for such a combination, it does not need to be checked again for other balances that have the same combination, thus avoiding multiple calls to the database.

  The expiry checking is balance context independent - the list of balance contexts is not passed to the expiry checking code.
- A - Assignment Action Level: no assumptions can be made, expiry checking code is always called. The expiry checking is balance context dependent - the list of the balance contexts is passed to the expiry checking code.
- D - Date Expiry: the date expiry checking mechanism looks at the balance dimension/balance contexts combination of the balance being expiry checked, and scans the in-memory list to see if a balance with the same combination has already been expiry checked.

  If so, the expiry date is taken from that stored on the in-memory balance.

  The expiry checking is balance context dependent-the list of the balance contexts is passed to the expiry checking code.

Initial Balance Loading for Oracle Payroll

This essay describes the functionality available with Oracle Payroll to assist in the loading of initial balance values from an existing payroll system.

Introduction

Whether you are implementing Oracle Payroll for the first time, or upgrading from an earlier release you will need to set initial values for your legislative balances. It is essential for the accurate calculation of legislated deductions in Oracle Payroll that the initial values for these balances are correct.

This section shows you how to set up and load these initial balance values before you begin to process payrolls. After you have begun processing payrolls you may need to repeat this process for additional user balances you define in the future.

> **Warning:** The steps you follow to load initial balances are completely different from the steps an end user follows to adjust a balance. You must not use the balance loading method to make balance adjustments.

Balances and Balance Adjustments in Oracle Payroll

In Oracle Payroll a balance is the accumulation of the results of a payroll calculation. The balance has a name, feeds and dimensions. The results that feed a specific balance are known as the 'balance feeds' and these can add or subtract from the total. The balance loading process calculates and inserts the correct run results to set the initial values with effect from the upload date.

Balances are calculated directly from the run results that are designated as feeding the balance. This approach ensures run results and balance values are always in step and it removes the need to store and maintain extra information in the database. In effect, the definition of a balance is really the definition of the 'calculation' that is performed to return the balance value.

The run results that feed a defined balance are usually the results of processing elements during a payroll run. However, there may be times when balance values have to be adjusted manually. You do this by making an entry of an element as a 'balance adjustment'. When you make a balance adjustment online, the effect is to create a single processed run result for the element. This run result automatically feeds, or adjusts, all the balances that are normally fed by the element. In this way, you are able to cascade the adjustment to all affected balances.

> **Important:** When performing an online balance adjustment you must be careful to choose the right element and input value. However, if you make a mistake you can always go back and delete and re-enter the adjustment. You delete balance adjustments from the Payroll or Assignment Actions windows.

Steps

There are three basic steps involved in loading initial balance values:

1. Define an element and input value to feed each specific balance
2. Set up the initial balance values in the tables

   PAY_BALANCE_BATCH_HEADERS

   PAY_BALANCE_BATCH_LINES

3. Run the *Initial Balance Upload* process
    - Use the SRS window.
    - Use Validate, Transfer, Undo and Purge modes as needed.

Balance Loading Process

When you run the initial balance loading process you set values for each balance relative to a specific date - the **Upload Date**. The process creates datetracked balance entries, or 'adjustments', to ensure your legislative balances are correct from the upload date. Maintenance of balance information after this date is managed by the system, or by using the balance adjustments.

Consider the following example of three dimensions for gross pay balance values for one employee.

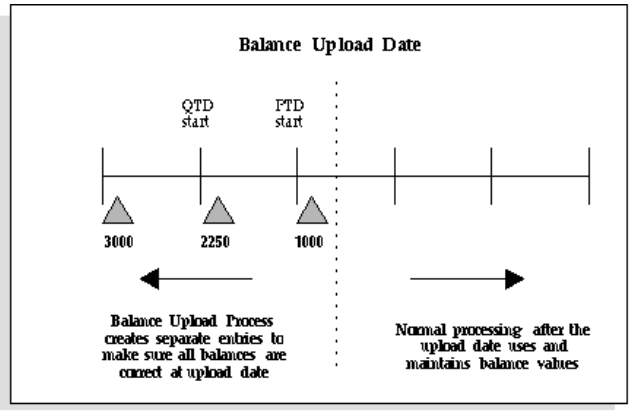
- Gross Pay Ptd 1000.00
- Gross Pay Qtd 3250.00
- Gross Pay Ytd 6250.00

The balance loading process must calculate the actual values required for each entry and the effective date for these entries. The result of the calculation is the creation of 3 balance entries.

- _PTD balance entry value is 1000.00
- _QTD balance entry value is 2250.00
- _YTD balance entry value is 3000.00

Balance Loading

Balance Loading



The result is that the cumulative values of the individual entries match the initial requirement for each balance.

- Gross Pay Ptd = 1000.00
- Gross Pay Qtd = 1000.00 + 2250.00 = 3250.00
- Gross Pay Ytd = 1000.00 + 2250.00 + 3000.00 = 6250.00

Latest Balances

To improve payroll run performance Oracle Payroll sets and maintains 'Latest Balance Values'. If these values are not set, the balance value is created by summing the run

results for the balance. If a large number of assignments have no value then there could be a significant impact on the first payroll run. Therefore, loading the latest balances prior to the first payroll run has significant implications for performance.

> **Note:** Some balances cannot have latest balances, such as those that are used in-memory but not stored.

When you are deciding which balances and dimensions you should include in the initial loading process, consider the balances that are used in the payroll run. For example, if the payroll run uses the balance bal_YTD, but the upload process loads bal_PTD only, then the latest balance value for bal_PTD exists but not for bal_YTD. The first payroll run would have to evaluate bal_YTD.

In the normal payroll run the latest balance value is associated with the last assignment action that uses the defined balance. The balance upload process attempts to simulate this action by creating a number of balance adjustment entries prior to the upload date.

> **Important:** If the defined balance includes contexts then the latest balance can only be created on a balance adjustment payroll action that has context values that do not contradict the latest balance that is to be created.

In Oracle Payroll, each balance adjustment entry is considered to be a separate assignment action. These adjustments are performed in date order - earliest first. The last balance adjustment, with the highest assignment action number, is used to create the latest balance.

Setting Up an Element to Feed Initial Balances

Because of the complex web of feeds that can exist for any specific balance there is a simple mechanism to let you set the initial value for any specific balance. The basic principle is that you require a special element input value to feed each specific balance; and you set each balance separately.

Elements to Initialize Legislative Balances

Oracle Payroll comes with the predefined elements and input values you need to set initial values for all your legislative balances.

> **Important:** US and Canadian users should run a special PL/SQL script (paybalup.pkb) to create the elements and inputs needed to feed the predefined legislative balances. This script has been registered as an SRS process - *Initial Balance Structure Creation*. You will need to create batch lines for each of these elements.

Users in other legislations need only link the predefined elements that feed the legislative balances that must be initialized.

Elements to Initialize User-defined Balances

For all other balances you need to set up the elements that will provide the entry values for each of your initial balances. There are some rules for setting up elements for initial balance feeds.

Element

- Must have a start date 01-JAN-0001

This rule simplifies the validation by making sure that the element and input value to feed the balance are always available.

- Must have a classification of 'Initial Balance Feed'

  This classification is excluded from the list of classifications available when you define a balance. You can only set up manual balance feeds for this type of element.

- Must be 'Adjustment Only'
- Must be a nonrecurring type
- Must be processable in a payroll run

Input Values

- Must have a start date 01-JAN-0001
- Each input value must feed only one balance

  If you need to set initial values for a large number of balances you can define multiple input values for a single element with each input value feeding a different balance.

Element Link

- Must have a start date 01-JAN-0001
- Criteria must be only Link To All Payrolls - 'Yes'

Supported Balances

All the balances supported by the initialization process are set at the assignment level. Balances at the person level are set indirectly by accumulating the values from all the assignments.

Setting Up the Initial Balance Values

There can be many different sources for the initial balance value to be loaded. For example, you may be migrating from a previous version of Oracle Payroll, or from another payroll system, or you may hold this information in another system.

Two batch interface tables are supplied with Oracle HRMS to standardize the process of loading the initial balance values. You can load information directly into these tables and you can also review, update and insert values manually. This gives you total flexibility for setting values. It also enables you to define and manage the loading of separate batches as logical groups.

PAY_BALANCE_BATCH_HEADERS

Name	Null?	Type
BUSINESS_GROUP_ID		NUMBER(15)
PAYROLL_ID		NUMBER(9)
BATCH_ID	NOT NULL	NUMBER(9)
BATCH_NAME	NOT NULL	VARCHAR2(30)
BATCH_STATUS	NOT NULL	VARCHAR2(30)
UPLOAD_DATE	NOT NULL	DATE
BATCH_REFERENCE		VARCHAR2(30)
BATCH_SOURCE		VARCHAR2(30)
BUSINESS_GROUP_NAME		VARCHAR2(60)
PAYROLL_NAME		VARCHAR2(80)

Each batch identifies the payroll that is being uploaded and the date of the upload. Other identifiers can be set to identify uniquely each batch as shown, for example, in the following table.

Batch Name	Batch Ref	Batch Source	Payroll	Upload Date
Weekly Payroll	0001	SQL*Loader	Pay1	01-Jan-1995
Weekly Payroll	0002	SQL*Loader	Pay1	01-Jan-1995
Monthly Payroll	0003	SQL*Loader	Pay2	01-Jan-1995
Semi Monthly Payroll	0001	Screen	Pay3	01-Aug-1995

PAY_BALANCE_BATCH_LINES

Name	Null?	Type
ASSIGNMENT_ID		NUMBER(10)
BALANCE_DIMENSION_ID		NUMBER(9)
BALANCE_TYPE_ID		NUMBER(9)
PAYROLL_ACTION_ID		NUMBER(9)
BATCH_ID	NOT NULL	NUMBER(9)
BATCH_LINE_ID	NOT NULL	NUMBER(9)
BATCH_LINE_STATUS	NOT NULL	VARCHAR2(30)
VALUE	NOT NULL	NUMBER
ASSIGNMENT_NUMBER		VARCHAR2(30)
BALANCE_NAME		VARCHAR2(80)
DIMENSION_NAME		VARCHAR2(80)
GRE_NAME		VARCHAR2(60)
JURISDICTION_CODE		VARCHAR2(30)
ORIGINAL_ENTRY_ID		NUMBER(15)

Each batch has a set of batch lines that include details of the assignment, the balance and the value for each dimension. You can also include other contexts for a specific balance.

Assignment	Balance	Dimension	Value
101	Gross Pay	PTD	1000.00
101	Gross Pay	QTD	3250.00
101	Gross Pay	YTD	6250.00
101-2	Gross Pay	PTD	750.00

Note: The tables provide support for either a system ID (such as assignment_id) or a user ID (such as assignment_number) for each piece of information. This allows maximum flexibility when you are populating the batch tables.

The rule is that if both are specified then the system ID overrides the user ID. Here is a list of the system IDs and user IDs that can be specified when setting up the tables:

System ID	User ID
BUSINESS_GROUP_ID	BUSINESS_GROUP_NAME
PAYROLL_ID	PAYROLL_NAME
ASSIGNMENT_ID	ASSIGNMENT_NUMBER
BALANCE_DIMENSION_ID	DIMENSION_NAME
BALANCE_TYPE_ID	BALANCE_NAME
ORIGINAL_ENTRY_ID	
GRE_NAME (US and Canada only)	
JURISDICTION_CODE (US and Canada only)	

If an error occurs during the processing of the batch, the error message is written to the PAY_MESSAGE_LINES table with a source_type of H (header) or L (line).

Running the Initial Balance Upload Process

You run the *Initial Balance Upload* process from the SRS window to upload values from the batch tables. You can run this process in one of four modes:

- Validate
- Transfer
- Undo Transfer
- Purge

Prerequisites

On the upload date, every assignment in the batch must belong to the payroll identified in the batch header.

The payroll must have a sufficient number of time periods prior to the upload date to allow the setting of the initial balances.

Other specific criteria, such as the GRE or Legal Company, are not validated by the initial balance loading process. It is your responsibility to validate this information.

> **Note:** The validation process contains a predefined hook to enable you to apply your own additional validation procedure to your own balances. The procedure should be named *validate_batch_line*.
>
> The process will check for valid data but will not set it.

Modes

Validate Mode

There is no validation of the batch tables prior to running this process. The process validates data in PAY_BALANCE_BATCH_LINES, but does not transfer these to the

Oracle HRMS database. It marks valid lines with V (Validated), and lines in error with E (Error), and sends error messages to the PAY_MESSAGE_LINES table.

The validation process is split into two phases:

- The first phase checks the integrity of the data in the batch tables.
- The second phase checks that it is possible to create all the required balance adjustment entries.

The validate process also populates the system ID entries in the table. This ensures that all subsequent processing has access to the system IDs.

All batch lines are validated independently and are marked with their individual status at the end of the process.

Transfer Mode

Transfer mode repeats the first phase of the validation check to ensure the integrity of the data in the batch tables and the existence of all system IDs.

The process calculates the balance adjustment entries required for each assignment. This list is checked and aggregated where values are shared and actual entries are then created for the assignment. This is repeated for each assignment in the batch. Successful transfer is marked with a status of T - Transferred.

> **Note:** If any line for an assignment is in error, none of the lines for the assignment are transferred into the HRMS database. Failures are logged in the messages table against the batch line being processed and the batch line is marked as I - Invalid.

If the value of the adjustment is zero then no entry is created. For example:

Balance_PTD = 500

Balance_QTD = 500

There is no need for an adjustment to the QTD dimension since the value is already set by the PTD.

It is likely that there will be large volumes of data to load, so the work is periodically committed to preserve successful work and to reduce the number of rollback segments required.

> **Note:** The commit size is specified by the CHUNK_SIZE parameter in PAY_ACTION_PARAMETERS. The default for CHUNK_SIZE is 20 successful assignments.
>
> This is the same parameter used by other payroll processes to determine commit frequency.

If a batch has been processed with partial success, you can resubmit the batch and only those assignments with batch lines that have not been Transferred are processed again. You can also restart the batch process if it failed during processing, for example if it ran out of tablespace.

Undo Transfer

This mode removes all the balance adjustment entries created by the transfer process and return the status of the batch lines to U.

Note: The data in the batch tables is kept. You can correct any batch lines with incorrect values and repeat the transfer.

Purge

Purges all data in a batch regardless of current status. When a batch is purged all the messages, batch lines and the batch header are removed. This enables you to reclaim space once a batch is successfully transferred.

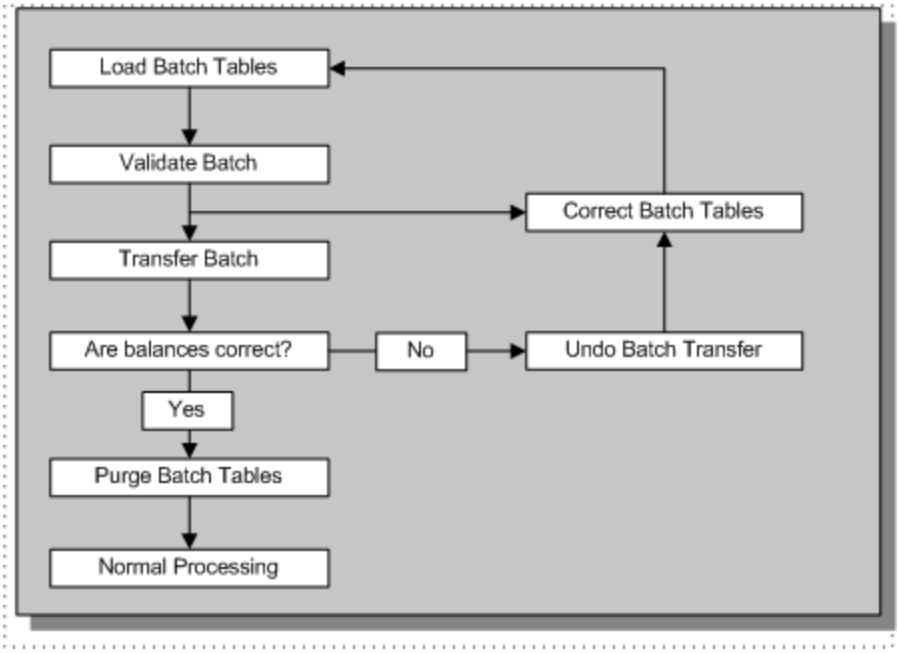
Use Purge mode only when you are sure that the balances for all assignments in a batch have been successfully entered into the HRMS database.

Warning: Once you have purged a batch, all the entries for that batch are deleted. This action cannot be undone.

Process Flow

The normal sequence for using these modes to load initial balances is shown in the following diagram:

Process Flow



Error Statuses

Any errors encountered are recorded in the messages table against the object being validated: either the batch itself or an individual batch line. The status set against the batch or batch lines is dependent on the mode the process is running in as well as the status of other batch lines.

Batch Line Status

The status of each batch line can be one of the following :

- V - Valid; the batch line is OK
- E - Invalid; the batch line has an error

- T - Transferred; the batch line has been successfully transferred

Batch Status

The status of the batch is dependent on the statuses of the batch lines within the batch:

- T - Transferred; all lines in the batch have been transferred
- P - Partially Transferred; some lines in the batch have been transferred
- V - Valid; all the lines in the batch are valid and none have been transferred
- E - Invalid; some of the lines in the batch are invalid and none have been transferred

Validation Problems

There are two common problems you should check.

The adjustment request for a balance dimension may be incorrect. For example, suppose an assignment has the following upload requests:

- <Balance>_QTD = 1500.00
- <Balance>_YTD = 1000.00

The YTD value is lower than the QTD value. This may be valid, if the balance decreases over time. However, balances normally increase so it is advisable to check a balance that has been decreased.

Secondly, an invalid adjustment error may occur, where the process could not find the correct date to do the adjustment. The cause of this error depend on the balance dimension that is being processed.

However, it is always good practice to make sure that all the business group details are correct, and there are enough payroll periods for the balance to be set. To check which date is being used for each assignment balance, use the following SQL:

```
select BL.dimension_name,
pay_balance_upload.dim_expiry_date
(BH.business_group_id
,BH.upload_date
,BL.dimension_name
,BL.assignment_id
,BL.gre_name
,BL.jurisdiction_code
,BL.original_entry_id)    expiry_date
from pay_balance_batch_headers BH
,pay_balance_batch_lines   BL
where BH.batch_name    = '&Batch_Name'
and BL.batch_id      = BH.batch_id
and BL.assignment_number = '&Assignment_Number'
and BL.balance_name  = '&Balance_Name'
;
```

If the expiry date is set to '31-DEC-4712' then the adjustment date could not be found.

Balance Initialization Steps

Here's a simple check list on how to set up the data:

1. Create payrolls in Oracle Payroll with periods going back to the start of the year. Enter all employees into Oracle HRMS and give them assignments to these payrolls.

Important: The next step applies to US and Canadian users only. Users in other legislations need only define links for the predefined balance loading elements.

2. From the Submit Requests window, run the *Initial Balance Structure Creation* process, selecting a batch name as the parameter. For each batch, this process creates:
   - An input value to hold the amount of each balance and of any context, and enough elements with the special classification Balance Initialization to hold all the input values created
   - The necessary links and balance feeds for these elements
3. Create any other elements you need to initialize balances for your own earnings and deductions.
   - Follow the requirements listed above. See: Setting Up an Element to Feed Initial Balances, page 2-168.
   - Use multiple input values to reduce the number of elements
   - Define one balance feed for each input value

     Note: Each balance must have one initial balance feed only.

     Multiple input values for one element must feed balances that have the same 'upload date'.

4. Group employees into batches for managing initialization of their balances. Enter an identifying header for each batch (these headers go into the PAY_BALANCE_BATCH_HEADERS table). Each header contains the following information:
   - Business Group name and payroll name
   - Batch name and ID number
   - Upload date: the date on which the balances in the current system will be correct and ready for transfer

   For example:

Batch Name	Business Group	Payroll Name	Upload Date
Upload 1	BG name	Full Time 1	13-AUG-1995

5. Create a batch line for each balance to be transferred (these lines go into the PAY_BALANCE_BATCH_LINES table). A batch line includes the following information:
   - Employee assignment number
   - Balance name and dimension, such as quarter to date or year to date
   - Balance value
   - Balance context where appropriate. For US and Canadian users the context may include a GRE and a jurisdiction (federal, state, local, or provincial).

     Note: The process uses your balance feed definitions to determine which element input value to use.

- For example:

Asg. Number	Balance	Dimension	Value
60001	Salary	PTD	700
60001	Salary	QTD	1400
60001	Salary	YTD	2400
60001	Tax Paid	PTD	2200
60001	Tax Paid	QTD	2400
60001	Tax Paid	YTD	2400

Important: The Tax Paid YTD value is not required because it has the same value as the QTD. However, this balance is included to create a value for the latest balance, and improve the performance of the first payroll run.

6. From the Submit Requests window, run the Initial Balance Upload process. Select the mode in which to run this process as a parameter. Available modes are:

   - **Validate**

     Validate batch lines but do not transfer

     Send error messages to PAY_MESSAGE_LINES

   - **Transfer**

     Validate and transfer batch lines

     If any line for an assignment is in error, none of the lines for the assignment are transferred

   - **Undo**

     Removes balance initialization entries from the database and marks the lines as U in the batch lines table.

   - **Purge**

     Purges all lines in the batch lines table, regardless of how they are marked.

     Note: Use Purge mode only when you are sure that the balances for all assignments in a batch have been successfully entered into the HRMS database.

Including Balance Values in Reports

This section describes the PL/SQL interface for the balance function that enables you to access balance values for inquiry and reporting tools.

UK users - see: Including Balance Values in Reports (UK Only), *Oracle HRMS Implementation Guide (UK)*

Tip: If you need to report the same balance value many times in different reports you might consider creating a reporting table. You would simply include the balance function in your PL/SQL script to populate this table.

Advantages

Using this PL/SQL function to retrieve balance values has several advantages:

- You can easily call the function from a form or SRW2 report.
- You can access latest balance values, where they exist. This will optimize performance automatically.

The Balance Function

The interface to the balance function is flexible and easy to use. Hard coded knowledge of contexts within the function are kept to a minimum and the balance function is controlled as follows:

- Before the function is called, calls are made to another PL/SQL function to set up the contexts to be used. These are held in package level PL/SQL tables. This enables the balance function to operate without hard coded knowledge of the contexts, and reduces client-server calls for several balances.
- The 'C' balance user exit works in two modes: date and assignment action. The balance function does not pass a mode parameter; instead the mode is resolved by using the PL/SQL overloading feature. This simplifies the interface.

The PL/SQL code resides in one package.

```
pay_balance_pkg
```

Procedure : Initialize the contexts:

```
procedure set_context (p_context_name  in varchar2, p_context_val
ue in varchar2);
```

For example:

```
pay_balance_pkg.set_context ('TAX_UNIT_ID', p_tax_unit_id);
```

This is called to set up ALL contexts required for a balance, with the exclusion of assignment action id. Context values are maintained throughout the entire session. Subsequent calls with the same context name update the value.

> **Note:** The context name can be specified in any case. The routine converts all context names to upper case.

Function : Get balance value (Assignment action mode):

```
function get_value (p_defined_balance_id    in number,
p_assignment_action_id  in number,
p_always_get_db_item   in boolean default false)
return number;
```

Function : Get balance value (Date mode):

```
function get_value  (p_defined_balance_id    in number,
p_assignment_id         in number,
p_virtual_date          in date,
p_always_get_db_item   in boolean default false)
return number;
```

The balance value is returned by this function. The parameters required for the function have been kept to a minimum. Legislation code and business group id are derived by the PL/SQL function when the balance SQL has to be built up from ff_routes.

> **Note:** If the balance uses business_group_id as a context then this must be set up using the set_context routine.

The parameter 'p_always_get_db_item' can be ignored. It is used for testing purposes. If this value is set to 'true' then the function will not even look for a latest balance value, and will always derive the balance from the database item.

Example

This example shows how to access parameterized balances supporting jurisdiction- and GRE-based taxation (US and Canada specific).

In the UK, with the exception of court orders, no use is made of parameterized balances.

Note: For balances that are not parameterized, no calls to pay_balance_pkg.set_context are necessary.

1. **Set up the contexts**

```
pay_balance_pkg.set_context ('TAX_UNIT_ID',  1);
pay_balance_pkg.set_context ('JURISDICTION_CODE', '01-123-4567');
```

2. **Retrieve the balance value**

```
bal_value := pay_balance_pkg.get_value (p_def_balance_id, p_asg_a
ction_id);
```

3. **Retrieve the balance for a different jurisdiction code but using the same value for tax unit id**

```
pay_balance_pkg.set_context ('JURISDICTION_CODE', '99-999-1234');
bal_value := pay_balance_pkg.get_value (p_def_balance_id, p_asg_ac
tion_id);
```

FastFormula

The FastFormula Application Dictionary

The FastFormula Application Dictionary is designed to hide the complexity of the application database from the FastFormula user. When you write a formula, you reference database items. The Dictionary contains the information that FastFormula requires to generate the SQL and PL/SQL error checking code that extracts these database items.

For example, in a formula you might refer to the database item EMPLOYEE_LAST_NAME. When the formula is run, FastFormula uses information in the Dictionary to build up a complete SELECT statement to extract the name from the database.

Normally, you do not need to be aware of the contents of the Dictionary. For example, when you define a new element, several database items are generated automatically. The information that enables FastFormula to extract these new items is generated at the same time.

However, if you do need to define new database items directly in the Dictionary, you must also load the associated information. The next section describes the entities that you must create in the Dictionary. The following section gives step-by-step instructions for defining new database items.

Suppose FastFormula is running a formula that references the database item EMPLOYEE_LAST_NAME from the table PER_PEOPLE. The SQL required to extract EMPLOYEE_LAST_NAME is as follows:

```
SELECT TARGET.last_name
 FROM per_people             TARGET
 ,    per_assignments        ASSIGN
 WHERE TARGET.person_id    = ASSIGN.person_id
 AND ASSIGN.assignment_id  = &B1
```

This section explains where this information is stored in the Dictionary and how FastFormula builds it up to form the SQL statement.

Note that the Dictionary stores information at the physical level. That is, it stores parts of the text of SQL statements, which are used by FastFormula to build up the complete statements. It does not store information about entities and relationships.

Database Items and User Entities

EMPLOYEE_LAST_NAME is a value in the USER_NAME column of table FF_DATABASE_ITEMS in the Dictionary. When FastFormula runs a formula in which EMPLOYEE_LAST_NAME is a variable, it accesses this table for two reasons:

- It gets the value in the DEFINITION_TEXT column. This is the value that appears in the SELECT clause of the SQL. In our example, it is PER_PEOPLE.LAST_NAME. (TARGET is an alias for PER_PEOPLE.)
- It identifies the user entity of which the database item is a part. A user entity is a group of one or more database items that can be accessed by the same route. In our example, the user entity might be EMPLOYEE_DETAILS.

Routes and Route Parameters

Using the user entity ID, FastFormula checks the table FF_USER_ENTITIES to identify the route associated with the user entity. The route is the text of the SQL statement following the FROM keyword. It is held in the table FF_ROUTES. In our example, the route is:

```
per_people                 TARGET,
 per_assignments            ASSIGN
 WHERE TARGET.person_id   = ASSIGN.person_id
 AND ASSIGN.assignment_id = &B1
```

If several user entities use the same route, the route contains one or more placeholders of the form &U# (where # is a sequence number). Each placeholder references a parameter in table FF_ROUTE_PARAMETERS. FastFormula identifies the parameter ID from this table.

The values of the parameters are different for each user entity. Using the parameter ID, FastFormula accesses the value of the parameter for the relevant user entity in table FF_ROUTE_PARAMETER_VALUES. Since each user entity has a different set of parameter values, the text of the route is different for each user entity.

In our example, only one user entity uses the route so there are no route parameters.

Contexts and Route Context Usage

The route may contain another type of placeholder of the form &B# (where # is a sequence number). These placeholders reference contexts in the table FF_ROUTE_CONTEXT_USAGES. FastFormula identifies the ID of the context from

this table, and then the name of the context from table FF_CONTEXTS. Contexts are predefined in FF_CONTEXTS and you should not change them. Examples are Payroll ID, Organization ID, and Date Earned.

The value of the context is not fixed. It is passed through by the formula at run time.

In our example, the route requires one context, which is Assignment ID.

Formula Types and Formula Type Context Usage

When you define a formula, you assign it to a formula type, such as Payroll formulas or QuickPaint formulas. The type of the formula determines the contexts for which it provides values. This is defined in table FF_FTYPE_CONTEXT_USAGES.

For example, a QuickPaint formula feeds through values for the contexts Assignment ID and Date Earned. Thus, when you define a QuickPaint formula, you can use database items that require the contexts Assignment ID and Date Earned. However, any database items that use the other contexts in their routes are not available to you. They do not appear in the list of values.

This is a mechanism to restrict the database items that a formula can use. It can only use database items that are appropriate to the formula context.

It follows that if a database item is based on a route that does not require any contexts (for example, a SELECT from DUAL), then every formula type in the system is able to access the database item.

Summary of How FastFormula Uses the Dictionary

1. FastFormula gets the value in the DEFINITION_TEXT column of FF_DATABASE ITEMS and puts it in the SELECT clause of the SQL.
2. It gets the user entity ID from FF_DATABASE ITEMS and uses it to get the route ID from FF_USER_ENTITIES.
3. It uses the route ID to get the route text from FF_ROUTES and puts it in the FROM clause of the SQL.
4. If the route contains a placeholder of the form &U#, FastFormula accesses FF_ROUTE_PARAMETERS to identify the parameter ID. Then it uses the parameter ID to get the value of the parameter for the relevant user entity in table FF_ROUTE_PARAMETER_VALUES.
5. If the route contains a placeholder of the form &B#, FastFormula accesses FF_ROUTE_CONTEXT_USAGES to identify the context ID. Then it uses the context ID to get the name of the context in table FF_CONTEXTS. This must be one of the contexts for which the formula passes through values (determined by the formula type in table FF_FTYPE_CONTEXT_USAGES).

Defining New Database Items

Before defining new items, you should consider the following issues:

- To which business group and legislation should the database item be available?
- Can the database item have a null value? Can it be non-existent?

Availability of Database Items

The two attributes Business Group ID and Legislation Code are associated with each user entity. These attributes determine the availability of the database items belonging to the user entity. If the Business Group ID is set to a particular value, then only formulas operating under that business group can 'see' the database item. If the Business

Group ID is set to null, the database item can be 'seen' by all business groups. The same principle applies to Legislation Code.

New database items that you define must be associated with a specific business code and legislation. Generic startup items supplied as part of the core system are available to all formulas. Your localization group has added legislation-specific items that are available to all business groups under that legislation.

> **Note:** The name of the database item must be unique within a business group.

Null & Not Found Conditions

To enable validation, you must define two flags in the FastFormula Application Dictionary:

- The NULL_ALLOWED_FLAG is a column on the table FF_DATABASE_ITEMS, and hence applies to each database item. If the SQL statement to extract the database item may return a null value, you must set this flag to yes (Y). If you set the flag to no and a null value is returned, FastFormula will report an error.
- The NOTFOUND_ALLOWED_FLAG is a column on the table FF_USER_ENTITIES, and hence applies to all the database items belonging to a particular user entity. If the SQL statement to extract database items may return no rows for any of the items, you must set this flag to yes ('Y'). If you set the flag to no and the SQL statement fails to return a row, FastFormula will report an error.

The formula writer must provide a default for a database item used in a formula, unless both of these flags are set to no. For more information on defaults, refer to the guide *Using Oracle FastFormula*.

Steps To Generate A Database Item

To illustrate the steps to generate database items, we will use the example of a user entity called GRADE_RATE_USER_ENTITY, which comprises three database items:

- GRADE_VALUE
- GRADE_MINIMUM
- GRADE_MAXIMUM

This user entity may share its route (GRADE_ROUTE) with other user entities. Each user entity uses a unique value for the route parameter RATE_ID, so that the WHERE clause for each entity is different. If the entities are in the same business group, the USER_NAME of each database item must be unique. One way to achieve this is to include the rate name in the USER_NAME; for example: <RATE_NAME>_GRADE_VALUE.

In this example, we suppose that the value of RATE_ID for GRADE_RATE_USER_ENT ITY is 50012. For simplicity we consider only one user entity for the route.

The three database items are stored in table PAY_GRADE_RULES. To extract these items, FastFormula uses an assignment ID passed by the formula. This is the formula context.

This is the SQL required to extract these database items:

```
SELECT <DEFINITION_TEXT>
 FROM   pay_grade_rules                    TARGET
 ,      per_assignments                    ASSIGN
 WHERE  TARGET.grade_or_spinal_point_id  = ASSIGN.grade_id
 AND    TARGET.rate_type                 = 'G'
 AND    ASSIGN.assignment_id             = &B1
 AND    TARGET.rate_id                   = &U1
```

<DEFINITION_TEXT> may be one of the three database items listed below:

Database Item Name	<DEFINITION_TEXT>
GRADE_VALUE	TARGET.value
GRADE_MINIMUM	TARGET.minimum
GRADE_MAXIMUM	TARGET.maximum

The following steps describe how to load the information into the Dictionary so that FastFormula can generate this SQL. An example of PL/SQL that loads the information is given at the end of this section.

1. Write the SQL

   Write and test the SQL statement using SQL*Plus to ensure that the statement is correct. The SQL statement must not return more than one row because FastFormula cannot process multiple rows.

2. Load the Route

   This is best done using a PL/SQL routine. Wherever possible, use the sequence value for the primary keys (such as FF_ROUTES_S.NEXTVAL) to populate the table. The route is held in the table FF_ROUTES as a 'long' data type. So, using the example above, you could assign the route to a long variable as follows:

   ```
   set escape \
    DECLARE
      l_text   long;
    BEGIN
        l_text := '/* route for grade rates */
           pay_grade_rules                        TARGET,
           per_assignments                        ASSIGN
    WHERE  TARGET.grade_or_spinal_point_id      = ASSIGN.grade_id
    AND    TARGET.rate_type                     = ''G''
    AND    ASSIGN.assignment_id                 = \&B1
    AND    TARGET.rate_id                       = \&U1';
    END;
   ```

   Note the following changes from the original SQL that was given earlier:

   - Each '&' is preceded with the escape character.
   - The single quote mark is replaced with two single quote marks.
   - A comment may be placed at the start of the route if required.

3. Load the Contexts

The next step is to load the contexts into the table FF_ROUTE_CONTEXT_USAGES. The columns in this table are as follows:

Name	Null?	Type
ROUTE_ID	NOT NULL	NUMBER(9)
CONTEXT_ID	NOT NULL	NUMBER(9)
SEQUENCE_NO	NOT NULL	NUMBER(9)

Use the current sequence number for the route ID. This is FF_ROUTES_S.CURRVAL if you used the sequence FF_ROUTES_S.NEXTVAL to populate the table FF_ROUTES. You can obtain the context ID for the particular formula context (assignment ID in our example) from the table FF_CONTEXTS. The sequence number is simply the 'B' number.

For the example, you would insert one row for the route into the table FF_ROUTE_CONTEXT_USAGES (see the PL/SQL for the example, at the end of this section).

4. Insert Rows in the User Entity Table

   For each route, insert at least one row in the table FF_USER_ENTITIES. This table holds the Business Group ID, Legislation Code, the ROUTE_ID, and the NOTFOUND_ALLOWED_FLAG.

5. Insert Rows for Route Parameters

   For each placeholder of the form &U# in the route, you must insert a row into two tables:

   - FF_ROUTE_PARAMETERS, which references the route, and
   - FF_ROUTE_PARAMETER_VALUES, which contains the actual value for the route parameter, and references the user entity.

   The columns in these tables are as follows:

SQL> desc ff_route_parameters

Name	Null?	Type
ROUTE_PARAMETER_ID	NOT NULL	NUMBER(9)
ROUTE_ID	NOT NULL	NUMBER(9)
DATA_TYPE	NOT NULL	VARCHAR2(1)
PARAMETER_NAME	NOT NULL	VARCHAR2(80)
SEQUENCE_NO	NOT NULL	NUMBER(9)

SQL> desc ff_route_parameter_values

Name	Null?	Type
ROUTE_PARAMETER_ID	NOT NULL	NUMBER(9)
USER_ENTITY_ID	NOT NULL	NUMBER(9)
VALUE	NOT NULL	VARCHAR2(80)
LAST_UPDATE_DATE		DATE
LAST_UPDATED_BY		NUMBER(15)
LAST_UPDATE_LOGIN		NUMBER(15)
CREATED_BY		NUMBER(15)
CREATION_DATE		DATE

The data type held in FF_ROUTE_PARAMETERS is either a number (N) or a text value (T).

In our example, the route parameter is RATE_ID. For GRADE_RATE_USER_ENT ITY, its value is 50012. The values you would insert into these tables for the example are shown in the sample PL/SQL at the end of this section.

6. Insert the Database Item

You can now insert the database items. For our example, there are three rows in the table FF_DATABASE_ITEMS that refer to the same user entity. The columns in this table are as follows:

```
SQL> desc ff_database_items
```

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(80)
USER_ENTITY_ID	NOT NULL	NUMBER(9)
DATA_TYPE	NOT NULL	VARCHAR2(1)
DEFINITION_TEXT	NOT NULL	VARCHAR2(240)
NULL_ALLOWED_FLAG	NOT NULL	VARCHAR2(1)
DESCRIPTION		VARCHAR2(240)
LAST_UPDATE_DATE		DATE
LAST_UPDATED_BY		NUMBER(15)
LAST_UPDATE_LOGIN		NUMBER(15)
CREATED_BY		NUMBER(15)
CREATION_DATE		DATE

The USER_NAME must be unique within the business group.

The values you would insert into this table for the three example database items are shown in the sample PL/SQL at the end of this section.

When you create the database items, it is useful to populate the other columns, such as LAST_UPDATE_DATE, and CREATION_DATE.

Example

The following PL/SQL creates the database items in the example::

```
set escape \
DECLARE
  l_text                long;
  l_user_entities_seq   number;
  l_route_id            number;
BEGIN
 --
 -- assign the route to a local variable
 --
l_text := '/* route for grade rates */
       pay_grade_rules                          TARGET,
       per_assignments                          ASSIGN
WHERE  TARGET.grade_or_spinal_point_id       = ASSIGN.grade_id
AND    TARGET.rate_type                      = ''G''
AND    ASSIGN.assignment_id                  = \&B1
AND    TARGET.rate_id                        = \&U1';
--
-- insert the route into the table ff_routes
--
insert into ff_routes
       (route_id,
        route_name,
        user_defined_flag,
        description,
        text,
        last_update_date,
        creation_date)
values (ff_routes_s.nextval,
        'GRADE_ROUTE',
        'Y',
        'Route for grade rates',
        l_text,
        sysdate,
        sysdate);
--
-- load the context
--
insert into ff_route_context_usages
       (route_id,
        context_id,
        sequence_no)
select ff_routes_s.currval,
       context_id,
       1
from   ff_contexts
where  context_name = 'ASSIGNMENT_ID';
--
```

```
-- create a user entity
--
select ff_user_entities_s.nextval
into   l_user_entities_seq
from   dual;
--
select ff_routes_s.currval
into   l_route_id
from   dual;
--
insert into ff_user_entities
       (user_entity_id,
        business_group_id,
        legislation_code,
        route_id,
        notfound_allowed_flag,
        user_entity_name,
        creator_id,
        creator_type,
        entity_description,
        last_update_date,
        creation_date)
values (l_user_entities_seq,
        1,                              -- example business group id
        'GB',                           -- example legislation
        l_route_id,
        'Y',
        'GRADE_RATE_USER_ENTITY',
        50012,                          -- example creator id
        'CUST',
        'Entity for the Grade Rates',
        sysdate,
        sysdate);
--
-- insert the route parameters
--
insert into ff_route_parameters
       (route_parameter_id,
        route_id,
        data_type,
        parameter_name,
        sequence_no)
select  ff_route_parameters_s.nextval,
        l_route_id,
        'N',
        'Grade Rate ID',
        1
from    dual;
--
insert into ff_route_parameter_values
       (route_parameter_id,
        user_entity_id,
        value,
        last_update_date,
        creation_date)
select ff_route_parameters_s.currval,
       l_user_entities_seq,
       50012,
```

```
       sysdate,
       sysdate
from   dual;
--
-- insert the three database items
--
insert into ff_database_items
       (user_name,
        user_entity_id,
        data_type,
        definition_text,
        null_allowed_flag,
        description,
        last_update_date,
        creation_date)
values ('GRADE_VALUE',
        l_user_entities_seq,
        'T',
        'TARGET.value',
        'Y',
        'Actual value of the Grade Rate',
        sysdate,
        sysdate);
--
insert into ff_database_items
       (user_name,
        user_entity_id,
        data_type,
        definition_text,
        null_allowed_flag,
        description,
        last_update_date,
        creation_date)
values ('GRADE_MINIMUM',
        l_user_entities_seq,
        'T',
        'TARGET.minimum',
        'Y',
        'Minimum value of the Grade Rate',
        sysdate,
        sysdate);
```

```
--
insert into ff_database_items
       (user_name,
        user_entity_id,
        data_type,
        definition_text,
        null_allowed_flag,
        description,
        last_update_date,
        creation_date)
values ('GRADE_MAXIMUM',
        l_user_entities_seq,
        'T',
        'TARGET.maximum',
        'Y',
        'Maximum value of the Grade Rate',
        sysdate,
        sysdate);
END;
/
```

Calling FastFormula from PL/SQL

Oracle FastFormula provides an easy to use tool for professional users. Using simple commands and syntax, users can write their own validation rules or payroll calculations.

Until R11 the execution engine for calling formulas and dealing with the outputs has been hidden within the Oracle HR and Payroll products. The original engine for calling PL/SQL was written in Pro*C. It is complex and can be called only from user exits or directly from another 'C' interface.

Now, there is a new execution engine or interface that lets you call formulas directly from Forms, Reports or other PL/SQL packages. This interface means that you can call existing validation or payroll formulas and include them in online or batch processes. It also means that you can define and call your own formulas for other types of validation and calculation. With FastFormula you automatically have access to the database items (DBIs) and functions of Oracle HRMS and you automatically have calculations and business rules that are datetracked.

The basic concepts of FastFormula remain the same as before:

Inputs -> Process -> Outputs

As you now have complete freedom to decide the inputs you provide and what happens to the outputs produced by a formula you must write the calling code to handle both inputs and outputs.

For optimal performance when calling FastFormula from PLSQL, generate the Formula Wrapper after compiling the formula. You can execute a formula even if you did not compile it before you generated the Formula Wrapper. The Bulk Compile Formulas process automatically generates the Formula Wrapper.

Generate the Formula Wrapper only when necessary. The Formula Wrapper generates a PLSQL package body, and the generation process may cause runtime errors in FastFormula calls that occur at the same time. You do not need to generate the Formula Wrapper when you test formulas.

This essay provides an overview and technical details to show you how to call FastFormula from PL/SQL. You should be familiar with PL/SQL coding techniques and with Oracle FastFormula but you will not need to understand the internal working of the execution engine.

The Execution Engine Interface

There are two interfaces to the execution engine for FastFormula.

- Server-side

  Use this interface for any formulas to be executed by batch processes or on the server. See: Server Side Interface, page 2-191

- Client-side

  Use this interface only when a direct call is required from forms and reports to execute a formula immediately. You could also write a custom 'wrapper' package to call the server engine from the client. See: Client Side Call Interface, page 2-196

  Note: Some Oracle tools currently use PL/SQL V1.x only. This version does not support the table of records data structure needed by the server interface. The client-side version was written to get around this current limitation.

Location of the Files

The execution engine files are stored in **$FF_TOP/admin/sql**

- ffexec.pkh and ffexec.pkb

  Server side execution engine package header and body files.

- ffcxeng.pkh and ffcxeng.pkb

  Client side versions of execution engine package header and body files.

  Note: There is a special interface in the *ff_client_engine* module that is designed specifically for the forms client. This interface avoids the overhead of a large number of network calls using a fixed number of parameters. See: Special Forms Call Interface, page 2-200

Datetracked Formulas

All formulas in Oracle HRMS products are datetracked, enabling you to use DateTrack to maintain a history of changes to your validation rules or calculations.

In the predefined interfaces to the execution engine the system automatically manages the setting or changing of the effective date. When you execute your own formulas you must also manage the setting of the effective date for the session. This means that before calling any of the execution engine interfaces you may need to insert a row into the FND_SESSIONS table. This is required if there is no row in FND_SESSIONS for the current SQL*PLUS session_id or the formula or formulas to be executed access database items that reference datetracked tables.

Important: Always check the effective date for the formula to be executed. This date affects the values of the database items and any functions that you include in the formula.

Changes in R11i

Server Side and Client Side Interfaces

In R11i the client side interfaces are provided for backwards compatibility. The client side PL/SQL environments used with R11i are able to use the server side interface.

NUMBER and DATE Inputs and Outputs

Input values must be passed in as strings in the correct formats. In R11i, use the routine FND_NUMBER.NUMBER_TO_CANONICAL to format NUMBER inputs. Use FND_DATE.DATE_TO_CANONICAL to format DATE inputs.

Output values are passed back as strings formatted as described above. To convert a NUMBER output to a NUMBER value, use the routine FND_NUMBER.CANONICAL_TO_NUMBER. Use FND_DATE.CANONICAL_TO_DATE to convert DATE outputs to DATE values.

For forms code, using the corresponding routines from the APP_NUMBER and APP_DATE packages may result in improved performance.

This set of changes applies to all the interfaces to the FastFormula execution engine.

DATE_EARNED and BALANCE_DATE Contexts

In R11i, the datatype of DATE_EARNED and BALANCE_DATE contexts is DATE. Prior to R11i, these contexts had a datatype of TEXT.

Server Side Interface

This section describes the interface to the server execution engine and how to call the module from other PL/SQL.

This version of the interface is preferred. It combines maximum flexibility with relatively low network demands. However, it can only be used with PL/SQL V2.3 and above as it requires support for the table of records data structure.

User Data Structures

There are two important user data structures when you use the server side interface. These are the inputs table and the outputs table:

Inputs Table

Name	Description
NAME	The input name, such as RATE, or ASSIGNMENT_ID
DATATYPE	Can be DATE, NUMBER, or TEXT
CLASS	The type of input : CONTEXT or INPUT This field is not required, as it is not necessary to know if an input is a context or a normal input value to call the formula correctly.
VALUE	The actual value to pass to the formula as a Context or an Input. This field is a type of varchar2(240). This means that for NUMBER and DATE datatypes the value passed in has to be in the appropriate format. See the example code for how this works.

Outputs Table

Name	Description
NAME	The output name, such as RESULT1, or MESSAGE
DATATYPE	Can be DATE, NUMBER, or TEXT
VALUE	The actual value returned from the formula

> **Note:** The names of all inputs and outputs must be in upper case and the same name can appear in both the inputs and the outputs tables, for example where an input value is also a return value from the formula. However, a CONTEXT can only appear in the inputs table.

Both inputs and outputs tables are initialized by a call to the *ff_exec.init_formula* procedure and then contain details of all the inputs, including contexts that are needed to execute the formula and all the outputs that will be returned.

You are responsible for holding these tables between the initialization and execution calls.

> **Important:** Although the index values for these tables are positive in value, the caller should not assume that they start at 1. Always use the "first" and "last" table attributes when accessing and looping through these tables. See also: Examples, page 2-193.

Available Calls

The following procedure calls are available. They are described below with some detail on the parameters that can be passed to them.

> **Note:** Refer to the appropriate package header for information on the class of parameter (in, out or in/out).

Procedure : init_formula

This call initializes the execution engine for a specific formula. That is, it declares to the engine that a formula is about to be run. It must be called before a formula is executed, but that formula can then be executed as many times as desired without having to call the initialization procedure again. This will be understood from the examples further on.

Table of parameters to init_formula

Parameter Name	Data Type	Comments
p_formula_id	number	Formula_id to execute
p_effective_date	date	Effective date to execute
p_inputs	ff_exec.inputs_t	Input variable information
p_outputs	ff_exec.outputs_t	Output variable information

Procedure : run_formula

This call actually executes the formula, taking inputs as specified and returning any results from the formula. The init_formula procedure must have been called before this is used (see examples).

Table of parameters to run_formula

Parameter Name	Data Type	Comments
p_inputs	ff_exec.inputs_t	Inputs to the formula
p_outputs	ff_exec.outputs_t	Outputs from the formula
p_use_dbi_cache	boolean	If TRUE, the database item cache will be active during execution, else will not. Defaults to TRUE

Further Comments

The p_inputs and p_outputs parameters could be NULL if the formula does not have any inputs and/or outputs (although the latter is rather unlikely).

The p_use_dbi_cache would only be set to FALSE under unusual circumstances requiring the disabling of the cacheing of database item values. This might be required if the engine is called from code that would invalidate the values for fetched database items.

For instance, if the database item ASG_STATUS was accessed from within a formula used in business rule validation used in turn to alter the Assignment's status, we might want to disable the Database Item cache in case we attempted to read that database item in a subsequent formula.

Examples

The following examples assume we are going to execute the following formula. Note that the DATABASE_ITEM requires an ASSIGNMENT_ID context.

The formula itself does not represent anything meaningful, it is for illustration only.

```
inputs are input1, input2 (date), input3 (text)

dbi = DATABASE_ITEM

ret1 = input1 * 2

return ret1, input2, input3
```

The following anonymous block of PL/SQL could be used to execute the formula. In this case, it is called a number of times, to show how we can execute many times having initialized the formula once.

```
declare

  l_input1         number;

  l_input2         date;

  l_input3         varchar2(80);
```

```
  l_assignment_id  number;
  l_formula_id     number;
  l_effective_date date;
  l_inputs         ff_exec.inputs_t;
  l_outputs        ff_exec.outputs_t;
  l_loop_cnt       number;
  l_in_cnt         number;
  l_out_cnt        number;
begin
  -- Set up some the values we will need to exec formula.
  l_formula_id     := 100;
  l_effective_date := to_date('06-05-1997', 'DD-MM-YYYY');
  l_input1         := 1000.1;
  l_input2         := to_date('01-01-1990', 'dd-mm-yyyy');
  l_input3         := 'INPUT TEXT';
  l_assignment_id  := 400;
  -- Insert FND_SESSIONS row.
  insert into fnd_sessions (
 session_id,
 effective_date)
  values (userenv('sessionid'),
 l_effective_date);
  -- Initialise the formula.
  ff_exec.init_formula(l_formula_id, l_effective_date, l_inputs,
l_outputs);
  -- We are now in a position to execute the formula.
  -- Notice that we are illustrating here that the formula can
  -- be executed a number of times, in this case setting a new
  -- input value for input1 each time.
```

```
  for l_loop_cnt in 1..10 loop

    -- The input and output table have been initialized.  We now
have

    -- to set up the values for the inputs required.  This includ
es

    -- those for the 'inputs are' statement and any contexts.

    for l_in_cnt in l_inputs.first..l_inputs.last loop

      if(l_inputs(l_in_cnt).name = 'INPUT1') then

        -- Deal with input1 value.

        l_inputs(l_in_cnt).value := fnd_number.number_to_canonica
l(l_input1);

      elsif(l_inputs(l_in_cnt).name = 'INPUT2') then

        -- Deal with input2 value.

        l_inputs(l_in_cnt).value := fnd_date.date_to_canonical(l_
input2);

      elsif(l_inputs(l_in_cnt).name = 'INPUT3') then

        -- Deal with input3 value.

        l_inputs(l_in_cnt).value := l_input3;

        -- no conversion required.

      elsif(l_inputs(l_in_cnt).name = 'ASSIGNMENT_ID') then

        -- Deal with the ASSIGNMENT_ID context value.

        l_inputs(l_in_cnt).value := l_assignment_id;

      end if;

    end loop;

    ff_exec.run_formula(l_inputs, l_outputs);

    -- Now we have executed the formula. We are able

    -- to display the results.

    for l_out_cnt in l_outputs.first..l_outputs.last loop

      hr_utility.trace('output name     : ' || l_outputs(l_out_cn
t).name);

      hr_utility.trace('output datatype : ' || l_outputs(l_out_cn
t).datatype);
```

```
      hr_utility.trace('output value    : ' || l_outputs(l_out_cn
t).value);

    end loop;

  end loop;

  -- We can now continue to call as many formulas as we like,

  -- always remembering to begin with a ff_exec.init_formula call
.

  -- Note: There is no procedure to be called to

  -- shut down the execution engine.

end;

/
```

As noted earlier, if you are attempting to call the execution engine from a client that is not running the appropriate version of PL/SQL, it will be necessary to create a package that 'covers' calls to the engine or consider calling the client engine, specified below.

Client Side Call Interface

This section attempts to describe in detail the interface to the client execution engine from a user perspective, and how to call the module from other PL/SQL.

> **Note:** These client side calls are designed to avoid any use of overloading, which causes problems when procedures are called from forms.

When Should I Use This Interface?

This interface can be used when the version of PL/SQL on the client is prior to V2.3 (does not support tables of records). It is probably the easiest interface to use. However, it is not recommended where high performance is required, due to the greater number of network round-trips. In these cases, consider using the special forms interface.

User Data Structures

There are no user visible data structures in the client side call.

Available Calls

The following procedure calls are available. They are described below with some detail on the parameters that can be passed to them.

> **Note:** Refer to the appropriate package header for information on the class of parameter (in, out, or in/out).

Procedure : init_formula

This call initializes the execution engine for a specific formula. That is, it declares to the engine that a formula is about to be run. It must be called before a formula is executed, but that formula can then be executed as many times as desired without having to call the initialization procedure again. This will be understood from the examples further on.

Table of parameters to init_formula

Parameter Name	Data Type	Comments
p_formula_id	number	Formula_id to execute
p_effective_date	date	Effective execution date

Procedure : set_input

This call sets the value of an input to a formula. To cope with the different datatypes that FastFormula can handle, the values have to be converted to the appropriate character strings.

Table of parameters to set_input

Parameter Name	Data Type	Comments
p_input_name	varchar2	Name of input to set
p_value	varchar2	Input value to set

Procedure : run_formula

This call actually executes the formula, taking inputs as specified and returning any results from the formula. The init_formula procedure must have been called before this is used (see examples).

There are no parameters to run_formula.

Procedure : get_output

This call gets the output values returned from a formula. To cope with the different datatypes that FastFormula can handle, the output has to be converted as appropriate.

Table of parameters to get_output

Parameter Name	Data Type	Comments
p_input_name	varchar2	Name of input to set
p_return_value	varchar2	Value of varchar2 output

Examples

The following examples rely on the same formula used above.

```
inputs are input1, input2 (date), input3 (text)

dbi = DATABASE_ITEM

ret1 = input1 * 2

return ret1, input2, input3
```

The following anonymous block of PL/SQL can be used to run the formula.

```
declare

  l_input1         number;

  l_input2         date;

  l_input3         varchar2(80);

  l_output1        number;

  l_output2        varchar2(12);

  l_output3        varchar2(80);

  l_assignment_id  number;

  l_formula_id     number;

  l_effective_date date;

  l_loop_cnt       number;

begin

-- Set up the values we need to execute the formula.

  l_formula_id     := 100;

  l_effective_date := to_date('06-05-1997', 'DD-MM-YYYY');

  l_input1         := 1000.1;

  l_input2         := to_date('01-01-1990', 'dd-mm-yyyy');

  l_input3         := 'INPUT TEXT';

  l_assignment_id  := 400;

-- Insert FND_SESSIONS row.

insert into fnd_sessions (

    session_id,

    effective_date)

values (userenv('sessionid'),

     l_effective_date);

-- Initialize the formula. ff_client_engine.init_formula(l_formul
a_id,l_effective_date);

-- We are not in a position to execute the formula.

-- Notice that we are illustrating here that the formula can
```

```
-- be executed a number of times, in this case setting a new

-- input value for input1 each time.

  for l_loop_cnt in 1..10 loop

-- The input and output tables have been initialized.

-- We now have to set up the values for the inputs required.

-- This includes those for the 'inputs are' statement

-- and any contexts.

-- Note how the user has to know the number of inputs the

-- formula has.

    ff_client_engine.set_input('INPUT1', fnd_number.number_to_can
onical(l_input1));

    ff_client_engine.set_input('INPUT2', fnd_date.date_to_canonic
al(l_input2));

    ff_client_engine.set_input('INPUT3', l_input3);

    ff_client_engine.set_input('INPUT3', l_input3);

    ff_client_engine.set_input('ASSIGNMENT_ID', l_assignment_id);

    ff_client_engine.run_formula;

-- Now we have executed the formula. Get the results.

    ff_client_engine.get_output('RET1',   l_output1);

    ff_client_engine.get_output('INPUT2', l_output2);

    ff_client_engine.get_output('INPUT3', l_output3);

-- OK. Finally, display the results.

    hr_utility.trace('RET1 value   : ' || output1);

    hr_utility.trace('INPUT2 value : ' || l_output2);

    hr_utility.trace('INPUT3 value : ' || output3)

  end loop;

-- We can now continue to call as many formulas as we like,

-- always remembering to begin with a

-- ff_client.init_formula call.

-- Note: There is no procedure to be called to
```

```
-- shut down the execution engine.

end;

/
```

Special Forms Call Interface

This section attempts to describe in detail the interface to the special forms client execution engine interface from a user perspective, and how to call the module from forms.

When Should I Use This Interface?

This interface is recommended for use when you want to execute a formula directly from a form or report client that does not support PL/SQL V2.3 or above (that is, does not allow PL/SQL tables of records).

User Data Structures

There are no user visible data structures in the client side call.

Available Calls

The following procedure calls are available. They are described below with some detail on the parameters that can be passed to them.

> **Note:** Refer to the appropriate package header for information on the class of parameter (in, out, or in/out).

Procedure : run_id_formula

This call initializes the execution engine for a specific formula, then runs the formula taking the input and context arguments specified. Finally it returns the appropriate results to the user via a further set of arguments. This form of call therefore requires only one network round-trip. The disadvantage is that it is limited to the number of inputs and returns that it can cope with (this is based round the PL/SQL V1.0 limitations).

> **Note:** Use this procedure call when the formula_id for the formula to execute is known. Another procedure call (run_name_formula - see below) is used where only the name is known.

Table of parameters to run_id_formula

Parameter Name	Data Type	Comments
p_formula_id	number	Formula_id to execute
p_effective_date	date	Effective execution date
p_input_name01 . . . 10	varchar2	input name 01 . . . 10
p_input_value01 . . . 10	varchar2	input value 01 . . . 10
p_context_name01 . . . 14	varchar2	context name 01 . . . 14
p_context_value01 . . . 14	varchar2	context value 01 . . . 14
p_return_name01 . . . 10	varchar2	return name 01 . . . 10
p_return_value01 . . . 10	varchar2	return value 01 . . . 10

Procedure : run_name_formula

This call initializes the execution engine for a specific formula, then runs the formula taking the input and context arguments specified. Finally it returns the appropriate results to the user via a further set of arguments. This form of call therefore requires only one network round-trip. The disadvantage is that it is limited to the number of inputs and returns that it can cope with (this is based round the PL/SQL V1.0 limitations).

> **Note:** Use this procedure call when you know the name and type for the formula to execute. Use the run_id_formula call (see above) when only the id is known.

Table of parameters to run_name_formula

Parameter Name	Data Type	Comments
p_formula_ type_name	number	Formula type
p_formula_ name	varchar2	Name of formula to execute
p_effective_ date	date	Effective execution date
p_input_ name01 . . . 10	varchar2	input name 01 . . . 10
p_input_ value01 . . . 10	varchar2	input value 01 . . . 10
p_context_ name01 . . . 14	varchar2	context name 01 . . . 14
p_context_ value01 . . . 14	varchar2	context value 01 . . . 14
p_return_ name01 . . . 10	varchar2	return name 01 . . . 10
p_return_ value01 . . . 10	varchar2	return value 01 . . . 10

Logging Options

Sometimes things may go wrong when attempting to execute formulas via the PL/SQL engine. In many cases, the error messages raised will make it obvious where the problem is. However, there are cases where some more information is needed.

You can set the execution engine to output logging information. This section explains how to activate and use the logging options

> **Note:** The logging output makes use of the standard Oracle HR trace feature.

Enabling Logging Options

You set logging options for the execution engine by calling the ff_utils.set_debug procedure. This procedure has the definition:

```
procedure set_debug
(
  p_debug_level in binary_integer
);
```

Since the numeric values for the options are power of two values, each represented by a constant, the appropriate values are added together.

For instance, to set the routing and dbi cache debug options (see below) use the following call (from SQLPLUS).

```
SQL> execute ff_utils.set_debug(9)
```

The value 9 is (1 + 8).

If preferred, you can use the constants that have been defined. For example:

```
SQL> execute ff_utils.set_debug(ff_utils.ROUTING +

ff_exec.DBI_CACHE_DBG)
```

FF_DEBUG Profile Option

If the execution engine is being called from a form, you can enable logging options using the FF_DEBUG profile option.

You use a series of characters to indicate which logging options you want to set. You must specify X, as this enables user exit logging. For example, if you set the profile option to XDR, you initiate the database item cache and routing information.

The full list of characters you can specify is as follows (see Summary of Available Information for a description of each logging option).

Table of Values for FF_DEBUG Profile Option

Character	Equivalent to . . .
R	ff_utils.ROUTING
F	ff_exec.FF_DBG
C	ff_exec.FF_CACHE_DBG
D	ff_exec.DBI_CACHE_DBG
M	ff_exec.MRU_DBG
I	ff_exec.IO_TABLE_DBG

Summary Of Available Information

What follows is a brief discussion of each logging option, with its symbolic and equivalent binary value used to set it.

> **Note:** To interpret the output of many of these options, you require some familiarity with the workings of the execution engine code.

ff_utils.ROUTING : 1

Routing. Outputs information about the functions and procedures that are accessed during an execution engine run. An example of the visible output would be:

- In : run_formula
- Out : run_formula

ff_exec.FF_DBG : 2

This debug level, although defined in the header, is not currently used.

ff_exec.FF_CACHE_DBG : 4

Formula Cache Debug. Displays information about the currently executing formula, including its data item usage rows.

ff_exec.DBI_CACHE_DBG: 8

Database Item Cache Debug. Displays information about those items held in the database item cache. These items are not constrained to a particular formula.

ff_exec.MRU_DBG : 16

Most Recently Used Formula chain. Displays information about those formulas currently held in the MRU chain. The information displayed includes the table index, formula_id, sticky flag and formula name.

ff_exec.IO_TABLE_DBG : 32

Input and Output Table Debug. Shows information about items currently held in the input and output tables. This includes both information set by the user and the formula engine.

How Should the Options Be Used?

Only general advice can be given, since there is no way of predicting what the problem may be. Some hints are:

ROUTING is useful only for those who understand the code. Tracing the procedures may illuminate a problem - perhaps an error is being raised and it is not obvious where from.

FF_CACHE_DBG will confirm what basic formula information is held by the execution engine. This is useful to see if it looks as you expect.

IO_TABLE_DBG will confirm what is really being passed to and from a formula.

Flexfields

Validation of Flexfield Values

Oracle Self Service HR, Web ADI and some forms use the HRMS APIs to record data in the database. Custom programs at your site, such as data upload programs, may also use the APIs.

From Release 11i (and R11.0 Patch Set D), the APIs validate flexfield values using value sets (in the same way as the professional Forms user interface). This provides the benefit that value set definitions only need to be implemented and maintained in one location. In previous releases, the APIs validated flexfield values using PL/SQL callouts to Skeleton Flexfield Validation server-side packages. These packages are no longer used.

This essay explains how to solve some problems you may encounter when the APIs use flexfield value sets. These problems occur when the value sets refer to objects that are not automatically available to API validation.

In summary, problems may occur when value sets refer to:

- User profile options
- Form block.field items
- A row in the FND_SESSIONS database table

Problems may also be caused by:

- Incomplete context field value lists

The rest of this essay explains these issues in more detail with recommended solutions. For all of these solutions, the changes are not apparent to end users and it is not necessary to change where the data is physically held in the database.

Referencing User Profile Options

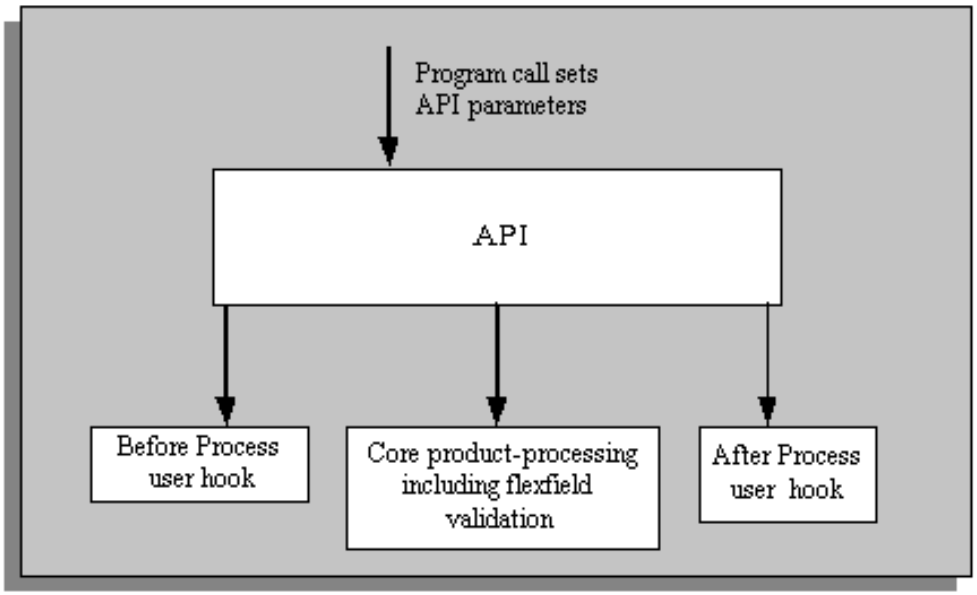
Referencing profile options in value sets does not cause a problem in the Professional Forms UI or Self Service HR. When a user logs on to these interfaces, the profiles are available, defined at site, application, responsibility, or user level.

However, when the APIs are executed directly in a SQL*Plus database session, there is no application log-on. If the profile is not defined at site level, its value will be null. Even if the profile is defined at site level, this may not give the appropriate values. For example, the PER_BUSINESS_GROUP_ID profile is defined at site level with a value of zero, for the Setup Business Group. If you do not use the Setup Business Group, the flexfield validation finds no rows and all data values are rejected as invalid.

Recommended Solution

Ensure any profiles you reference in value sets are set to the appropriate values before the flexfield validation is performed. You can do this using API user hooks. The following example uses the PER_BUSINESS_GROUP_ID profile.

Using API User Hooks to Set Business Group ID



hr_401.gif

Define a Before Process user hook call to set the PER_BUSINESS_GROUP_ID profile. Where the API user hook provides a mandatory p_business_group_id parameter, the profile can be set directly from this parameter value. Otherwise first derive the business_group_id value from the database tables using the API's mandatory primary key parameter value.

The PER_BUSINESS_GROUP_ID profile must only be populated when it is undefined or set to zero. If the profile is defined with a non-zero value then it should not be changed. This is to ensure there is no impact on the Professional UI and Self Service HR.

The Before Process user hook package should also remember when it has actually set the PER_BUSINESS_GROUP_ID profile. This can be done with a package global variable.

The second part of the solution is to define an After Process user hook to reset the PER_BUSINESS_GROUP_ID profile back to its original zero or null value. This is only necessary when the Before Process actually changed the value. This is to ensure the profile will be populated with the correct value when the API is called a second time.

For further information on using API user hooks, see the "APIs in Oracle HRMS", page 2-217 essay.

Alternative Solution

If you have only one program experiencing this problem, you could modify the program to set the PER_BUSINESS_GROUP_ID profile immediately before each API call. However, if you introduce any other programs in the future calling the same API, you would have to remember to set the PER_BUSINESS_GROUP_ID profile in these programs too.

Referencing Form block.field Items

If a value set references Form block.field items, an error is raised when the API executes the flexfield validation because the Form item values cannot be resolved on the server-side. This problem affects Oracle Self Service HR and any custom code that calls the API.

Recommended Solution

There are three parts to this solution:

1. Modify the value sets so all block.item references are changed to custom profile names. These profiles do not have to be defined within the Oracle Applications data dictionary because profiles can be created and set dynamically at run-time.
2. To ensure the modified value sets work, the profiles must be populated before the APIs execute the flexfield validation. As with the PER_BUSINESS_GROUP_ID profile problem, this requires an API Before Process user hook to populate the profile values. Some of the required values will not be immediately available from the user hook package parameters. However any missing values can be derived from the HRMS tables.
3. To ensure the flexfield validation continues to work in the Professional UI, the profile values need to be populated before the flexfield pop-up window is displayed. This can be done using the CUSTOM library. For the specific Forms when certain events occur, read the Form items to populate the custom profiles.

   Important: There may be some instances in the Self Service screens where it is not possible to display these flexfield values. This is because there is no Web page equivalent to the Forms' CUSTOM library to ensure the custom profiles are correctly populated. This will not be resolved until a future Release.

Alternative Solution

Another method would be to extend the value set Where clauses to obtain the required values from the database. This may require joins to additional database tables. This removes the need to reference Form block.field items. However, this solution is only suitable where values can be obtained from records already in the database. Attempting to reference columns on the record being processed by the current API call will fail. During an insert operation those values will not be available from the database

table when the flexfield validation executes. During an update operation the pre-update values will be obtained.

Referencing FND_SESSIONS Row

The FND_SESSIONS database table is used to obtain the current user's DateTrack effective date. This table is only maintained by the Professional UI. The APIs and Self Service modules do not insert or update any rows in this table. So when the value set is executed from these modules, the join fails to find any rows.

Recommended Solution

Using an API Before Process user hook, if a row does not already exist in the FND_SESSIONS table for this database session, then insert one. The EFFECTIVE_DATE column should be set from the p_effective_date parameter made available at the user hook. It is important to ensure the EFFECTIVE_DATE column is set to a date value with no time component, that is, trunc(<date>). Otherwise some join conditions will still fail to find valid table rows.

When the API Before Process user hook has inserted a row into FND_SESSIONS, the After Process user hook should delete it. This ensures that when a second call to the same API is made, the FND_SESSIONS.EFFECTIVE_DATE column is set to the correct value.

If performance is a concern for batch uploading of data, it may be more efficient for the batch upload program to insert the FND_SESSIONS row before the first API call. That will only be acceptable if the set of records will be processed with the same effective date. The API user hooks will still need to be defined to ensure that other programs and interfaces work as required.

Alternative Solution

Another method would be to follow the same approach as the referencing Form block.field items solution. Instead of the value set using the FND_SESSIONS table to obtain the effective date, it could use a custom profile. This avoids the insert and delete DML steps. However, there is an impact on the Professional UI so the CUSTOM library will need to be changed to set the profile value.

Incomplete Context Field Value Lists

Using the APIs, you might see the following error if a flexfield's reference value does not appear in the flexfield Context Field Values list:

ORA-20001: Column ATTRIBUTE_CATEGORY, also known as CONTEXT, cannot have value X.

Suppose a flexfield uses the business_group_id as the reference field. When the API is called, the p_attribute_category parameter should be set to the business_group_id value. When the API validates the Flexfield Context Field (ATTRIBUTE_CATEGORY), it checks whether the business_group_id being used exists in the Flexfield Context Field Values list. If not, the API raises an error.

Recommended Solution

Ensure that the flexfield Context Field Values lists contain all possible values.

Alternative Solution

In some flexfield structures, there are some contexts where only the global data elements apply (there are no context-specific segments). You might consider setting the p_attribute_category parameter to null for these context values. This avoids the need to list these context values in the Context Field Values list. However, this is not recommended because it may cause other data errors to go undetected. For example, if

the context field is set to null when a more specific value should be used, any mandatory segment validation associated with that other value will not be executed.

Security

Extending Security in Oracle HRMS

Oracle Human Resources provides a flexible approach to controlling access to tables, records, fields, forms, and functions. You can match each employee's level of access to their responsibilities.

For a discussion of security in Oracle HRMS and how to set it up to meet your requirements, refer to the help topics on Security, *Oracle HRMS Configuring, Reporting, and System Administration Guide*, and to the implementation steps for Defining User Security, page 2-69

This essay does not repeat the definitions and description in the setup steps and security chapter. It builds on that information to describe the objects and processes that implement the security system. Read this essay if you need to:

- Add custom tables to the standard security system
- Integrate your own security system with the supplied mechanisms

Security Profiles

All Oracle Applications users access the system through a responsibility that is linked to a security group and a security profile. The security group determines which business group the user can access. The security profile determines which records (related to organizations, positions and payrolls) the user can access within the business group.

There are two types of security profile:

- Unrestricted
- Restricted

A responsibility with an unrestricted security profile has unrestricted access to data in Oracle HRMS tables. It connects to the APPS Oracle User. If you connect to an unrestricted security profile, the data you see when you select from a secure view is the same data you see if you select from the table on which the secure view is based.

When you connect to the APPS Oracle User with a restricted security profile you can access the secure tables directly if you want to bypass the security restrictions defined in your security profile. You might want to do this to perform uniqueness checks, or to resolve foreign keys.

Restricted security profiles can optionally make use of read-only, or reporting users. These are separate Oracle Users, one per restricted security profile, that have read-only access to Oracle tables and views. Reporting users do not have execute privilege on Oracle HRMS PL/SQL packages, and do not have direct access to the secured Oracle HRMS tables.

Restricted security profiles may restrict access to the following entities (the exact restrictions are determined by the definition of the security profiles):

- Organizations
- People

- Assignments
- Positions
- Vacancies
- Payrolls

All other entities are unrestricted; that is, restricted security profiles can access all records of tables, views and sequences associated with these entities.

Secure Tables and Views

The following Oracle HRMS tables are secured:

- HR_ALL_ORGANIZATION_UNITS
- PER_ALL_POSITIONS
- HR_ALL_POSITIONS_F
- PER_ALL_VACANCIES
- PER_ALL_PEOPLE_F
- PER_ALL_ASSIGNMENTS_F
- PAY_ALL_PAYROLLS_F

Some of these tables (namely PER_ALL_PEOPLE_F, PER_ALL_ASSIGNMENTS_F, HR_ALL_POSITIONS_F, and PAY_ALL_PAYROLLS_F) are datetracked. The following table details the views that are based on the secured tables listed above.

Table of Secure Tables and Views

Table or View	Description
HR_ORGANIZATION_UNITS	Secure view of Organization table
HR_ALL_ORGANIZATION_UNITS	Organization table
PER_ORGANIZATION_UNITS	Secure view of Organization view (HR Orgs only)
PER_ALL_ORGANIZATION_UNITS	Unsecured view of Organization view (HR Orgs only)
HR_ALL_POSITIONS	Unrestricted view of datetracked Positions table, effective at session date
HR_ALL_POSITIONS_F	Datetracked Positions table
HR_POSITIONS	Secure view of datetracked Positions table, effective at session date
HR_POSITIONS_F	Secure view of datetracked Positions table
HR_POSITIONS_X	Secure view of datetracked Positions table, effective at system date
PER_POSITIONS	Secure view of non-datetracked Positions table

Table or View	Description
PER_ALL_POSITIONS	Non-datetracked Positions table
PER_VACANCIES	Secure view of Vacancies table
PER_ALL_VACANCIES	Vacancies table
PER_ASSIGNMENTS	Secure view of Assignments table, effective at session date
PER_ASSIGNMENTS_F	Secure view of Assignments table
PER_ASSIGNMENTS_X	Secure view of Assignments table, effective at system date
PER_ALL_ASSIGNMENTS	Unrestricted view of Assignments table, effective at session date
PER_ALL_ASSIGNMENTS_F	Assignments table
PER_PEOPLE	Secure view of Person table, effective at session date
PER_PEOPLE_F	Secure view of Person table
PER_PEOPLE_X	Secure view of Person table, effective at system date
PER_ALL_PEOPLE	Unrestricted view of Person table, effective at session date
PER_ALL_PEOPLE_F	Person table
PAY_PAYROLLS	Secure view of Payrolls table, effective at session date
PAY_PAYROLLS_F	Secure view of Payrolls table
PAY_PAYROLLS_X	Secure view of Payrolls table, effective at system date
PAY_ALL_PAYROLLS	Unrestricted view of Payrolls table, effective at session date
PAY_ALL_PAYROLLS_F	Payrolls table

Accessing Oracle HRMS Data Through Restricted Security Profiles

When you connect to the APPS Oracle User you can access all Oracle HRMS database objects without having to perform any additional setup.

This is not the case for reporting users: two conditions must be met to enable reporting users to access Oracle HRMS tables and views:

- A public synonym must exist for each table and view. Public synonyms have the same name as the tables and views to which they point. They are created during installation of Oracle HRMS.
- The reporting user must have been granted permissions to access the tables and views by the SECGEN process. Reporting users are granted SELECT permission only. See below for more information about SECGEN.

How Secure Views Work

The information that is visible through a secure view depends on the definition of the security profile through which the view is being accessed.

If you have connected with a restricted security profile the information you can see is derived from denormalized lists of organizations, positions, people and payrolls.

The lists are used only when required. For example, the payroll list is empty for a security profile that can see all payrolls, and in the case of a security profile that can see all applicants but a restricted set of employees, the Person list contains employees but no applicants.

If the HR:Cross Business Groups profile option is 'N', the secure views return data only for the current business group.

If the HR:Cross Business Groups profile option is 'Y', the secure views return data for all business groups, subject to any further restrictions that apply by virtue of the current security profile.

Here is the text of the HR_ORGANIZATION_UNITS secure view:

```
SELECT HAO.ORGANIZATION_ID, HAOTL.NAME .....
FROM HR_ALL_ORGANIZATION_UNITS HAO,
  HR_ALL_ORGANIZATION_UNITS_TL HAOTL
WHERE DECODE(HR_SECURITY.VIEW_ALL, 'Y', 'TRUE',
  HR_SECURITY.SHOW_RECORD
  ('HR_ALL_ORGANIZATION_UNITS',HAOTL.ORGANIZATION_ID))='TRUE'
 AND DECODE(HR_GENERAL.GET_XBG_PROFILE,
  'Y', HAO.BUSINESS_GROUP_ID,
  HR_GENERAL.GET_BUSINESS_GROUP_ID_ = HAO.BUSINESS_GROUP_ID
 AND HAO.ORGANIZATION_ID = HAOTL.ORGANIZATION_ID
 AND HAOTL.LANGUAGE = USERENV('LANG')
```

Most HR security logic is encapsulated in a PL/SQL package, HR_SECURITY.

HR_SECURITY.VIEW_ALL returns the value of the VIEW_ALL_FLAG for the current security profile.

HR_SECURITY.SHOW_RECORD is called if the current security profile is a restricted security profile. It validates whether the row in question is visible through the current security profile.

HR_GENERAL.GET_XBG_PROFILE returns the value of the HR:Cross Business Group profile option.

HR_GENERAL.GET_BUSINESS_GROUP_ID returns the current business group ID. The HR: Business Group profile option supplies this ID.

Security Context

The HR security context contains values for all the attributes of the current security profiles. It is implemented using PL/SQL globals. The current security profile is derived as follows:

1. If you have logged onto Oracle Applications using the Oracle Applications sign-on screen, your security context is automatically set as part of the Oracle Applications sign-on procedure. Your current security_profile_id is derived from the responsibility and security group you select during sign-on.
2. If you have connected to an HR reporting user your current security_profile_id is taken from the PER_SECURITY_PROFILES table, where REPORTING_ORACLE_USERNAME matches the name of the Oracle User to which you have connected.
3. If it is not possible to derive a security_profile_id by either of the above two methods, the system looks for the default view-all security profile created for the business group. This gives you unrestricted access to the business group. If it cannot find this, the current security_profile_id is set to null, which prevents you from accessing any records.

So, if you connect directly to the APPS Oracle User through SQL*Plus, you will have unrestricted access to the HRMS tables. But if you connect to an HR reporting user, your access is restricted according to the definition of your security profile.

You can simulate the security context for an Oracle Applications session by calling FND_GLOBAL.APPS_INITIALIZE (user_id, resp_id, resp_appl_id, and security_group_id), passing the IDs of the user, responsibility, application, and security group for the sign-on session you want to simulate. The security_group_id is defaulted to zero (that is, the setup business group).

Note: FND_GLOBAL is not accessible from HR reporting users.

Security Lists

The security profile list tables contain denormalized lists of people, positions, organizations and payrolls.

Security profile lists are intersection tables between a security profile and secured tables, as follows:

Security List Table Name	Columns
PER_PERSON_LIST	SECURITY_PROFILE_ID, PERSON_ID
PER_POSITION_LIST	SECURITY_PROFILE_ID, POSITION_ID
PER_ORGANIZATION_LIST	SECURITY_PROFILE_ID, ORGANIZATION_ID
PAY_PAYROLL_LIST	SECURITY_PROFILE_ID, PAYROLL_ID
PER_PERSON_LIST_CHANGES	SECURITY_PROFILE_ID, PERSON_ID

These tables are periodically refreshed by the Security List Maintenance process (PERSLM). They are also written to when some relevant business processes are performed through Oracle HR, for example, employee hire or transfer.

If people are being secured via the supervisor hierarchy and organizations, positions and payrolls are not secured, the security list tables mentioned above are not used, and the Security List Maintenance process need not be run. The list of visible people is derived dynamically based on the current user.

If, however, supervisor security is being used in conjunction with organization and/or position and/or payroll security, you must run the Security List Maintenance process periodically to refresh the security list tables. The list of visible people is derived dynamically based on the current user, and is a subset of the people that are visible via the PER_PERSON_LIST table.

Security Processes

Three processes are used to implement Oracle HRMS security:

- Grant Secure Role Permission (ROLEGEN)
- Generate Secure User (SECGEN)
- Security List Maintenance (PERSLM)

ROLEGEN runs automatically as part of an installation or upgrade. If you are not setting up reporting users, you need not run SECGEN.

Refer to the topic on Security Processes, *Oracle HRMS Configuring, Reporting, and System Administration Guide* for details of how to submit SECGEN and PERSLM from the Submit Requests window. This section describes how the processes work.

ROLEGEN: Grant Secure Role Permission Process

A role is a set of permissions that can be granted to Oracle users or to other roles. Roles are granted to users by the SECGEN process (see below).

The ROLEGEN process must run before you run SECGEN. ROLEGEN dynamically grants select permissions on Oracle HRMS tables and views to the HR_REPORTING_USER role. This role must exist before ROLEGEN runs.

The HR_REPORTING_USER role is created during the install of Oracle HRMS. ROLEGEN is run during the install of Oracle HRMS.

> **Note:** As ROLEGEN runs as part of the installation and upgrade processes, you do not need to run ROLEGEN manually.

ROLEGEN performs the following actions:

- Creates public synonyms for HRMS tables and views, excluding unsecured tables (%_ALL_%)
- Revokes all existing permissions from HR_REPORTING_USER roles
- Grants SELECT permissions to HR_REPORTING_USER role for HRMS tables and views

SECGEN - Generate Secure User Process

You run SECGEN for a specified security profile. It grants the HR_REPORTING_USER role to the Oracle User associated with the security profile.

SECGEN must be run after ROLEGEN. However, once SECGEN has been run for a particular security profile, you need not rerun it even if ROLEGEN is run again.

SECGEN is a PRO*C process with embedded SQL statements. You initiate it from the Submit Requests window.

PERSLM - Security List Maintenance Process

You should run PERSLM periodically (for example, nightly) to refresh the security lists upon which the secure views are built.

Important: This process has the capability to run multi-threaded, allowing it to take advantage of the capabilities of your hardware. To take full advantage of this feature, you need to perform a number of additional setup steps. Details of these are available in the *Oracle HRMS & Benefits Tuning & System Health Check - Release 11i*, available on Metalink (Note 226987.1).

PERSLM is a PL/SQL procedure that you submit from the Submit Requests window. It builds the required security lists based on the restrictions defined for the security profiles being processed.

For each security profile within the scope specified when the process is submitted, PERSLM performs the following steps:

1. If the View All flag is Y, the process ends leaving all security lists empty for the specified security profile.
2. Builds a payroll list.

   If the View All Payrolls flag is Y, the process leaves the payroll list empty. If the View All Payrolls flag is N, the process checks the Include Payroll flag. If this flag is Y, the process makes a list of all payrolls in the pay_security_payrolls list. If the flag is N, the process makes a list of all payrolls except those in the pay_security_payrolls list. The pay_security_payrolls list is populated when you enter payrolls on the Define Security Profile screen.
3. Builds an organization list.

   If the View All Organizations flag is Y, the process leaves the organization list empty. If this flag is N, the process builds a list of all organizations below the top one you specified for the organization hierarchy you chose on the Define Security Profile screen. The process uses the version of the hierarchy that is effective on the date passed to PERSLM. If the Include Top Organization flag is Y, the top organization you specified is included in the list. Any organizations specifically listed in the Define Security Profile window are included or excluded as specified. If the Exclude Business Group flag is N, the business group is included in the list to allow newly entered employees and applicants to be visible before they are assigned to an organization.
4. Builds a position list.

   If the View All Positions flag is Y, the process builds a list of all positions within the organizations on the organization list. If this flag is N, the process builds a list of all positions below the top one you specified for the position hierarchy you chose on the Define Security Profile screen. The process uses the version of the hierarchy that is effective on the date passed to PERSLM. If the Include Top Position flag is Y, the top position you specified is included in the list. The list of positions is built up for all organizations on the organization list, or for all organizations if the View All Organizations flag is Y.
5. Builds a person list.

   The process creates person list information for all people within the specified scope, including terminated employees, applicants, and contingent workers. The process uses the assignment data effective on the date passed to PERSLM to determine eligibility for current employees, applicants, and contingent workers. For terminated employees, applicants, and contingent workers, the process identifies the

maximum effective end date for any assignment defined for the person, and uses the assignment data effective on this date to determine eligibility.

The system processes future-dated hires, placements, or applicants using the date when the assignment becomes effective.

> **Note:** If the process is running for a terminated person with a future-dated hire or placement, the system uses the future-dated assignment information to determine eligibility. If a terminated employee or contingent worker is also a current or future applicant, the system determines eligibility using both the terminated assignment data and the current or future assignment data.

The mechanism used to generate the person list depends on the value selected for the "Generate For" parameter selected at the time PERSLM was submitted:

- **One Named Security Profile**

  The process generates the list of people visible to the security profile identified by the Security Profile parameter. This process runs single-threaded.

  For the named profile, PERSLM determines what security restrictions have been entered and dynamically builds a SQL statement to identify all the people who match the restriction criteria as follows:

  - If the View Employee field is Restricted then process all Employee assignments. If the View Employee field is either None or All then no data for employees is written to the person list table and access is controlled within the secure view.
  - If the View Contingent Worker field is Restricted then process all Contingent Worker assignments. If the View Contingent Worker field is either None or All then no data for contingent workers is written to the person list table and access is controlled within the secure view.
  - If the View Applicant field is Restricted then process all Applicant assignments provided that the Applicant is not also an Employee or Contingent Worker, in which case access is granted based on their Employee or Contingent Worker assignment. If the View Applicant field is either None or All then no data for applicants is written to the person list table and access is controlled within the secure view.
  - If the Organization Security Type is "Secure by Organization Hierarchy and/or Organization List" then PERSLM restricts access to Employees, Applicants and Contingent Workers with a current assignment to organizations in the organization list.
  - If the View All Positions flag is N, then PERSLM restricts access to Employees, Applicants and Contingent Workers with a current assignment to positions in the position list. PERSLM includes people who are not currently assigned to a position.
  - If the View All Payrolls flag is N, then PERSLM restricts access to Employees and Applicants with a current assignment to payrolls in the payroll list.

- If the Custom Restriction flag is Y, then PERSLM restricts access to Employees, Applicants and Contingent Workers using the conditions defined within the custom restriction.

  If a security profile contains multiple restrictions then data is only written to the person list table for people who satisfy all the restrictions defined.

- **All Security Profiles, All Global Security Profiles, All Security Profiles in One Named Business Group**

  The process generates the list of people visible to the security profiles within the scope of the option selected. For example, if the "All Global Security Profiles" option is selected, the person list information is regenerated for all global security profiles but for *no* business group-specific profiles. This process is implemented using the Oracle Payroll Archiver process, allowing it to run multi-threaded if your system has been configured correctly. See the *Oracle HRMS & Benefits Tuning & System Health Check - Release 11i*, available on Metalink (Note 226987.1), for information on configuring your system to run multi-threaded processes.

  When generating security lists for one of these options, all people in the system are checked and processed for eligibility. (If you elect to generate list information for security profiles in a single named business group, then only people defined within that business group are processed. For the other two options all people within the database are processed.)

  The individual assignments for people requiring processing are examined to determine which security profiles can access them. Based on the Organization, Position, and Payroll data present on the assignment the set of security profiles that can see the assignment is determined using the restrictions defined on each security profile within scope and Organization, Position and Payroll list information previously generated for those security profiles. In addition, any custom restriction for each of the profiles is evaluated to ensure that the assignment and person comply with any criteria entered.

6. Adds person list changes.

   PERSLM adds a person to the person list if an entry exists in the PER_PERSON_LIST_CHANGES table, there is no current period of service, and there is no current application for the person. It only adds people if they are not already in the list.

7. Contacts (Persons with system person type 'OTHER')

   Some security profiles restrict a user's access to contacts. If this is the case, the process generates access to the contacts who are related to the current and terminated employees, applicants, and contingent workers within the user's security profile. The process also allows access to contacts who are unrelated to any person within the system. The process uses the PER_CONTACT_RELATIONSHIPS table to determine a contact's relationships.

Securing Custom Tables

If you have created your own custom tables, perform the following steps to make them accessible to reporting users:

1. Create table.

Select a table name that does not conflict with any tables or views that might exist in Oracle Applications.

Do not use two or three character prefixes such as HR, PER, PAY, FF, DT, SSP, GHR, BEN, OTA, HXT, EDW, HRI, HXC, PQH, PQP or IRC.

2. Grant select access on the table to HR_REPORTING_USER role, from the user that owns the custom table.

```
GRANT SELECT ON custom_table TO hr_reporting_user;
```

You must repeat this step every time you perform an installation or upgrade. However, you do not need to rerun SECGEN as existing reporting users that have already been granted access to the HR_REPORTING_USER role will automatically receive any new permissions added to the role.

3. Create a synonym to the table.

If you use public synonyms, remember that the Oracle user from which you create the public synonym must have CREATE PUBLIC SYNONYM system privilege.

```
CREATE PUBLIC SYNONYM custom_table
   FOR base_table_account.custom_table;
```

APIs

APIs in Oracle HRMS

An Application Programmatic Interface (API) is a logical grouping of external process routines. The Oracle HRMS strategy delivers a set of PL/SQL packaged procedures and functions that together provide an open interface to the database. For convenience we have called each of these packaged procedures an API.

This document provides all the technical information you need to be able to use these APIs and covers the following topics:

- API Overview, page 2-218

  Describes how you can use the Oracle HRMS APIs and the advantages of this approach.

- Understanding the Object Version Number (OVN), page 2-220

  Explains the role of the object version number. The APIs use it to check whether a row has been updated by another user, to prevent overwriting their changes.

- API Parameters, page 2-222

  Explains where to find information about the parameters used in each API; parameter naming conventions; the importance of naming parameters in the API call instead of relying on parameter list order; and how to use default values to avoid specifying all parameters. Also explains the operation of certain control parameters, such as those controlling DateTrack operations.

- API Features, page 2-236

  Explains that commits are handled by the calling program, not the APIs, and the advantages of this approach. Also explains how to avoid deadlocks when calling more than one API in the same commit unit.

- Flexfields with APIs, page 2-237

  Describes how the APIs validate key flexfield and descriptive flexfield values.

- Multilingual Support, page 2-238

  Explains how to use the Multilingual Support APIs.

- Alternative APIs, page 2-239

  Explains that we provide legislation-specific APIs for some business processes, such as Create Address.

- API Errors and Warnings, page 2-240

  Explains how the APIs raise errors and warnings, and how the calling code can handle them. A message table is provided for handling errors in batch processes.

- Example PL/SQL Batch Program, page 2-242

  Shows how to load a batch of person address data and how to handle validation errors.

- WHO Columns and Oracle Alert, page 2-244

  Explains how to populate the WHO columns (which record the Applications user who caused the database row to be created or updated) when you use the APIs.

- API User Hooks, page 2-245

  A user hook is a location where you can add processing logic or validation to an API. There are hooks in the APIs for adding validation associated with a particular business process. There are also hooks in table-level modules for validation on specific data items. This section explains where user hooks are available and how to implement them. It also explains their advantages over database triggers.

- Using APIs as Building Blocks, page 2-264

  Explains how you can write your own APIs that call one or more of the supplied APIs.

- Handling Object Version Numbers in Oracle Forms, page 2-265

  Explains how to implement additional Forms logic to manage the object version number if you write your own forms that call the APIs.

API Overview

Fundamental to the design of all APIs in Oracle HRMS is that they should provide an insulating layer between the user and the data-model that would simplify all data-manipulation tasks and would protect customer extensions on upgrade. They are parameterized and executable PL/SQL packages that provide full data validation and manipulation.

The API layer enables us to capture and execute business rules within the database - not just in the user interface layer. This layer supports the use of alternative interfaces to HRMS, such as web pages or spreadsheets, and guarantees all transactions comply with the business rules that have been implemented in the system. It also simplifies integration of Oracle HRMS with other systems or processes and provides supports for the initial loading

Alternative User Interfaces

The supported APIs can be used as an alternative data entry point into Oracle HRMS. Instead of manually typing in new information or altering existing data using the online forms, you can implement other programs to perform similar operations.

These other programs do not modify data directly in the database. They call the APIs which:

1. Ensure it is appropriate to allow that particular business operation
2. Validate the data passed to the API
3. Insert/update/delete data in the HR schema

APIs are implemented on the server-side and can be used in many ways. For example:

- Customers who want to upload data from an existing system. Instead of employing temporary data entry clerks to type in data, a program could be written to extract data from the existing system and then transfer the data into Oracle HRMS by calling the APIs.
- Customers who purchase a number of applications from different vendors to build a complete solution. In an integrated environment a change in one application may require changes to data in another. Instead of users having to remember to go into each application repeating the change, the update to the HRMS applications could be applied electronically. Modifications can be made in batches or immediately on an individual basis.
- Customers who want to build a custom version of the standard forms supplied with Oracle HRMS. An alternative version of one or more forms could be implemented using the APIs to manage all database transactions.
- Customers who want to develop web-based interfaces to allow occasional users to access and maintain HR information without the cost of deploying or supporting standard Oracle HRMS forms. This is the basis of most Self-Service functions that allow employees to query and update their own information, such as change of name, address, marital status. This also applies to managers who want to query or maintain details for the employees they manage.
- Managers who are more familiar with spreadsheet applications may want to export and manipulate data without even being connected to the database and then upload modifications to the HRMS database when reconnected.

In all these examples, the programs would not need to modify data directly in the Oracle HRMS database tables. The specific programs would call one or more APIs and these would ensure that invalid data is not written to the Oracle HRMS database and that existing data is not corrupted.

Advantages of Using APIs

Why use APIs instead of directly modifying data in the database tables?

Oracle does not support any direct manipulation of the data in any application using PL/SQL. APIs provide you with many advantages:

- APIs enable you to maintain HR and Payroll information without using Oracle forms.
- APIs insulate you from the need to fully understand every feature of the database structure. They manage all the inter-table relationships and updates.
- APIs are guaranteed to maintain the integrity of the database. When necessary, database row level locks are used to ensure consistency between different

tables. Invalid data cannot be entered into the system and existing data is protected from incorrect alterations.

- APIs are guaranteed to apply all parts of a business process to the database. When an API is called, either the whole transaction is successful and all the individual database changes are applied, or the complete transaction fails and the database is left in the starting valid state, as if the API had not been called.
- APIs do not make these changes permanent by issuing a commit. It is the responsibility of the calling program to do this. This provides flexibility between individual record and batch processing. It also ensures that the standard commit processing carried out by client programs such as Forms is not affected.
- APIs help to protect any customer-specific logic from database structure changes on upgrade. While we cannot guarantee that any API will not change to support improvements or extensions of functionality, we are committed to minimize the number of changes and to provide appropriate notification and documentation if such changes occur.

  Note: Writing programs to call APIs in Oracle HRMS requires knowledge of PL/SQL version 2. The rest of this essay explains how to call the APIs and assumes the reader has knowledge of programming in PL/SQL.

Understanding the Object Version Number (OVN)

Nearly every row in every database table is assigned an object_version_number. When a new row is inserted, the API usually sets the object version number to 1. Whenever that row is updated in the database, the object version number is incremented. The row keeps that object version number until it is next updated or deleted. The number is not decremented or reset to a previous value.

> **Note:** The object version number is not unique and does not replace the primary key. There can be many rows in the same table with the same version number. The object version number indicates the version of a specific primary key row.

Whenever a database row is transferred (queried) to a client, the existing object version number is always transferred with the other attributes. If the object is modified by the client and saved back to the server, then the current server object version number is compared with the value passed from the client.

- If the two object version number values are the same, then the row on the server is in the same state as when the attributes were transferred to the client. As no other changes have occurred, the current change request can continue and the object version number is incremented.
- If the two values are different, then another user has already changed and committed the row on the server. The current change request is not allowed to continue because the modifications the other user made may be overwritten and lost. (Database locks are used to prevent another user from overwriting uncommitted changes.)

The object version number provides similar validation comparison to the online system. Forms interactively compare all the field values and displays the "Record has been modified by another user" error message if any differences are found. Object version numbers allow transactions to occur across longer periods of time without holding long term database locks. For example, the client application may save the row

locally, disconnect from the server and reconnect at a later date to save the change to the database. Additionally, you do not need to check all the values on the client and the server.

Example

Consider creating a new address for a Person. The *create_person_address* API automatically sets the object_version_number to 1 on the new database row. Then, two separate users query this address at the same time. User A and user B will both see the same address details with the current object_version_number equal to 1.

User A updates the Town field to a different value and calls the *update_person_address* API passing the current object_version_number equal to 1. As this object_version_number is the same as the value on the database row the update is allowed and the object_version_number is incremented to 2. The new object_version_number is returned to user A and the row is committed in the database.

User B, who has details of the original row, notices that first line of the address is incorrect. User B calls the update_person_address API, passing the new first line and what he thinks is the current object_version_number (1). The API compares this value with the current value on the database row (2). As there is a difference the update is not allowed to continue and an error is returned to user B.

To correct the problem, user B then re-queries this address, seeing the new town and obtains the object_version_number 2. The first line of the address is updated and the update_person_address API is called again. As the object_version_number is the same as the value on the database row the update is allowed to continue.

Therefore both updates have been applied without overwriting the first change.

Understanding the API Control Parameter p_object_version_number

Most published APIs have the p_object_version_number control parameter.

- For create style APIs, this parameter is defined as an OUT and will always be initialized.
- For update style APIs, the parameter is defined as an IN OUT and is mandatory.

The API ensures that the object version number(s) match the current value(s) in the database. If the values do not match, the application error HR_7155_OBJECT_LOCKED is generated. At the end of the API call, if there are no errors the new object version number is passed out.

For delete style APIs when the object is not DateTracked, it is a mandatory IN parameter. For delete style APIs when the object is DateTracked, it is a mandatory IN OUT parameter.

The API ensures that the object version number(s) match the current value(s) in the database. When the values do not match, the application error HR_7155_OBJECT_LOCKED is raised. When there are no errors for DateTracked objects that still list, the new object version number is passed out.

See:

Understanding the p_datetrack_update_mode control parameter, page 2-232

Understanding the p_datetrack_delete_mode control parameter, page 2-234

Handling Object Version Numbers in Oracle Forms, page 2-265

Detecting and Handling Object Conflicts

When the row being processed does not have the correct object version number, the application error HR_7155_OBJECT_LOCKED is raised. This error indicates that a particular row has been successfully changed and committed since you selected the information. To ensure that the other changes are not overwritten by mistake, re-select the information, reapply your changes, and re-submit to the API.

API Parameters

This section describes parameter usage in Oracle HRMS.

Locating Parameter Information

You can find the parameters for each API in one of two ways, either by looking at the documentation in the package header creation scripts or by using SQL*Plus.

Package Header Creation Scripts

For a description of each API, including a list of IN parameters and OUT parameters, refer to the documentation in the package header creation scripts.

For core product APIs, which are included in the first version of a main release, scripts are located in the product TOP admin/sql directories. Refer to filenames such as *api.pkh. Localization-specific APIs follow a *LLi.pkh naming standard, where LL is the two letter localization code.

For example, details for all the APIs in the hr_employee_api package can be found in the $PER_TOP/admin/sql/peempapi.pkh file.

New APIs that were not included in the first version of a main release, or are localization-specific, may be provided in different operating system directories.

Oracle only supports the APIs listed in the following documentation:

- Publicly Callable Business Process APIs in Oracle HRMS, *Oracle HRMS Configuring, Reporting, and System Administration Guide*
- The What's New in Oracle HRMS topic in the help system. This will list any new APIs introduced after the first version of a main Release.

These lists are a reduced set of the server side code that matches all of the following three criteria:

- The database package name ends with "_API".
- The package header creation script filename conforms to the *api.pkh or *LLi.pkh naming standard, where LL is a two letter localization code.
- The individual API documentation has an "Access" section with a value of "Public".

Many other packages include procedures and functions, which may be called from the API code itself. Direct calls to any other routines are not supported, unless explicitly specified, because API validation and logic steps will be bypassed. This may corrupt the data held within the Oracle HRMS application suite.

Using SQL*Plus to List Parameters

If you simply want a list of PL/SQL parameters, use SQL*Plus. At the SQL*Plus prompt, use the describe command followed by the database package name, period, and the name of the API. For example, to list the parameters for the create_grade_rate_value API, enter the following at the SQL> prompt:

```
describe hr_grade_api.create_grade_rate_value
```

Parameter Names

Each API has a number of parameters that may or may not be specified. Most parameters map onto a database column in the HR schema. There are some control parameters that affect the processing logic that are not explicitly held on the database.

Every parameter name starts with *p_*. If the parameter maps onto a database column, the remaining part of the name is usually the same as the column name. Some names may be truncated due to the 30 character length limit. The parameter names have been made slightly different to the actual column name, using a *p_* prefix, to avoid coding conflicts when a parameter and the corresponding database column name are both referenced in the same section of code.

When a naming conflict occurs between parameters, a three-letter short code (identifying the database entity) is included in the parameter name. Sometimes there is no physical name conflict, but the three-letter short code is used to avoid any confusion over the entity with which the parameter is associated.

For example, create_employee contains examples of both these cases. Part of the logic to create a new employee is to insert a person record and insert an assignment record. Both these entities have an object_version_number. The APIs returns both object_version_number values using two OUT parameters. Both parameters cannot be called p_object_version_number, so p_per_object_version_number holds the value for the person record and p_asg_object_version_number holds the value for the assignment record.

Both these entities can have text comments associated with them. When any comments are passed into the create_employee API, they are only noted against the person record. The assignment record comments are left blank.

To avoid any confusion over where the comments have allocated in the database, the API returns the id using the p_per_comment_id parameter.

Parameter Named Notation

When calling the APIs, it is strongly recommended that you use "Named Notation," instead of "Positional Notation." Thus, you should list each parameter name in the call instead of relying on the parameter list order.

Using "Named Notation" helps protect your code from parameter interface changes. With future releases, it eases code maintenance when parameters are added or removed from the API.

For example, consider the following procedure declaration:

```
procedure change_age
  (p_name    in     varchar2
  ,p_age     in     number
  ;
```

Calling by 'Named Notation':

```
begin
  change_age
    (p_name => 'Bloggs'
    ,p_age  => 21
    );
end;
```

Calling by 'Positional Notation':

```
begin
  change_age
    ('Bloggs'
    ,21
    );
end;
```

Using Default Parameter Values

When calling an API it may not be necessary to specify every parameter. Where a PL/SQL default value has been specified it is optional to specify a value.

If you want to call the APIs from your own forms, then all parameters in the API call must be specified. You cannot make use of the PL/SQL declared default values because the way Forms calls server-side PL/SQL does not support this.

Default Parameters with Create Style APIs

For APIs that create new data in the HR schema, optional parameters are usually identified with a default value of null. After validation has been completed, the corresponding database columns will be set to null. When calling the API, you must specify all the parameters that do not have a default value defined.

However, some APIs contain logic to derive some attribute values. When you pass in the PL/SQL default value the API determines a specific value to set on the database column. You can still override this API logic by passing in your own value instead of passing in a null value or not specifying the parameter in the call.

Take care with IN OUT parameters, because you must always include them in the calling parameter list. As the API can pass values out, you must use a variable to pass values into this type of parameter.

These variables must be set with your values before calling the API. If you do not want to specify a value for an IN OUT parameter, use a variable to pass a null value to the parameter.

> **Important:** Check the comments in each API package header creation script for details of when each IN OUT parameter can and cannot be set with a null value.

The create_employee API contains examples of all these different types of parameter.

```
procedure create_employee
  (
  ...
  ,p_sex                          in     varchar2
  ,p_person_type_id               in     number

                                    default null
  ...
  ,p_email_address                in     varchar2
```

```
                                    default null
,p_employee_number                in out varchar2
...
,p_person_id                        out number
,p_assignment_id                    out number
,p_per_object_version_number        out number
,p_asg_object_version_number        out number
,p_per_effective_start_date         out date
,p_per_effective_end_date           out date
,p_full_name                        out varchar2
,p_per_comment_id                   out number
,p_assignment_sequence              out number
,p_assignment_number                out varchar2
,p_name_combination_warning         out boolean
,p_assign_payroll_warning           out boolean
,p_orig_hire_warning                out boolean
);
```

Because no PL/SQL default value has been defined, the p_sex parameter must be set. The p_person_type_id parameter can be passed in with the ID of an Employee person type. If you do not provide a value, or explicitly pass in a null value, the API sets the database column to the ID of the active default employee system person type for the business group. The comments in each API package header creation script provide more information.

The p_email_address parameter does not have to be passed in. If you do not specify this parameter in your call, a null value is placed on the corresponding database column. (This is similar to the user of a form leaving a displayed field blank.)

The p_employee_number parameter must be specified in each call. When you do not want to set the employee number, the variable used in the calling logic must be set to null. (For the p_employee_number parameter, you must specify a value for the business group when the method of employee number generation is set to manual. Values are only passed out when the generation method is automatic or national identifier.)

Example 1

An example call to the create_employee API where the business group method of employee number generation is manual, the default employee person type is required and the e-mail attributes do not need to be set.

```
declare
  l_emp_num                    varchar2(30);
  l_person_id                  number;
  l_assignment_id              number;
  l_per_object_version_number  number;
  l_asg_object_version_number  number;
  l_per_effective_start_date   date;
  l_per_effective_end_date     date;
  l_full_name                  varchar2(240);
  l_per_comment_id             number;
  l_assignment_sequence        number;
  l_assignment_number          varchar2(30);
  l_name_combination_warning   boolean;
  l_assign_payroll_warning     boolean;
  l_orig_hire_warning          boolean;
begin
  --
  -- Set variable with the employee number value,
  -- which is going to be passed into the API.
  --
  l_emp_num := 4532;
  --
  -- Put the new employee details in the database
  -- by calling the create_employee API
  --
  hr_employee.create_employee
    (p_hire_date                 =>

                to_date('06-06-1996','DD-MM-YYYY')
    ,p_business_group_id         => 23
    ,p_last_name                 => 'Bloggs'
    ,p_sex                       => 'M'
    ,p_employee_number           => l_emp_num
    ,p_person_id                 => l_person_id
    ,p_assignment_id             => l_assignment_id
    ,p_per_object_version_number => l_per_object_version_number
    ,p_asg_object_version_number => l_asg_object_version_number
    ,p_per_effective_start_date  => l_per_effective_start_date
    ,p_per_effective_end_date    => l_per_effective_end_date
    ,p_full_name                 => l_full_name
    ,p_per_comment_id            => l_per_comment_id
    ,p_assignment_sequence       => l_assignment_sequence
    ,p_assignment_number         => l_assignment_number
    ,p_name_combination_warning  => l_name_combination_warning
    ,p_assign_payroll_warning    => l_assign_payroll_warning
    ,p_orig_hire_warning        => l_orig_hire_warning
    );
end;
```

Note: The database column for employee_number is defined as varchar2 to allow for when the business group method of employee_number generation is set to National Identifier.

Example 2

An example call to the create_employee API where the business group method of employee number generation is Automatic, a non-default employee person type must be used and the email attribute details must be held.

```
declare
  l_emp_num                   varchar2(30);
  l_person_id                 number;
  l_assignment_id             number;
  l_per_object_version_number number;
  l_asg_object_version_number number;
  l_per_effective_start_date  date;
  l_per_effective_end_date    date;
  l_full_name                 varchar2(240);
  l_per_comment_id            number;
  l_assignment_sequence       number;
  l_assignment_number         varchar2(30);
  l_name_combination_warning  boolean;
  l_assign_payroll_warning    boolean;
  l_orig_hire_warning         boolean;
begin
  --
  -- Clear the employee number variable
  --
  l_emp_num := null;
  --
  -- Put the new employee details in the database
  -- by calling the create_employee API
  --
  hr_employee.create_employee
    (p_hire_date                =>

                  to_date('06-06-1996','DD-MM-YYYY')
    ,p_business_group_id        => 23
    ,p_last_name                => 'Bloggs'
    ,p_sex                      => 'M'
    ,p_person_type_id           => 56
    ,p_email_address            => 'bloggsf@uk.uiq.com'
    ,p_employee_number          => l_emp_num
    ,p_person_id                => l_person_id
    ,p_assignment_id            => l_assignment_id
    ,p_per_object_version_number => l_per_object_version_number
    ,p_asg_object_version_number => l_asg_object_version_number
    ,p_per_effective_start_date  => l_per_effective_start_date
    ,p_per_effective_end_date    => l_per_effective_end_date
    ,p_full_name                => l_full_name
    ,p_per_comment_id           => l_per_comment_id
    ,p_assignment_sequence      => l_assignment_sequence
    ,p_assignment_number        => l_assignment_number
    ,p_name_combination_warning  => l_name_combination_warning
    ,p_assign_payroll_warning    => l_assign_payroll_warning
    ,p_orig_hire_warning        => l_orig_hire_warning
    );
    --
    -- The l_emp_num variable is now set with the
    -- employee_number allocated by the HR system.
    --
```

```
end;
```

Default Parameters with Update Style APIs

With update style APIs the primary key and object version number parameters are usually mandatory. In most cases it is not necessary provide all the parameter values. You only need to specify any control parameters and the attributes you are actually altering. It is not necessary (but it is possible) to pass in the existing values of attributes that are not being modified. Optional parameters have one of the following PL/SQL default values, depending on the datatype as shown in the following table:

Data Type	Default value
varchar2	hr_api.g_varchar2
number	hr_api.g_number
date	hr_api.g_date

These hr_api.g_ default values are constant definitions, set to special values. They are not hard coded text strings. If you need to specify these values, use the constant name, not the value. The actual values are subject to change.

Care must be taken with IN OUT parameters, because they must always be included in the calling parameter list. As the API is capable of passing values out, you must use a variable to pass values into this type of parameter. These variables must be set with your values before calling the API. If you do not want to explicitly modify that attribute you should set the variable to the hr_api.g_... value for that datatype. The update_emp_asg_criteria API contains examples of these different types of parameters.

```
procedure update_emp_asg_criteria

(...

,p_assignment_id                in     number

,p_object_version_number        in out number

...

,p_position_id                  in     number

                            default hr_api.g_number

...

,p_special_ceiling_step_id      in out number

...

,p_employment_category          in     varchar2

                            default hr_api.g_varchar2

,p_effective_start_date            out date
```

```
,p_effective_end_date              out date
,p_people_group_id                 out number
,p_group_name                      out varchar2
,p_org_now_no_manager_warning      out boolean
,p_other_manager_warning           out boolean
,p_spp_delete_warning              out boolean
,p_entries_changed_warning         out varchar2
,p_tax_district_changed_warning    out boolean
);
```

> **Note:** Only the parameters that are of particular interest have been shown. Ellipses (...) indicate where irrelevant parameters to this example have been omitted.

The p_assignment_id and p_object_version_number parameters are mandatory and must be specified in every call. The p_position_id parameter is optional. If you do not want to alter the existing value, then exclude the parameter from your calling logic or pass in the hr_api.g_varchar2 constant or pass in the existing value.

The p_special_ceiling_step_id parameter is IN OUT. With certain cases the API sets this attribute to null on the database and the latest value is passed out of the API. If you do not want to alter this attribute, set the calling logic variable to hr_api.g_number.

Example

The following is an example call to the update_emp_asg_criteria API, with which you do not want to alter the position_id and special_ceiling_step_id attributes, but you do want to modify the employment_category value.

```
declare
  l_assignment_id                 number;
  l_object_version_number         number;
  l_special_ceiling_step_id       number;
  ...
begin
  l_assignment_id           := 23121;
  l_object_version_number   := 4;
  l_special_ceiling_step_id := hr_api.g_number;
  hr_assignment_api.update_emp_asg_criteria
    (...
    ,p_assignment_id                => l_assignment_id
    ,p_object_version_number        => l_object_version_number
    ...
    ,p_special_ceiling_step_id      => l_special_ceiling_step_id
    ...
    ,p_employment_category          => 'FT'
    ...
    );
  --
  -- As p_special_ceiling_step_id is an IN OUT parameter the
  -- l_special_ceiling_step_id variable is now set to the same
   -- value as on the database. i.e. The existing value before
    -- the API was called or the value which was derived by the
     -- API. The variable will not be set to hr_api.g_number.
  --
end;
```

Default Parameters with Delete Style APIs

Most delete style APIs do not have default values for any attribute parameters. In rare cases parameters with default values work in a similar way to those of update style APIs.

Parameters with NOCOPY

Starting from Applications Release 11.5.9, many PL/SQL APIs have been enhanced to make use of the PL/SQL pass by reference feature. The NOCOPY compiler directive is defined with OUT and IN OUT parameters. This improves run-time performance and reduces memory usage.

For the majority of calling programs, when an API with or without NOCOPY is called with valid data values, there will be no noticeable difference in behavior. However, there are some subtle differences, which calling programs need to take into consideration.

Use Different Variables

When calling a PL/SQL API, ensure that different variables are used to capture values returned from the OUT and IN OUT parameters. Using the same variable with multiple OUT parameters, or an IN only parameter and also an OUT parameter, can lead to the API behaving incorrectly. In some circumstances this can cause data corruption. Even if you are not interested in knowing or processing the returned value you must use different variables.

Error Processing

At the start of any procedure call, PL/SQL sets the variables from the calling program used with OUT only NOCOPY parameters to null. If a validation issue or other problem is detected by the API, an error is raised as a PL/SQL exception. Any OUT parameter values that the API has calculated before the error is detected are cleared with null. This ensures that the variables in the calling program used with the OUT parameters do not contain any misleading values.

When NOCOPY has not been specified, the variables contain the values that existed immediately before the procedure call began. This difference in behavior is noticed only by calling programs that contain an exception handler and that attempt to read the variable expecting to see the value that the variable contained before the call.

If the calling program needs to know the variable value that existed before the API was called, you must declare and populate a separate variable.

There is no change to the behavior of IN only and IN OUT parameters, regardless of the existence of the NOCOPY compiler directive. After an error occurs, the variable used with the IN or IN OUT parameter holds the value that existed immediately before the procedure call began.

Understanding the p_validate Control Parameter

Every published API includes the p_validate control parameter. When this parameter is set to FALSE (the default value), the procedure executes all validation for that business function. If the operation is valid, the database rows/values are inserted or updated or deleted. Any non warning OUT parameters, warning OUT parameters and IN OUT parameters are all set with specific values.

When the p_validate parameter is set to TRUE, the API only checks that the operation is valid. It does so by issuing a savepoint at the start of the procedure and rolling back to that savepoint at the end. You do not have access to these internal savepoints. If the procedure is successful, without raising any validation errors, then non-warning OUT parameters are set to null, warning OUT parameters are set to a specific value, and IN OUT parameters are reset to their IN values.

In some cases you may want to write your PL/SQL routines using the public API procedures as building blocks. This enables you to write routines specific to your business needs. For example, say that you have a business requirement to apply a DateTracked update to a row and then apply a DateTrack delete to the same row in the future. You could write an "update_and_future_del" procedure that calls two of the standard APIs.

When calling each standard API, p_validate must be set to false. If true is used the update procedure call is rolled back. So when the delete procedure is called, it is working on the non-updated version of the row. However when p_validate is set to false, the update is not rolled back. Thus, the delete call operates as if the user really wanted to apply the whole transaction.

If you want to be able to check that the update and delete operation is valid, you must issue your own savepoint and rollback commands. As the APIs do not issue any commits, there is no danger of part of the work being left in the database. It is the responsibility of the calling code to issue commits. The following simulates some of the p_validate true behavior.

Example

[Dummy text - remove in Epic]

```
savepoint s1;
update_api_prc(.........);
delete_api_prc(..........);
rollback to s1;
```

You should not use our API procedure names for the savepoint names. An unexpected result may occur if you do not use different names.

Understanding the p_effective_date Control Parameter

Most APIs that insert/update/delete data for at least one DateTrack entity have a p_effective_date control parameter. This mandatory parameter defines the date you want an operation to be applied from. The PL/SQL datatype of this parameter is date.

As the smallest unit of time in DateTrack is one day, the time portion of the p_effective_date parameter is not used. This means that the change always comes into effect just after midnight.

Some APIs have a more specific date for processing. For example, the create_employee API does not have a p_effective_date parameter. The p_hire_date parameter is used as the first day the person details come into effect.

Example 1

This example creates a new grade rate that starts from today.

```
hr_grade_api.create_grade_rate_value

(...

,p_effective_date => trunc(sysdate)

 ...);
```

Example 2

This example creates a new employee who joins the company at the start of March 1997.

```
hr_employee_api.create_employee

(...

,p_hire_date => to_date('01-03-1997','DD-MM-YYYY')

...);
```

Some APIs that do not modify data in DateTrack entities still have a p_effective_date parameter. The date value is not used to determine when the changes take effect. It is used to validate Lookup values. Each Lookups value can be specified with a valid date range. The start date indicates when the value can first be used. The end date shows the last date the value can be used on new records and set when updating records. Existing records, which are not changed, can continue to use the Lookup after the end date.

Understanding the p_datetrack_update_mode Control Parameter

Most APIs that update data for at least one DateTrack entity have a p_datetrack_update_mode control parameter. It enables you to define the type of DateTrack change to be made. This mandatory parameter must be set to one of the values in the following table:

p_datetrack_update_mode Value	Description
UPDATE	Keep history of existing information
CORRECTION	Correct existing information
UPDATE_OVERRIDE	Replace all scheduled changes
UPDATE_CHANGE_INSERT	Insert this change before next scheduled change

It may not be possible to use every mode in every case. For example, if there are no existing future changes for the record you are changing, the DateTrack modes UPDATE_OVERRIDE and UPDATE_CHANGE_INSERT cannot be used.

Some APIs that update DateTrack entities do not have a p_datetrack_update_mode parameter. These APIs automatically perform the DateTrack operations for that business operation.

Each dated instance for the same primary key has a different object_version_number. When calling the API the p_object_version_number parameter should be set to the value that applies as of the date for the operation (that is, p_effective_date).

Example

Assume grade rate values shown in the following table already exist in the pay_grade_rules_f table:

Grade_rule_id	Effective Start_ Date	Effective_ End_ Date	Object_ Version_ Number	Value
12122	01-JAN-1996	20-FEB-1996	2	45
12122	21-FEB-1996	20-JUN-1998	3	50

Also assume that the grade rate value was updated to the wrong value on 21-FEB-1996. The update from 45 to 50 should have been 45 to 55 and you want to correct the error.

```
declare
  l_object_version_number number;
  l_effective_start_date  date;
  l_effective_end_date    date;
begin
  l_object_version_number := 3;
  hr_grade_api.update_grade_rate_value
    (p_effective_date      => to_date('21-02-1996','DD-MM-YYYY')
    ,p_datetrack_update_mode => 'CORRECTION'
    ,p_grade_rule_id         => 12122
    ,p_object_version_number => l_object_version_number
    ,p_value                 => 55
    ,p_effective_start_date  => l_effective_start_date
    ,p_effective_end_date    => l_effective_end_date
    );
  -- l_object_version_number will now be set to the value
  -- as on database row, as of 21st February 1996.
end;
```

Understanding the p_datetrack_delete_mode Control Parameter

Most APIs that delete data for at least one DateTrack entity have a p_datetrack_delete_mode control parameter. It enables you to define the type of DateTrack deletion to be made. This mandatory parameter must be set to one of the values in the following table:

p_datetrack_delete_mode Value	Description
ZAP	Completely remove from the database
DELETE	Set end date to effective date
FUTURE_CHANGE	Remove all scheduled changes
DELETE_NEXT_CHANGE	Remove next change

It may not be possible to use every mode in every case. For example, if there are no existing future changes for the record you are changing, the DateTrack modes FUTURE_CHANGE and DELETE_NEXT_CHANGE cannot be used. Some APIs that update DateTrack entities do not have a p_datetrack_delete_mode parameter. These APIs automatically perform the DateTrack operations for that business operation. Refer to the comments in each API package header creation script for further details.

Each dated instance for the same primary key has a different object_version_number. When calling the API the p_object_version_number parameter should be set to the value that applies as of the date for the operation (that is, p_effective_date).

Example

Assume that the grade rate values shown in the following table already exist in the pay_grade_rules_f table:

Grade_rule_id	Effective_ Start_Date	Effective_ End_ Date	Object_ Version_ Number	Value
5482	15-JAN-1996	23-MAR-1996	4	10
5482	24-MAR-1996	12-AUG-1996	8	20

Also assume that you want to remove all dated instances of this grade rate value from the database.

```
declare
  l_object_version_number number;
  l_effective_start_date  date;
  l_effective_end_date    date;
begin

  l_object_version_number := 4;

  hr_grade_api.update_grade_rate_value
    (p_effective_date     => to_date('02-02-1996', 'DD-MM-YYYY')
    ,p_datetrack_delete_mode => 'ZAP'
    ,p_grade_rule_id         => 5482
    ,p_object_version_number => l_object_version_number
    ,p_effective_start_date  => l_effective_start_date
    ,p_effective_end_date    => l_effective_end_date
    );

  -- As ZAP mode was used l_object_version_number now is null.
end;
```

Understanding the p_effective_start_date and p_effective_end_date Parameters

Most APIs that insert/delete/update data for at least one DateTrack entity have the p_effective_start_date and p_effective_end_date control parameters.

Both of these parameters are defined as OUT.

The values returned correspond to the effective_start_date and effective_end_date database column values for the row that is effective as of p_effective_date.

These parameters are set to null when all the DateTracked instances of a particular row are deleted from the database (that is, when a delete style API is called with a DateTrack mode of ZAP).

Example

Assume that the grade rate values in the following table already exist in the pay_grade_rules_f table:

Grade_rule_id	Effective_ Start_Date	Effective_ End_Date
17392	01-FEB-1996	24-MAY-1996
17392	25-MAY-1996	01-SEP-1997

The update_grade_rate_value API is called to perform a DateTrack mode of UPDATE_CHANGE_INSERT with an effective date of 10-MAR-1996. The API then modifies the database rows as shown in the following table:

Grade_rule_id	Effective_ Start_Date	Effective_ End_Date
17392	01-FEB-1996	09-MAR-1996
17392	10-MAR-1996	24-MAY-1996
17392	25-MAY-1996	01-SEP-1997

The API p_effective_start_date parameter is set to 10-MAR-1996 and p_effective_end_date to 24-MAY-1996.

Understanding the p_language_code Parameter

The p_language_code parameter is only available on create and update style Multilingual Support APIs. It enables you to specify which language the translation values apply to. The parameter can be set to the base or any installed language. The parameter default value of hr_api.userenv_lang is equivalent to:

```
select userenv('LANG')

    from dual;
```

If this parameter is set to null or hr_api.g_varchar2, the hr_api.userenv_lang default is still used.

See: Multilingual Support, page 2-238

API Features

Commit Statements

None of the HRMS APIs issue a commit. It is the responsibility of the calling code to issue commit statements. This ensures that parts of a transaction are not left in the database. If an error occurs, the whole transaction is rolled back. Therefore API work is either all completed or none of the work is done. You can use the HRMS APIs as "building blocks" to construct your own business functions. This gives you the flexibility to issue commits where you decide.

It also avoids conflicts with different client tools. For example, Oracle Forms only issues a commit if all the user's changes are not in error. This could be one or more record changes, which are probably separate API calls.

Avoiding Deadlocks

If calling more than one API in the same commit unit, take care to ensure deadlock situations do not happen. Deadlocks should be avoided by accessing the tables in the order they are listed in the table locking ladder. For example, you should update or delete rows in the table with the lowest Processing Order first.

If more than one row in the same table is being touched, then lock the rows in ascending primary key order. For example, if you are updating all the assignments for one person, then change the row with the lowest assignment_id first.

If it is impossible or impractical for operations to be done in locking ladder order, explicit locking logic is required. When a table is brought forward in the processing order, any table rows that have been jumped and will be touched later must be explicitly locked

in advance. Where a table is jumped and none of the rows are going to be updated or deleted, no locks should be taken on that table.

Example

Assume that the locking ladder order is as shown in the following table:

Table	Processing Order
A	10
B	20
C	30
D	40

Also assume that your logic has to update rows in the following order:

A	1st
D	2nd
C	3rd

Then your logic should:

1. Update rows in table A.
2. Lock rows in table C. (Only need to lock the rows that are going to be updated in step 4.)
3. Update rows in table D.
4. Update rows in table C.

Table B is not locked because it is not accessed after D. Your code does not have to explicitly lock rows in tables A or D, because locking is done as one of the first steps in the API.

In summary, you can choose the sequence of updates or deletes, but table rows must be locked in the order shown by the table locking ladder.

Flexfields with APIs

APIs validate the Descriptive Flexfield and Key Flexfield column values using the Flexfield definitions created using the Oracle Application Object Library Forms.

As the API Flexfield validation is performed within the database, the value set definitions should not refer directly to Forms objects such as fields. Server-side validation cannot resolve these references so any checks will fail. Care should also be taken when referencing profiles, as these values may be unavailable in the server-side.

Even where the Forms do not currently call the APIs to perform their commit time processing, it is strongly recommended that you do not directly refer to any Form fields in your value set definitions. Otherwise problems may occur with future upgrades. If you want to perform other field validation or perform Flexfield validation that cannot be implemented in values sets, use API User Hooks.

See: API User Hooks, page 2-245

For further information about, and solutions to, some problems that you may encounter with flexfield validation, see: Validation of Flexfield Values, page 2-204.

The APIs do not enforce Flexfield value security. This can only be done when using the Forms user interface.

For each Descriptive Flexfield, Oracle Applications has defined a structure column. In most cases the structure column name ends with the letters, or is called, "ATTRIBUTE_CATEGORY". The implementation team can associate this structure column with a reference field. The structure column value can affect which Flexfield structure is for validation. When reference fields are defined and you want to call the APIs, it is your responsibility to populate and update the ATTRIBUTE_CATEGORY value with the reference field value.

For Descriptive Flexfields, the APIs usually perform the Flexfield validation after other column validation for the current table. For Key Flexfield segments, values are held on a separate table, known as the combination table. As rows are maintained in the combination table ahead of the main product table, the APIs execute the Flexfield validation before main product table column validation.

In Release 11.0 and before, it was necessary to edit copies of the skeleton Flexfield validation package body creation scripts before the APIs could perform Flexfield validation. The technology constraints that made this technique necessary have now been lifted. These skeleton files *fli.pkb are no longer shipped with the product.

Multilingual Support

Several entities in the HRMS schema provide Multilingual Support (MLS), where translated values are held in _TL tables. For general details of the MLS concept refer to the following documentation:

See: *Oracle Applications Concepts Manual* for Principles of MLS, and *Oracle Applications Install Guide* for Configuration of MLS.

As the non-translated and translated values are identified by the same surrogate key ID column and value, the Multilingual Support APIs manage both groups of values in the same PL/SQL procedure call.

Create and update style APIs have a p_language_code parameter which you use to indicate which language the translated values apply to. The API maintains the required rows in the _TL table, setting the source_lang and language columns appropriately. These columns, and the p_language_code parameter, hold a language_code value from the FND_LANGUAGES table.

The p_language_code parameter has a default value of hr_api.userenv_lang, which is equivalent to:

```
select userenv('LANG')

  from dual;
```

Setting the p_language_code parameter enables you to maintain translated data for different languages within the same database session. If this parameter is set to null or hr_api.g_varchar2 then the hr_api.userenv_lang default is still used.

When a create style Multilingual Support API is called, a row is inserted into the _TL table for each base and installed language. For each row, the source_lang column equals the p_language_code parameter and the translated column values are the same. When

the other translated values are available they can be set by calling the update API, setting the p_language_code parameter to the appropriate language code.

Each call to an update style Multilingual Support API can amend the non-translated values and one set of translated values. The API updates the non-translated values in the main table and translated data values on corresponding row, or rows, in the _TL table. The translated columns are updated on rows where the p_language_code parameter matches the language or source_lang columns. Including a matching against the source_lang column ensures translations that have not been explicitly set remain synchronised with the created language. When a translation is being set for the first time the source_lang column is also updated with the p_language_code value. If you want to amend the values for another translation, call the update API again setting the p_language_code and translated parameters appropriately.

For delete style Multilingual Support APIs there is no p_language_code parameter. When the non-translated data is removed, all corresponding translation rows in the _TL table are also removed. So the API does not need to perform the process for a particular language.

When a Multilingual Support API is called more than one row may be processed in the _TL table. To avoid identifying every row that will be modified, _TL tables do not have an object_version_number column. The main table, holding the non-translated values, does have an object_version_number column. When you use a Multilingual Support API, set the p_object_version_number parameter to the value from the main table, even when only updating translated values.

Alternative APIs

In some situations it is possible to perform the same business process using more than one API. This is especially the case where entities hold extra details for different legislations. Usually there is a main API, which can be used for any legislation, and also specific versions for some legislations. Whichever API is called, the same validation and changes are made to the database.

For example, there is an entity to hold addresses for people. For GB style addresses some of the general address attributes are used to hold specific details, as shown in the following table:

PER_ADDRESSES Table Column Name	create_person_address API Parameter Name	create_gb_person_ address API Parameter Name
style	p_style	N/A
address_line1	p_address_line1	p_address_line1
address_line2	p_address_line2	p_address_line2
address_line3	p_address_line3	p_address_line3
town_or_city	p_town_or_city	p_town
region_1	p_region_1	p_county
region_2	p_region_2	N/A for this style
region_3	p_region_3	N/A for this style
postal_code	p_postal_code	p_postcode
country	p_country	p_country
telephone_number_1	p_telephone_number_1	p_telephone_number
telephone_number_2	p_telephone_number_2	N/A for this style
telephone_number_3	p_telephone_number_3	N/A for this style

Note: Not all database columns names or API parameters have been listed.

The p_style parameter does not exist on the create_gb_person_address API because this API only creates addresses for one style.

Not all of the address attributes are used in every style. For example, the region_2 attribute cannot be set for a GB style address. Hence, there is no corresponding parameter on the create_gb_person_address API. When the create_person_address API is called with p_style set to "GB" then p_region_2 must be null.

Both interfaces are provided to give the greatest flexibility. If your company only operates in one location, you may find it more convenient to call the address style interface that corresponds to your country. If your company operates in various locations and you want to store the address details using the local styles, you may find it more convenient to call the general API and specify the required style on creation.

Refer to comments in each API package header creation script for further details of where other alternative interfaces are provided.

See also: User Hooks and Alternative Interface APIs, page 2-262

API Errors and Warnings

Failure Errors

When calling APIs, validation or processing errors may occur. These errors are raised like any other PL/SQL error in Oracle applications.

When an error is raised, all the work done by that single API call is rolled back. As the APIs do not issue any commits, there is no danger that part of the work will be left in the database. It is the responsibility of the calling code to issue commits.

Warning Values

Warnings are returned using OUT parameters. The names of these parameters ends with _WARNING. In most cases the datatype is boolean. When a warning value is raised, the parameter is set to true. Other values are returned when the datatype is not boolean. Refer to the comments in each API package header creation script for further details.

The API assumes that although a warning situation has been flagged, it is acceptable to continue. If there was risk of a serious data problem, a PL/SQL error would have been raised and processing for the current API call would have stopped.

However, in your particular organization you may need to make a note about the warning or perform further checks. If you do not want the change to be kept in the database while this is done, you will need to explicitly roll back the work the API performed.

Example

When the create_employee API is called, the *p_name_combination_warning* parameter is set to true when person details already in the database include the same combination of last_name, first_name and date_of_birth.

```
declare
  l_name_combination_warning  boolean;
  l_assign_payroll_warning    boolean;
begin
  savepoint on_name_warning;
  hr_employee.create_employee
    (p_validate           => false
    ...
    ,p_last_name          => 'Bloggs'
    ,p_first_name         => 'Fred'
    ,p_date_of_birth      => to_date('06-06-1964', 'DD-MM-YYYY')
    ...
    ,p_name_combination_warning  => l_name_combination_warning
    ,p_assign_payroll_warning    => l_assign_payroll_warning
    );
  if l_name_combination_warning then
    -- Note that similar person details already exist.
    -- Do not hold the details in the database until it is
    -- confirmed this is really a different person.
    rollback to on_name_warning;
  end if;
end;
```

> **Note:** It would not have been necessary to rollback the API work if the p_validate parameter had been set to true.

You should not use our API procedure names for the savepoint names. An unexpected result may occur if you do not use different names.

Handling Errors in PL/SQL Batch Processes

In a batch environment, errors raised to the batch process must be handled and recorded so that processing can continue. To aid the development of such batch processes, we

provide a message table called HR_API_BATCH_MESSAGE_LINES and some APIs, as shown in the following table:

API Name	Description
create_message_line	Adds a single error message to the HR_API_BATCH_MESSAGE_LINES table.
delete_message_line	Removes a single error message to the HR_API_BATCH_MESSAGE_LINES table.
delete_message_lines	Removes all error message lines for a particular batch run.

For a full description of each API, refer to the comments in the package header creation script.

For handling API errors in a PL/SQL batch process it is recommended that any messages should be stored in the HR_API_BATCH_MESSAGE_LINES table.

Example PL/SQL Batch Program

Assume a temporary table has been created containing employee addresses. The addresses need to be inserted into the HR schema. The temporary table holding the address is called temp_person_address, as in the following table. It could have been populated from an ASCII file using Sql*Loader.

TEMP_PERSON_ADDRESSES Table

Column Name	DataType
person_id	number
primary_flag	varchar2
date_from	date
address_type	varchar2
address_line1	varchar2
address_line2	varchar2
address_line3	varchar2
town	varchar2
county	varchar2
postcode	varchar2
country	varchar2
telephone_number	varchar2

Sample Code

```
declare
  --
  l_rows_processed  number := 0; -- rows processed by api    l_com
mit_point      number := 20; - Commit after X successful rows
  l_batch_run_number         hr_api_batch_message_lines.batch_run_
number%type;
  l_dummy_line_id         hr_api_batch_message_lines.line_id%type;
    l_address_id            per_addresses.address_id%type;
  l_object_version_number_id per_addresses.object_version_number_i
d%type;
  --
  -- select the next batch run number
  --
  cursor csr_batch_run_number is
    select nvl(max(abm.batch_run_number), 0) + 1
      from hr_api_batch_message_lines abm;
  --
  -- select all the temporary 'GB' address rows
  --
  cursor csr_tpa is
    select tpa.person_id
         , tpa.primary_flag
         , tpa.date_from
         , tpa.address_type
         , tpa.address_line1
         , tpa.address_line2
         , tpa.address_line3
         , tpa.town
         , tpa.county
         , tpa.postcode
         , tpa.country
         , tpa.telephone_number
         , tpa.rowid
      from temp_person_addresses tpa
     where tpa.address_style = 'GB';
begin
  -- open and fetch the batch run number
  open csr_batch_run_number;
  fetch csr_batch_run_number into l_batch_run_number;
  close csr_batch_run_number;
  -- open and fetch each temporary address row
  for sel in csr_tpa loop
    begin
      -- create the address in the HR Schema
      hr_person_address_api.create_gb_person_address
        (p_person_id             => sel.person_id
        ,p_effective_date        => trunc(sysdate)
        ,p_primary_flag          => sel.primary_flag
        ,p_date_from             => sel.date_from
        ,p_address_type          => sel.address_type
        ,p_address_line1         => sel.address_line1
        ,p_address_line2         => sel.address_line2
        ,p_address_line3         => sel.address_line3
        ,p_town                  => sel.town
        ,p_county                => sel.county
        ,p_postcode              => sel.postcode
```

```
        ,p_country               => sel.country
        ,p_telephone_number      => sel.telephone_number
        ,p_address_id            => l_address_id
        ,p_object_version_number => l_object_version_number
        );
      -- increment the number of rows processed by the api
      l_rows_processed := l_rows_processed + 1;
      -- determine if the commit point has been reached
      if (mod(l_rows_processed, l_commit_point) = 0) then
        -- the commit point has been reached therefore commit
        commit;
      end if;
    exception
      when others then
        --
        -- An API error has occurred
        -- Note: As an error has occurred only the work in the
         -- last API call will be rolled back. The
          -- uncommitted work done by previous API calls will not
be          -- affected. If the error is ora-20001 the fnd_message
.get          -- function will retrieve and substitute all tokens
for            -- the short and extended message text. If the erro
r is            -- not ora-20001, null will be returned.
        --
        hr_batch_message_line_api.create_message_line
          (p_batch_run_number       => l_batch_run_number
          ,p_api_name               =>
                          'hr_person_address_api.create_gb_person
_address'
          ,p_status                 => 'F'
          ,p_error_number           => sqlcode
          ,p_error_message          => sqlerrm
          ,p_extended_error_message => fnd_message.get
          ,p_source_row_information => to_char(sel.rowid)
          ,p_line_id                => l_dummy_line_id);
    end;
  end loop;
  -- commit any final rows
  commit;
end;
```

You can view any errors that might have been created during the processes by selecting from the HR_API_BATCH_MESSAGE_LINES table for the batch run completed, as follows:

```
select *

  from hr_api_batch_message_lines abm

 where abm.batch_run_number = :batch_run_number

 order by abm.line_id;
```

WHO Columns and Oracle Alert

In many tables in Oracle Applications there are standard WHO columns. These include:

- LAST_UPDATE_DATE

- LAST_UPDATED_BY
- LAST_UPDATE_LOGIN
- CREATED_BY
- CREATION_DATE

The values held in these columns usually refer to the Applications User who caused the database row to be created or updated. In the Oracle HRMS Applications these columns are maintained by database triggers. You cannot directly populate these columns, as corresponding API parameters have not been provided.

When the APIs are executed from an Application Form or concurrent manager session, then these columns will be maintained just as if the Form had carried out the database changes.

When the APIs are called from a SQL*Plus database session, the CREATION_DATE and LAST_UPDATE_DATE column will still be populated with the database *sysdate* value. As there are no application user details, the CREATED_BY, LAST_UPDATED_BY and LAST_UPDATE_LOGIN column will be set to the "anonymous user" values.

If you want the CREATED_BY and LAST_UPDATED_BY columns to be populated with details of a known application user in a SQL*Plus database session, then before executing any HRMS APIs, call the following server-side package procedure once:

```
fnd_global.apps_initialize
```

If you call this procedure it is your responsibility to pass in valid values, as incorrect values are not rejected. The above procedure should also be called if you want to use Oracle Alert and the APIs.

By using AOL profiles, it is possible to associate a HR security profile with an AOL responsibility. Care should be taken when setting the apps_initialize resp_id parameter to a responsibility associated with a restricted HR security profile. To ensure API validation is not over restrictive, you should only maintain data held within that responsibility's business group.

To maintain data in more than one business group in the same database session, use a responsibility associated with an unrestricted HR security profile.

API User Hooks

APIs in Oracle HRMS support the addition of custom business logic. We have called this feature 'API User Hooks'. These hooks enable you to extend the standard business rules that are executed by the APIs. You can include your own validation rules or further processing logic and have it executed automatically whenever the associated API is executed.

Consider:

- Customer-specific data validation

  For example, when an employee is promoted you might want to restrict the change of grade to a single step, unless they work at a specific location, or have been in the grade for longer than six months.

- Maintenance of data held in extra customer-specific tables

  For example, you may want to store specific market or evaluation information about your employees in database tables that were not supplied by Oracle Applications.

- Capturing the fact that a particular business event has occurred

  For example, you may want to capture the fact that an employee is leaving the enterprise to send an electronic message directly to your separate security database, so the employee's office security pass can be disabled.

User hooks are locations in the APIs where extra logic can be executed. When the API processing reaches a user hook, the main processing stops and any custom logic is executed. Then, assuming no errors have occurred, the main API processing continues.

> **Caution:** You must not edit the API code files supplied by Oracle. These are part of the delivered product code and, if they are modified, Oracle may be unable to support or upgrade your implementation. Oracle Applications support direct calls only to the published APIs. Direct calls to any other server-side package procedures or functions that are written as part of the Oracle HRMS product set are not supported, unless explicitly specified.

Implementing API User Hooks

All the extra logic that you want to associate with APIs should be implemented as separate server-side package procedures using PL/SQL. The analysis and design of your business rules model is specific to your implementation. This essay focuses on how you can associate the rules you decide to write with the API user hooks.

After you have written and loaded into the database your server-side package, you need to associate your package with one or more specific user hooks. There are 3 special APIs to insert, update and delete this information. To create the links between the delivered APIs and the extra logic, execute the supplied pre-processor program. This looks at the data you have defined, the package procedure you want to call and builds logic to execute your PL/SQL from the specific user hooks. This step is provided to optimize the overall performance of API execution with user hooks. Effectively each API knows the extra logic to perform without needing to check explicitly.

As the link between the APIs and the extra logic is held in data, upgrades are easier to support. Where the same API user hooks and parameters exist in the new version, the pre-processor program can be executed again. This process rebuilds the extra code needed to execute your PL/SQL from the specific user hooks without the need for manual edits to Oracle applications or your own source code files.

To implement API user hooks

1. Identify the APIs and user hooks where you want to attach your extra logic. See: Available User Hooks, page 2-247
2. Identify the data values available at the user hooks you intend to use. See: Data Values Available at User Hooks, page 2-250
3. Implement your extra logic in a PL/SQL server-side package procedure. See: Implementing Extra Logic in a Separate Procedure Package, page 2-251
4. Register your extra PL/SQL packages with the appropriate API user hooks by calling the *hr_api_hook_call_api.create_api_hook_call* API. Define the mapping data between the user hook and the server-side package procedure. See: Linking Custom Procedures to User Hooks, page 2-253
5. Execute the user hook pre-processor program. This validates the parameters to your PL/SQL server-side package procedure and dynamically generates another

package body directly into the database. This generated code contains PL/SQL to call the custom package procedures from the API user hooks. See: The API User Hook Pre-processor Program, page 2-258

Available User Hooks

API user hooks are provided in the HRMS APIs that create, maintain or delete information. For example, the create_employee and update_emp_asg_criteria APIs.

> **Note:** User hooks are not provided in alternative interface APIs. For example, create_us_employee and create_gb_employee are both alternatives to the create_employee API. You should associate any extra logic with the main API. Also user hooks are not provided in utility style APIs such as create_message_line.

A PL/SQL script is available that lists all the different user hooks.

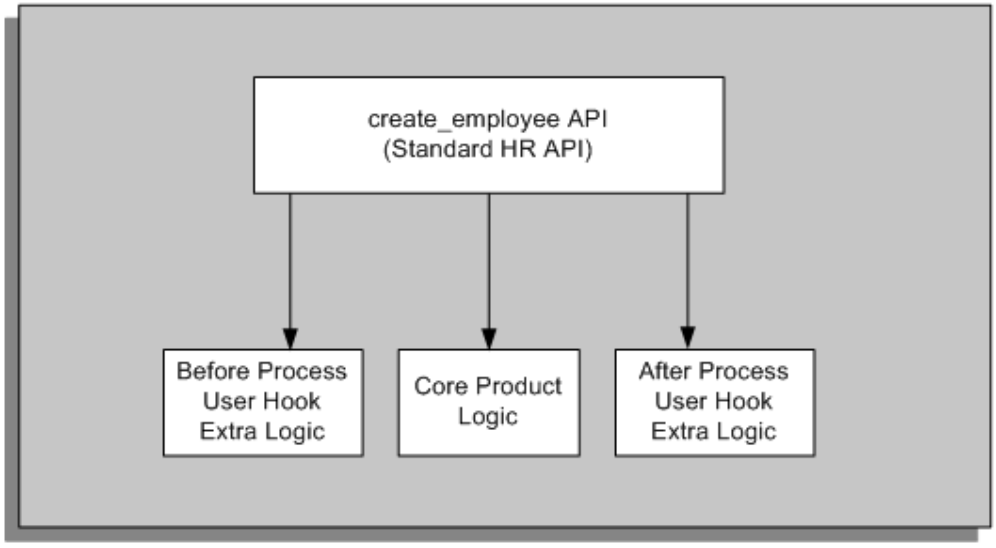
See: API User Hook Support Scripts, page 2-264

In the main APIs for HRMS there are two user hooks:

- Before Process
- After Process

There are different versions of these two user hooks in each API. For example, there is a *Before Process* and an *After Process* user hook in the create_employee API and a different *Before Process* and *After Process* user hook in the update_person API. This enables you to link your own logic to a specific API and user hook.

Main API User Hooks



Before Process Logic

Before Process user hooks execute any extra logic before the main API processing logic modifies any data in the database. In this case, the majority of validation will not have been executed. If you implement extra logic from this type of user hook, you must remember that none of the context and data values have been validated. It is possible the values are invalid and will be rejected when the main API processing logic is executed.

After Process Logic

After Process user hooks execute any extra logic after all the main API validation and processing logic has successfully completed. All the database changes that are going to be made by the API have been made. Any values provided from these user hooks have passed the validation checks. Your extra validation can assume the values provided are correct. If the main processing logic does not finish, due to an error, the After Process user hook is not called.

> **Note:** You cannot alter the core product logic, which is executed between the 'Before Process' and 'After Process' user hooks. You can only add extra custom logic at the user hooks.

Core Product Logic

Core Product Logic is split into a number of components. For tables that can be altered by an API there is an internal row handler code module. These rows handlers are implemented for nearly all the tables in the system where APIs are available. They control all the insert, update, delete and lock processing required by the main APIs. For example, if a main API needs to insert a new row into the PER_ALL_PEOPLE_F table it will not perform the DML itself. Instead it will execute the PER_ALL_PEOPLE_F row handler module.

Oracle Applications does not support any direct calls to these internal row handlers, as they do not contain the complete validation and processing logic. Calls are only allowed to the list of supported and published APIs.

This list is provided in the Publicly Callable Business Process APIs in Oracle HRMS , *Oracle HRMS Configuring, Reporting, and System Administration Guide* topic. Any new APIs introduced in the new version of a release will be listed in What's New in Oracle HRMS available on Metalink.

This list is provided in the Publicly Callable Business Process APIs in Oracle HRMS topic in the guide *Configuring, Reporting and System Administration in Oracle HRMS* and in Oracle HRMS Help. Any new APIs introduced in the new version of a release will be listed in the *What's New in Oracle HRMS* topic in the help system.

In each of the row handler modules three more user hooks are available, *After Insert*, *After Update* and *After Delete*. The user hook extra logic is executed after the validation specific to the current table columns has been successfully completed and immediately after the corresponding table DML statement.

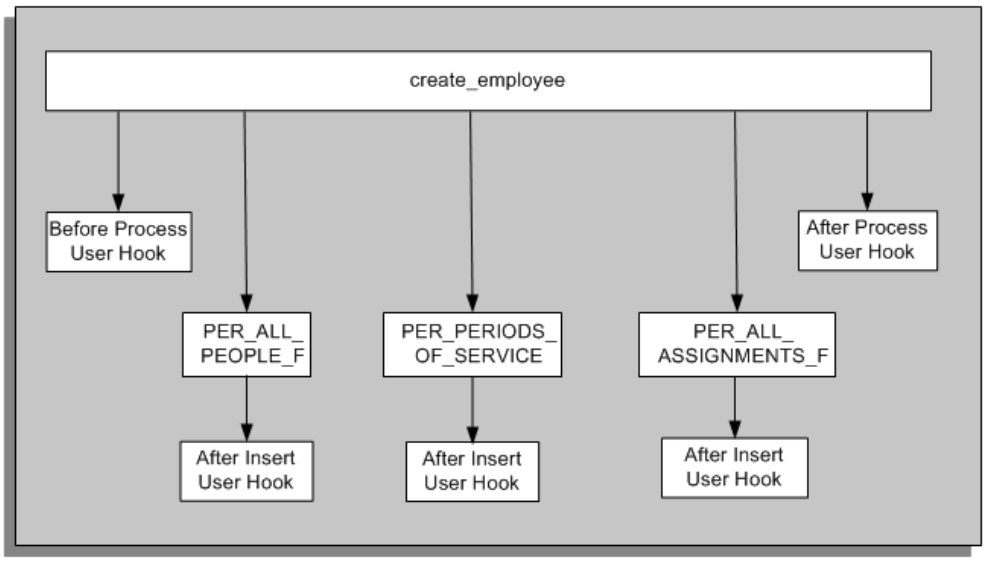
These row handler user hooks are provided after the DML has been completed for two reasons:

- All core product validation has been carried out. So you know that the change to that particular table is valid.
- For inserts, the primary key value is not known until the row has actually been inserted.

> **Note:** Although the update or delete DML statements may have been executed, the previous - before DML, column values are still available for use in any user hook logic. This is explained in more detail in a later section of this essay.

When an API inserts, updates or deletes records in more than one table there are many user hooks available for your use. For example, the create_employee API can create data in up to six different tables.

Create Employee API Summary Code Module Structure



In the above diagram, *create_employee* is the supported and published API. Only three of the internal row handlers have been shown, *PER_ALL_PEOPLE_F*, *PER_PERIODS_OF_SERVICE* and *PER_ALL_ASSIGNMENTS_F*. These internal row handlers must not be called directly.

Order of user hook execution:

1st) Create employee API *Before Process* user hook.

2nd) PER_ALL_PEOPLE_F row handler *After Insert* user hook.

3rd) PER_PERIODS_OF_SERVICE row handler *After Insert* user hook.

4th) PER_ALL_ASSIGNMENT_F row handler *After Insert* user hook.

...

last) Create employee API *After Process* user hook.

> **Note:** Core product validation and processing logic is executed between each of the user hooks.

When a validation or processing error is detected, processing is immediately aborted by raising a PL/SQL exception. API validation is carried out in each of the separate code modules. For example, when the create_employee API is used, validation logic is executed in each of the row handlers that are executed. Let's assume that a validation check is violated in the PER_PERIODS_OF_SERVICE row handler. The logic defined against the first two user hooks is executed. As a PL/SQL exception is raised, the 3rd and all remaining user hooks for that API call are not executed.

> **Note:** When a DateTrack operation is carried out on a particular record, only one row handler user hook is executed. For example, when

updating a person record using the DateTrack mode 'UPDATE', only the *After Update* user hook is executed in the PER_ALL_PEOPLE_F row handler.

The published APIs are also known as Business Processes as they perform a business event within HRMS.

Data Values Available at User Hooks

In general, where a value is known inside the API it will be available to the custom user hook code.

All values are read only. None of the values can be altered by user hook logic.

None of the AOL WHO values are available at any user hook, including:

- LAST_UPDATE_DATE
- LAST_UPDATED_BY
- LAST_UPDATE_LOGIN
- CREATED_BY
- CREATION_DATE

The p_validate parameter value is not available at any user hook. Any additional processing should be done regardless of the p_validate value.

Data values are made available to user hook logic using individual PL/SQL procedure parameters. In most cases the parameter name matches the name of the corresponding database column name with a *p_* prefix. For example, the NATIONALITY column on the PER_ALL_PEOPLE_F table has a corresponding user hook parameter name of p_nationality.

Before Process and After Process User Hook Data Values

- IN parameter values on each published API are available at the Before Process and After Process user hooks. At the Before Process hook none of the values are validated.
- OUT parameter values on the published API are only available from the After Process user hook. They are unavailable from the Before Process user hook because no core product logic has been executed to derive them.
- IN OUT parameter values on the published API are available at the Before Process and After Process user hooks. The potentially invalid IN value is available at the Before Process user hook. The value passed out of the published API is available at the After Process user hook.

From the row handler *After Insert* user hook only column values that can be populated or are derived during insert are available.

From the *After Update* user hook two sets of values are available: the new values and the old values. That is, the values that correspond to the updated record and the values that existed on the record before the DML statement was executed. The new value parameter names correspond to the database column name with a *p_* prefix. The old values parameter names match the database column name with a *p_* prefix and a *_o* suffix. For example, the new value parameter name for the NATIONALITY column on the PER_ALL_PEOPLE_F table is p_nationality. The old value parameter name is p_nationality_o.

Except for the primary key ID, if a database column cannot be updated a new value parameter is not available. There is still a corresponding parameter without the *_o* suffix. For example, the BUSINESS_GROUP_ID column cannot be updated on the PER_ALL_PEOPLE_F table. At the *After Update* user hook a p_business_group_id_o parameter is available. But there is no new value p_business_group_id parameter.

From the *After Delete* user hooks only old values are available with *_o* suffix style parameter names. The primary key ID value is available with a parameter that does not have the *_o* suffix.

Old values are only made available at the row handler *After Update* and *After Delete* user hooks. Old values are NOT available from any of the *Before Process*, *After Process* or *After Insert* user hooks.

Wherever the database column name is used, the end of the name may be truncated, to fit the PL/SQL 30 character limit for parameter names.

For DateTrack table row handlers, whenever data values are made available from the *After Insert*, *After Update* or *After Delete* user hooks, the provided new and old values apply as of the operation's effective_date. If past or future values are required the custom logic needs to select them explicitly from the database table. The effective_start_date and effective_end_date column and DateTrack mode value are made available.

A complete list of available user hooks and the data values provided can be found by executing a PL/SQL script.

See: API User Hook Support Scripts, page 2-264

Implementing Extra Logic In a Separate Package Procedure

Any extra logic that you want to link to an API with a user hook must be implemented inside a PL/SQL server-side package procedure.

> **Note:** These procedures can do anything that can be implemented in PL/SQL except 'commit' and full 'rollbacks'.

The APIs have been designed to perform all of the work associated with a business process. If it is not possible to complete all of the database changes then the API fails and rolls back all changes. This is achieved by not committing any values to the database within an API. If an error occurs in later processing all database changes made up to that point are rolled back automatically.

> **Important:** Commits or full rollbacks are not allowed in any API code as they would interfere with this mechanism. This includes user-hooks and extra logic. If you attempt to issue a commit or full rollback statement, the user hook mechanism will detect this and raise its own error.

When an invalid value is detected by extra validation, you should raise an error using a PL/SQL exception. This automatically rolls back any database changes carried out by the current call to the published API. This rollback includes any changes made by earlier user hooks.

The user hook code does not support any optional or decision logic to decide when your custom code should be executed. If you link extra logic to a user hook it will always be called when that API processing point is reached. You must implement any conditional logic inside your custom package procedure. For example, suppose you want to check that 'Administrators' are promoted by one grade step only with each change. As your

extra logic will be called for all assignments, regardless of job type, you should decide if you need to check for the job of 'Administrator' before checking the grade details.

Limitations

There are some limitations to implementing extra logic as custom PL/SQL code. Only calls to server-side package procedures are supported. But more than one package procedure can be executed from the same user hook. Custom PL/SQL cannot be executed from user hooks if it is implemented in:

- Stand alone procedures (not defined within a package)
- Package functions
- Stand alone package functions (not defined within a package)
- Package procedures that have overloaded versions

> **Note:** Do not try to implement commit or full rollback statements in your custom PL/SQL. This will interfere with the API processing and will generate an error.

When a parameter name is defined it must match exactly the name of a data value parameter that is available at the user hooks where it will be executed. The parameter must have the same datatype as the user hook data value. Any normal implicit PL/SQL data conversions are not supported from user hooks. All the package procedure parameters must be defined as IN, without any default value. OUT and IN OUT parameters are not supported in the custom package procedure.

At all user hooks many data values are available. When implementing a custom package procedure every data value does not have to be listed. Only the data values for parameters that are required for the custom PL/SQL need to be listed.

A complete list of available user hooks, data values provided and their datatypes can be found by executing a PL/SQL script.

See: API User Hook Support Scripts, page 2-264

When you have completed your custom PL/SQL package you should execute the package creation scripts on the database and test that the package procedure compiles. Then test that this carries out the intended validation on a test database.

Example

A particular enterprise requires the previous last name for all married females when they are entered in the system. This requirement is not implemented in the core product, but an implementation team can code this extra validation in a separate package procedure and call it using API user hooks. When marital status is 'Married' and sex is 'Female', use a PL/SQL exception to raise an error if the previous last name is null. The following sample code provides a server-side package procedure to perform this validation rule.

```
Create Or Replace Package cus_extra_person_rules as

procedure extra_name_checks

  (p_previous_last_name              in     varchar2

  ,p_sex                             in     varchar2

  ,p_marital_status                  in     varchar2
```

```
  );
end cus_extra_person_rules;
/
exit;
Create Or Replace Package Body cus_extra_person_rules as
procedure extra_name_checks
  (p_previous_last_name              in     varchar2
  ,p_sex                             in     varchar2
  ,p_marital_status                  in     varchar2
  ) is
begin
   -- When the person is a married female raise an
   -- error if the previous last name has not been
   -- entered
   if p_marital_status = 'M' and p_sex = 'F' then
     if p_previous_last_name is null then
       dbms_standard.raise_application_error
         (num => -20999
         ,msg => 'Previous last name must be entered for married
females'
         );
     end if;
   end if;
end extra_name_checks;
end cus_extra_person_rules;
/
exit;
```

Linking Custom Procedures to User Hooks

After you have executed the package creation scripts on your intended database, link the custom package procedures to the appropriate API user hooks. The linking between user

hooks and custom package procedures is defined as data in the HR_API_HOOK_CALLS table.

There are three special APIs to maintain data in this table:

- hr_api_hook_call_api.create_api_hook_call
- hr_api_hook_call_api.update_api_hook_call
- hr_api_hook_call_api.delete_api_hook_call

HR_API_HOOK_CALLS

- The HR_API_HOOK_CALLS table must contain one row for each package procedure linking to a specific user hook.
- The API_HOOK_CALL_ID column is the unique identifier.
- The API_HOOK_ID column specifies the user hook to link to the package procedure.

  This is a foreign key to the HR_API_HOOKS table. Currently the user hooks mechanism only support calls to package procedures, so the API_HOOK_CALL_TYPE column must be set to 'PP'.
- The ENABLED_FLAG column indicates if the user hook call should be included.

  It must be set to 'Y' for Yes, or 'N' for No.
- The SEQUENCE column is used to indicate the sequence of hook calls. Lowest numbers are processed first.

  The user hook mechanism is also used by Oracle to supply application, legislation, and vertical market specific PL/SQL. The sequence numbers from 1000 to 1999 inclusive, are reserved for Oracle internal use.

  You can use sequence numbers less than 1000 or greater than 1999 for custom logic. Where possible we recommend you use sequence numbers greater than 2000. Oracle specific user hook logic will then be executed first. This will avoid the need to duplicate Oracle's additional logic in the custom logic.

There are two other tables that contain data used by the API user hook mechanism, HR_API_MODULES and HR_API_HOOKS.

HR_API_MODULES Table

The HR_API_MODULES table contains a row for every API code module that contains user hooks.

HR_API_MODULES Main Columns	Description
API_MODULE_ID	Unique identifier
API_MODULE_TYPE	A code value representing the type of the API code module. 'BP' for Business Process APIs - the published APIs. 'RH' for the internal Row Handler code modules.
MODULE_NAME	The value depends on the module type. For 'BP' the name of the published API, such as CREATE_EMPLOYEE. For 'RH' modules the name of the table, such as PER_PERIODS_OF_SERVICE.

HR_API_HOOKS Table

The HR_API_HOOKS table is a child of the HR_API_MODULES table. It contains a record for each user hook in a particular API code module.

HR_API_HOOKS Main Columns	Description
API_HOOK_ID	Unique identifier
API_MODULE_ID	Foreign key. Parent ID to the HR_API_MODULES table.
API_HOOK_TYPE	Code value representing the type of user hook.

The API_HOOK_TYPE code represents the type of user hook, as shown in the following table:

User Hook Type	API_HOOK_TYPE
After Insert	AI
After Update	AU
After Delete	AD
Before Process	BP
After Process	AP

Caution: Data in the HR_API_MODULES and HR_API_HOOKS tables is supplied and owned by Oracle. Oracle also supplies some data in the HR_API_HOOK_CALLS table. Customers must not modify data in these tables. Any changes you make to these tables may affect product functionality and may invalidate your support agreement with Oracle.

Note: Data in these tables may come from more than one source and API_MODULE_IDs and API_HOOK_IDs may have different values on different databases. Any scripts you write must allow for this difference.

Full details for each of these tables can be found in the Oracle HRMS electronic Technical Reference Manual (eTRM) available on MetaLink.

Example

For the example where you want to make sure previous name is entered, the extra validation needs to be executed whenever a new person is entered into the system. The best place to execute this validation is from the PER_ALL_PEOPLE_F row handler *After Insert* user hook.

The following PL/SQL code is an example script to call the *create_api_hook_call* API. This tells the user hook mechanism that the *cus_extra_person_rules.extra_name_checks* package procedure should be executed from the PER_ALL_PEOPLE_F row handler *After Insert* user hook.

```
declare


  --

  -- Declare cursor statements

  --

  cursor cur_api_hook is

    select ahk.api_hook_id

      from hr_api_hooks   ahk

         , hr_api_modules ahm

     where ahm.module_name   = 'PER_ALL_PEOPLE_F'

       and ahm.api_module_type   = 'RH'

       and ahk.api_hook_type = 'AI'

       and ahk.api_module_id = ahm.api_module_id;

  --

  -- Declare local variables

  --

  l_api_hook_id            number;

  l_api_hook_call_id       number;

  l_object_version_number  number;

begin
```

```
--
-- Obtain the ID if the PER_ALL_PEOPLE_F
-- row handler After Insert API user hook.
--
open cursor csr_api_hook;
fetch csr_api_hook into l_api_hook_id;
if csr_api_hook %notfound then
  close csr_api_hook;
  dbms_standard.raise_application_error
       (num => -20999
       ,msg => 'The ID of the API user hook was not found'
       );
end if;
close csr_api_hook;
--
-- Tell the API user hook mechanism to call the
-- cus_extra_person_rules.extra_name_checks
-- package procedure from the PER_ALL_PEOPLE_F row
-- handler module 'After Insert' user hook.
--
hr_api_hook_call_api.create_api_hook_call
  (p_validate              => false
  ,p_effective_date        =>
               to_date('01-01-1997', 'DD-MM-YYYY')
  ,p_api_hook_id           => l_api_hook_id
  ,p_api_hook_call_type    => 'PP'
  ,p_sequence              => 3000
  ,p_enabled_flag          => 'Y'
  ,p_call_package          =>
```

```
                    'CUS_EXTRA_PERSON_RULES'

    ,p_call_procedure        => 'EXTRA_NAME_CHECKS'

    ,p_api_hook_call_id      => l_api_hook_call_id

    ,p_object_version_number =>

                    l_object_version_number

    );

  commit;

end;
```

In this example, the previous_last_name, sex and marital_status values can be updated. If you want to perform the same checks when the marital_status is changed, then the same validation will need to be executed from the PER_ALL_PEOPLE_F *After Update* user hook. As the same data values are available for this user hook, the same custom package procedure can be used. Another API hook call definition should be created in HR_API_HOOK_CALLS by calling the *create_api_hook_call* API again. This time the *p_api_hook_id* parameter needs to be set to the ID of the PER_ALL_PEOPLE_F *After Update* user hook.

The API User Hook Pre-processor Program

Adding rows to the HR_API_HOOK_CALLS table does not mean the extra logic will be called automatically from the user hooks. You must run the API user hooks pre-processor program after the definition and the custom package procedure have both been created in the database. This looks at the calling definitions in the HR_API_HOOK_CALLS table and the parameters listed on the custom server-side package procedures.

> **Note:** Another package body will be dynamically built in the database. This is known as the hook package body.

There is no operating system file that contains a creation script for the hook package body. It is dynamically created by the API user hook pre-processor program. Assuming the various validation checks succeed, this package will contain hard coded calls to the custom package procedures.

If no extra logic is implemented, the corresponding hook package body will still be dynamically created. It will have no calls to any other package procedures.

The pre-processor program is automatically executed at the end of some server-side Oracle install and upgrade scripts. This ensures versions of hook packages bodies exist in the database. If you do not want to use API user hooks then no further setup steps are required.

The user hook mechanism is used by Oracle to provide extra logic for some applications, legislations, and vertical versions of the products. Calls to this PL/SQL are also generated into the hook package body.

> **Caution:** It is IMPORTANT that you do not make any direct edits to the generated hook package body. Any changes you make may affect product functionality and may invalidate your support agreement with Oracle. If you choose to make alternations, these will be lost the next time the pre-processor program is run. This will occur when the

Oracle install or upgrade scripts are executed. Other developers in the implementation team could execute the pre-processor program.

If any changes are required, modify the custom packages or the calling definition data in the HR_API_HOOK_CALLS table. Then rerun the pre-processor program to generate a new version of the hook package body. For example, if you want to stop calling a particular custom package procedure then:

1. Call the *hr_api_hook_call_api.update_api_hook_call* API, setting the *p_enabled_flag* parameter to 'N'.
2. Execute the API user hook pre-processor program so the latest definitions are read again and the hook package body is dynamically recreated.

If you want to include the call again, then repeat these steps and set the *p_enabled_flag* parameter in the *hr_api_hook_call_api.update_api_hook_call* API to 'Y'.

If you want to permanently remove a custom call from a user hook then remove the corresponding calling definition. Call the *hr_api_hook_call_api.delete_api_hook_call* API.

Remember that the actual call from the user hook package body will be removed only when the pre-processor program is rerun.

Running the Pre-processor Program

The pre-processor program can be run in two ways.

- Execute the *hrahkall.sql* script in SQL*Plus

  This creates the hook package bodies for all of the different API code modules.

- Execute the *hrahkone.sql* script in SQL*Plus

  This creates the hook package bodies for just one API code module - one main API or one internal row handler module.

  An *api_module_id* must be specified with this script. The required ID values are found in the HR_API_MODULES table.

Both the *hrahkall.sql* and *hrahkone.sql* scripts are stored in the $PER_TOP/admin/sql operating system directory.

Example

Continuing the previous example: After the calling definitions and custom package procedure have been successfully created in the database the api_module_id can be found with the following SQL statement:

```
select api_module_id

  from hr_api_modules

 where api_module_type = 'RH'

       and module_name = 'PER_ALL_PEOPLE_F';
```

Then execute the *hrahkone.sql* script. When prompted, enter the *api_module_id* returned by the SQL statement above. This will generate the hook package bodies for all of the PER_ALL_PEOPLE_F row handler module user hooks *After Insert*, *After Update* and *After Delete*.

Log Report

Both pre-processor programs produce a log report. The hrahkall.sql script only lists errors. So if no text is shown after the 'Created on' statement, all the hook package bodies have been created without any PL/SQL or application errors. The hrahkone.sql script outputs a successful comment or error details. If any errors occurred, a PL/SQL exception is deliberately raised at the end of both scripts. This highlights to the calling program that a problem has occurred.

When errors do occur the hook package body code may still be created with valid PL/SQL. For example, if a custom package procedure lists a parameter that is not available, the hook package body is still successfully created. No code is created to execute that particular custom package procedure. If other custom package procedures need to be executed from the same user hook, code to perform those calls is still created - assuming they pass all the standard PL/SQL checks and validation checks.

> **Important:** It is important that you check these log reports to confirm the results of the scripts. If a call could not be built the corresponding row in the HR_API_HOOK_CALLS table will also be updated. The STATUS column will be set to 'I' for Invalid Call and the ENCODED_ERROR column will be populated with the AOL application error message in the encoded format.

The encoded format can be converted into translated text by the following PL/SQL:

```
declare

  l_encoded_error varchar2(2000);

  l_user_read_text varchar2(2000);

begin

  -- Substitute ??? with the value held in the

  -- HR_API_HOOK_CALLS.ENCODED_ERROR column.

  l_encoded_error := ???;

  fnd_message.set_encoded(encoded_error);

  l_user_read_text := fnd_message.get;

end;
```

It is your responsibility to review and resolve any problems recorded in the log reports. Options:

- Alter the parameters in the custom package procedures.
- If required, change the data defined in the HR_API_HOOK_CALLS table.

When you have resolved any problems, rerun the pre-processor program.

The generated user hook package bodies must be less than 32K in size. This restriction is a limit in PL/SQL. If you reach this limit, you should reduce the number of separate package procedures called from each user hook. Try to combine your custom logic into fewer procedures.

Note: Each linked custom package procedure can be greater than 32K in size. Only the user hook package body that is dynamically created in the database must be less than 32K.

One advantage of implementing the API user hook approach is that your extra logic is called every time the APIs are called. This includes any HRMS Forms or Web pages that perform their processing logic by calling the APIs.

Important: The user hook mechanism that calls your custom logic is supported as part of the standard product. However the logic in your own custom PL/SQL procedures cannot be supported by Oracle Support.

Recommendations for Using the Different Types of User Hook

Consider your validation rules in two categories:

- Data Item Rules

  Rules associated with a specific field in a form or column in a table. For example, grade assigned must *always* be valid for the Job assigned.

- Business Process Rules

  Rules associated with a specific transaction or process. For example, when you create a secondary assignment you must include a special descriptive segment value.

Data Item Rules

The published APIs are designed to support business processes. This means that individual data items can be modified by more than one API. To perform extra data validation on specific data items (table columns), use the internal row handler module user hooks.

By implementing any extra logic from the internal row handler code user hooks, you will cover all of the cases where that column value can change. Otherwise you will need to identify all the APIs that can set or alter that database column.

Use the *After Insert*, *After Update* or *After Delete* user hooks for data validation. These hooks are preferred because all of the validation associated with the database table row must be completed successfully before these user hooks are executed. Any data values passed to custom logic will be valid as far as the core product is concerned.

If the hook call definition is created with a sequence number greater than 1999, then any Oracle legislation or vertical market specific logic will also have been successfully executed.

Note: If extra validation is implemented on the *After Insert* user hook, and the relevant data values can be updated, then you should consider excluding similar logic from the *After Update* user hook. Old values - before DML, are available from the *After Update* and *After Delete* user hooks.

Business Process Rules

If you want to detect that a particular business event has occurred, or you only want to perform some extra logic for a particular published API, use the *Before Process* and *After Process* user hooks.

Where possible, use the *After Process* user hook, as all core product validation for the whole API will have been completed. If you use the *Before Process* user hook you must consider that all data values could be invalid in your custom logic. None of the core product validation has been carried out at that point. References to the HR_LOOKUPS view, any views that join to HR_LOOKUPS and lookup code validation cannot be performed at the *Before Process* user hook. Values that affect the lookup code validation are not derived and set until after this point.

Data values provided at the *Before Process* and *After Process* user hooks will be the same as the values passed into the API. For update type business processes the API caller has to specify only the mandatory parameters and the values they actually want to change. When the API caller does not explicitly provide a parameter value, the system reserved default values will be used, as shown in the vollowing table:

Data Type	Default Value
varchar2	hr_api.g_varchar2
number	hr_api.g_number
date	hr_api.g_date

Depending on the parameters specified by the API caller, these default values may be provided to *Before Process* and *After Process* user hooks. That is, the existing column value in the database is only provided if the API calling code happens to pass the same new value. If the real database value is required then the custom package procedures must select it explicitly from the database.

This is another reason why *After Update* and *After Delete* user hooks are preferred. At the row handler user hooks the actual data value is always provided. Any system default values will have been reset with their existing database column value in the row handler modules. Any extra logic from these user hooks does need to be concerned with the system reserved default values.

If any *After Process* extra logic must access the old database values then a different user hook needs to be used. It will not be possible to use the *After Process* user hook because all the relevant database rows will have been modified and the old values will not be provided by the user hook mechanism. Where API specific extra logic requires the old values, they will need to be explicitly selected in the *Before Process* user hook.

User Hooks and Alternative Interface APIs

Alternative Interface APIs provide an alternative version of the generic APIs. Currently there are legislative or vertical specific versions of the generic APIs.

For example, *create_us_employee* and *create_gb_employee* are two alternative interfaces to the generic *create_employee* API. These alternatives make clear how specific legislative parameters are mapped onto the parameters of the generic API.

In the future other alternative APIs may be provided to support specific implementations of generic features, such as elements and input values.

> **Important:** User hooks are not provided in alternative interface APIs. User hooks are provided only in the generic APIs. In this example the user hooks are provided in the *create_employee* API and not in the *create_us_employee* and *create_gb_employee* APIs.

Alternative interface APIs always perform their processing by executing the generic API and any extra logic in the generic API user hooks is executed automatically when the alternative APIs are called. This guarantees consistency in executing any extra logic and reduces the administrative effort to set up and maintain the links.

Example 1

You want to perform extra validation on the job and payroll components of employee assignments to make sure only 'Machine Workers' are included in the 'Weekly' payroll. There is more than one published API that allows the values to be set when a new assignment is created or an existing assignment is updated.

> **Tip:** Implement the extra validation in a custom server-side package procedure. Link this to the two user hooks, *After Insert and After Update*, in the PER_ALL_ASSIGNMENTS_F table internal row handler module.

Example 2

You have a custom table and you want to create data in this table when a new employee is created in the system, or an existing applicant is converted into an employee. The data in the custom table does not need to be created in any other scenario.

> **Tip:** Implement the third party table; insert DML statements in a custom server-side package procedure. Link this to two user hooks: *After Process* in the *create_employee* API module and *After Process* in the *hire_applicant* API module.

Comparison with Database Triggers

User hooks have a number of advantages over database triggers for implementing extra logic.

- Database triggers can only be defined against individual table DML statements. The context of a particular business event may be unavailable at the table level because the event details are not held in any of the columns on that table.
- Executing a database trigger is inefficient compared with executing a server-side package procedure.
- The *mutating table* restriction stops values being selected from table rows that are being modified. This prevents complex multi-row validation being implemented from database triggers. This complex validation can be implemented from API user hooks, as there are no similar restrictions.
- On DateTrack tables it is extremely difficult to implement any useful logic from database triggers. With many DateTrack modes, a single transaction may affect more than one row in the same database table. Each dated instance of a DateTrack record is physically held on a different database row.

  For example, a database trigger that fires on insert cannot tell the difference between a new record being created or an insert row from a DateTrack 'UPDATE' operation.

  > **Note:** DateTrack 'UPDATE' carries out one *insert* and one *update* statement. The context of the DateTrack mode is lost at the database table level. You cannot re-derive this in a database trigger due to the mutating table restriction.

- With DateTrack table row handler user hooks more context and data values are available. The *After Insert* user hook is only executed when a new record is created. The DateTrack mode name is available at *After Update* and *After Delete* user hooks. The date range over which the record is being modified is also available at these user hooks. The *validation_start_date* value is the first day the record is affected by the current DateTrack operation. The last day the record is affected is known as the *validation_end_date*.

API User Hook Support Scripts

You can create a complete list of available user hooks and the data values provided by executing the *hrahkpar.sql* script in SQL*Plus. This script can be found in the $PER_TOP/admin/sql operating system directory. As the output is long, it is recommended to spool the output to an operating system text file.

The user hook pre-processor program can be executed in two ways. To create the hook package bodies for all of the different API code modules, execute the *hrahkall.sql* script in SQL*Plus. To create the hook package bodies for just one API code module, such as one main API or one internal row handler module, execute the *hrahkone.sql* script in SQL*Plus. An *api_module_id* must be specified with this second script. The required *api_module_id* value can be obtained from the HR_API_MODULES table. Both the *hrahkall.sql* and *hrahkone.sql* scripts can be found in the $PER_TOP/admin/sql operating system directory.

Using APIs as Building Blocks

The API code files supplied with the product must not be edited directly for any custom use.

> **Caution:** Any changes you make may affect product functionality and may invalidate your support agreement with Oracle and prevent product upgrades.

Oracle Applications supports direct calls to the published APIs. Direct calls to any other server-side package procedures or functions written as part of the Oracle HRMS product set are not supported, unless explicitly specified.

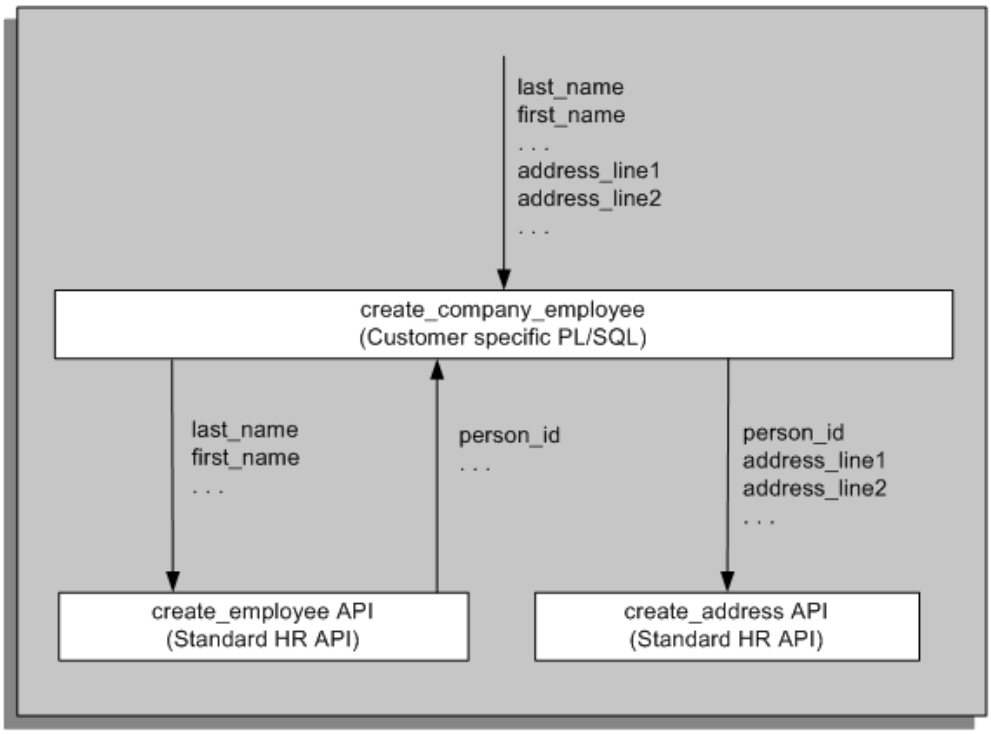
There are supported methods for adding custom logic, using the APIs provided. In addition to the API user hook mechanism, you can use the published APIs as building blocks to construct custom APIs.

Example

Suppose you always obtain a new employee's home address when they join your enterprise. The address details must be recorded in the HR system because you run reports that expect every employee to have an address.

You could write your own API to create new employees with an address. This API would call the standard *create_employee* API and then immediately afterwards call the standard *create_address* API.

Create Employee/Create Address APIs



With API user hooks it is not possible to change any of the data values. So the building block approach can be used to default or set any values before the published API is called.

The major disadvantage with the building block approach is that any Forms or Web pages supplied by Oracle will NOT call any custom APIs. If a user interface is required then you must also create your own custom Forms or Web pages to implement calls to your custom APIs.

Handling Object Version Numbers in Oracle Forms

If you intend to write your own Forms that call the APIs, you will need to implement additional Forms logic to correctly manage the object version number. This is required because of the way Forms can process more than one row in the same commit unit.

Example

Consider the following example of what can happen if only one form's block item is used to hold the object version number:

1. The user queries two rows and updates both.

Row	OVN in Database	OVN in Form
A	6	6
B	3	3

2. The user presses commit.

Row A has no user errors and is validated in the API. The OVN is updated in the database and the new OVN is returned to the form.

Row	OVN in Database	OVN in Form
A	7	7
B	3	3

3. The form calls the API again for row B.

   This time there is a validation error on the user-entered change. An error message is raised in the form and Forms issues a rollback to the database. However, the OVN for row A in the form is now different from the OVN in the database.

Row	OVN in Database	OVN in Form
A	6	7
B	3	3

4. The user corrects the problem with row B and commits again.

   Now the API will error when it validates the changes to row A. The two OVNs are different.

Solution

The solution to this problem is to use a non-basetable item to hold the new version number. This item is not populated at query time.

1. The user queries two rows and updates both.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	6	6	
B	3	3	

2. The user presses commit.

   Row A is valid, so the OVN is updated in the database and the new OVN is returned to the form.

   Note: The actual OVN in the form is not updated.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	6	7
B	3	3	

3. The forms calls the API again for row B.

The validation fails and an error message is raised in the form. Forms issues a rollback to the database.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	6	6	7
B	3	3	

4. The user corrects the problem with row B and commits again.

   The API is called to validate row A again. The OVN value is passed, not the NEW_OVN. There is no error because the OVN in the database now matches the OVN it was passed. The API passes back the updated OVN value.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	6	7
B	3	3	

5. The API is called again to validate row B.

   The validation is successful; the OVN is updated in the database and the new OVN value is returned to the form. The commit in the form and the database is successful.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	6	7
B	4	3	4

What would happen when the user updates the same row again without re-querying? Following on from the previous step:

6. When the user starts to update row A, the on-lock trigger will fire.

   The trigger updates the OVN when New_OVN is not null. (Theoretically the on-lock trigger will only fire if the previous commit has been successful. Therefore the New_OVN is the OVN value in the database.)

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	7	7

7. The on-lock trigger then calls the API to take out a lock using OVN.

   The lock is successful as the OVN values match.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	7	7

8. The user continues with the update, the update API is called, and the commit is successful.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	8	7	8

If user does delete instead of update, the on_lock will work in the same way. When key_delrec is pressed, the delete API should be called with p_validate set to true. Doing so ensures that the delete is valid without removing the row from the database.

Therefore, the OVN value in the form should be set with the New_OVN, when New_OVN is not null. This ensures that the delete logic is called with the OVN value in the database.

However, there is another special case that has to be taken into consideration. It is possible for the user to update a row (causing a new OVN value to be returned from the API), the update of the next row in the same commit unit fails, the user navigates back to the first row and decides to delete it. To stop the new_OVN from being copied into the OVN in the form, only do the copy in key_delrec if the record_status is query.

Example Code Using the Grade Rate Values

The above descriptions are handled in the following example. In this example, <block_name>.object_version_number is a basetable item and <block_name>.new_object_version_number is non-basetable.

Forms Procedure Called from the ON-INSERT Trigger

```
  procedure insert_row is
  begin
    --
    -- Call the api insert routine
    --
    hr_grade_api.create_grade_rate_value
      (<parameters>
      ,p_object_version_number => :<block_name>.object_version_num
ber
      ,p_validate              => false
      );
  end insert_row;
```

Forms Procedure Called from the ON-UPDATE Trigger

```
  procedure update_row is
    l_api_ovn  number;
  begin
    -- Send the old object version number to the API
    l_api_ovn := :<block_name>.object_version_number;
    --
    -- Call the api update routine
    --
    hr_grade_api.update_grade_rate_values
      (<parameters>
      ,p_object_version_number => l_api_ovn
      ,p_validate              => false
      );
    -- Remember the new object version number returned from the AP
I
    :<block_name>.new_object_version_number := l_api_ovn;
  end update_row;
```

Forms Procedure Called from the ON-DELETE Trigger

```
  procedure delete_row is
  begin
    --
    -- Call the api delete routine
    --
    hr_grade_api.delete_grade_rate_values
      (<parameters>
      ,p_object_version_number => :<block_name>.object_version_num
ber
      ,p_validate              => false
      );
  end delete_row;
```

Forms Procedure Called from the KEY-DELREC Trigger

```
  procedure key_delrec_row is
    l_api_ovn     number;

    l_rec_status  varchar2(30);
  begin
    -- Ask user to confirm they really want to delete this row.
    --

    -- Only perform the delete checks if the

    -- row really exists in the database.

    --

    l_rec_status := :system.record_status;

    if (l_rec_status = 'QUERY') or (l_rec_status = 'CHANGED') the
n

      --
      -- If this row just updated then the
```

```
      -- new_object_version_number will be not null.

      -- If that commit was successful then the

      -- record_status will be QUERY, therefore use

      -- the new_object_version_number. If the commit

      -- was not successful then the user must have

      -- updated the row and then decided to delete

      -- it instead. Therefore just use the

      -- object_version_number.

      --(Cannot just copy the new_ovn into ovn

      -- because if the new_ovn does not match the

      -- value in the database the error message will

      -- be displayed twice. Once from key-delrec and

      -- again when the on-lock trigger fires.)

      --
      if (:<block_name>.new_object_version_number is not null)  an
d
         (l_rec_status = 'QUERY') then
        l_api_ovn := :<block_name>.new_object_version_number;
      else
        l_api_ovn := :<block_name>.object_version_number;
      end if;
      --
      -- Call the api delete routine in validate mode
      --
      hr_grade_api.delete_grade_rate_values
        (p_validate              => true

        ,<parameters>
        ,p_object_version_number => l_api_ovn
        ,p_validate              => true
        );

    end if;
    --
    delete_record;
  end key_delrec_row;
```

Forms Procedure Called from the ON-LOCK Trigger

```
  procedure lock_row is
    l_counter  number;
  begin
    l_counter := 0;
    LOOP
      BEGIN
       l_counter := l_counter + 1;
       --
       -- If this row has just been updated then
       -- the new_object_version_number will be not null.
       -- That commit unit must have been successful for the
       -- on_lock trigger to fire again, so use the
       -- new_object_version_number.
       --
       if :<block_name>.new_object_version_number is not null then
          :<block_name>.object_version_number :=
             :<block_name>.new_object_version_number;
        end if;
        --
        -- Call the table handler api lock routine
        --
        pay_grr_shd.lck
          (<parameters>
          ,p_object_version_number => :<block_name>.object_version
_number
          );
        return;
      EXCEPTION
        When APP_EXCEPTIONS.RECORD_LOCK_EXCEPTION then
          APP_EXCEPTION.Record_Lock_Error(l_counter);
      END;
    end LOOP;
  end lock_row;
```

DataPump

Oracle HRMS Data Pump

This essay provides the information that you need to understand and use the Oracle HRMS Data Pump. To understand this information you should already have a good functional and technical knowledge of the Oracle HRMS product architecture, including:

- The data model for Oracle HRMS and the importance of DateTrack.
- The API strategy and how to call APIs directly.
- How to code PL/SQL. Some PL/SQL code is normally required to convert legacy data for use with Data Pump.
- The HRMS parameters that control the running of concurrent processes (for example, to make the process run in parallel).

Restrictions

This essay does not describe the entire Data Pump schema in detail. Details are given as needed for some of the tables and in most cases you will use the PL/SQL routines to

insert data to these batch interface tables. Full details are provided in the Oracle HRMS electronic Technical Reference Manual (eTRM), available on MetaLink.

Oracle delivers seed data to enable Data Pump API calls to use features such as passing in user values instead of system identifiers. This support is not available for all of the APIs that are delivered with Oracle HRMS. This essay describes a mechanism for calling APIs using Data Pump where the supporting seed data is not present.

For the list of supported APIs, see Publicly Callable Business Process APIs. , *Oracle HRMS Configuring, Reporting, and System Administration Guide*Support for other APIs is planned in future releases.

When purging data from the Data Pump tables, take extra care that you do not delete information on User Keys that you might need for future loading of external data. See: User Key Values, page 2-296.

Contents

This essay includes the following sections:

Overview

Oracle HRMS has a set of predefined APIs that are business process related and you are strongly advised always to use these APIs to load data. The predefined APIs enforce all the business rules in the system and guarantee the integrity of any data loaded into the system.

The Oracle HRMS Data Pump supports rapid implementation by simplifying and standardizing the common tasks associated with loading batch data into the Oracle HRMS tables. This is done by providing a set of predefined batch tables and standard processes that simplify the tasks of data-loading using the supported APIs.

With the Oracle Data Pump you:

1. Map the data items from your external system to the parameter values of the appropriate APIs.

   Because you map data to the parameters of the APIs you do not need to know the complexity of the HRMS data model. For example, to create an employee you need to co-ordinate inserting data into multiple tables. The create_employee API does this automatically, using the parameter values you pass in.

   A special feature of the Data Pump is that you can use user values in place of system IDs for the API parameters. These are translated automatically by the Data Pump.

2. Load your data into a single generic batch lines table. (There is also a single batch header table to help you manage your batch loading processes.)

   The Data Pump works with a single generic batch lines table. It generates a specific view for each API so that you can easily review and update the data for each API using the parameter names for the API.

   Also, there are PL/SQL interface routines to insert your external data into the generic batch lines table.

3. Run a standard process that automatically calls the appropriate API for each line of data in the batch table.

Components of Data Pump

Data Pump consists of the following components:

Meta-Mapper Process

This process generates the specific PL/SQL procedures and views for each of the supported API modules you want to use.

Use the Meta-Mapper to generate a set of views that you can use to examine or update data in the batch tables. For example you might want to correct data or change the order in which data is loaded.

> **Note:** The Meta-Mapper is similar to an install process. You must run the Meta-Mapper before making a data pump API call. Meta-Mapper usually runs during the loading of your software, but there are occasions when you may need to run Meta-Mapper manually. For example, if you cannot find Meta-Mapper, or if you version displays as invalid, then you should run Meta-Mapper manually.

Batch Header Table and Batch Lines Table

Use these two tables to hold the header and lines information from your external data.

- HR_PUMP_BATCH_HEADERS

- HR_PUMP_BATCH_LINES

  Note: The Meta-Mapper creates views based on the batch lines table called HRDPV_<API Procedure Name>, for example, HRDPV_CREATE_EMPLOYEE.

PL/SQL Routines

Use the predefined and generated PL/SQL routines to insert your external or legacy data into the batch lines table. Meta-Mapper generates a separate routine for each API that is supported by the Data Pump.

- HR_PUMP_UTILS.CREATE_BATCH_HEADER(...)
- HRDPP_<API Procedure Name>.INSERT_BATCH_LINES

  For example, HRDPP_ CREATE_EMPLOYEE .INSERT_BATCH_LINES

There is also a help routine to provide detailed information on the parameter options for specific procedures.

- HR_PUMP_META_MAPPER.HELP (<package_name>, <procedure_name>)

The Data Pump Engine Process

The Data Pump Engine process is a standard concurrent process that performs the actual data validation and loading operations. It takes these parameters:

- Batch name
- Processing mode
- Action Parameter Group

Special Features of Data Pump

The following is a list of the special features provided with Data Pump:

User Keys

Data Pump enables you to define the combination of data items that uniquely identify records for loading into Oracle HRMS. For example, when you are loading data for a Person, you could use a combination of Last Name, First Name, Date of Birth, and Gender to identify that person uniquely in Oracle HRMS.

You store these user key definitions in the table HR_PUMP_BATCH_LINES_USER_KEYS.

Use Actual Values

In nearly all cases you can load data using actual names or values without having to identify a system value in Oracle HRMS. The conversion of name to ID is transparent to the user. For example, you can use a real Job Name without needing to identify the JOB_ID in Oracle HRMS; or you can use the value 'Male' for gender without needing to know that the code value is 'M'.

Alternative Meta-Mapper Generation Mode

It is possible to call the Meta-Mapper so that Data Pump API call is essentially a direct call to the API. This feature is most useful in the absence of seed data for Data Pump support.

Automatic Parallel Processing Of Batch Load Process

Data Pump automatically supports parallel processing on multi-processor systems without any extra code. You turn this on by inserting or updating a row for THREADS in the PAY_ACTION_PARAMETER_VALUES table.

This is the same parameter that controls parallel processing for the Payroll Run and other processes in Oracle HRMS.

> **Note:** When you are using parallel processing, use the P_LINK_VALUE parameter in the batch lines to group transactions that must be run within the same thread.

Explicit User Ordering of Operations

When loading batch lines with related data you must perform some operations in a strict sequence. For example, entering salary information for an employee must take place after the employee record has been created.

With Data Pump, you use the P_USER_SEQUENCE parameter to control the order of processing of batch lines.

> **Note:** Data Pump cannot validate the sequence numbers you enter. It accepts the sequence and tries to process as instructed. If you use incorrect numbers the process may return validation errors when it tries to load your data in the wrong sequence. See: Running the Data Pump, page 2-286.

Validation Mode Operation

When you submit the Data Pump concurrent process you can choose to run it in validation mode. This enables you to review errors in batches or in related records in a batch and to change them before any of them are committed to the HRMS database.

Processing Batches

When you run Data Pump the process only loads data that has not already been processed successfully. This means that you can run a batch, review and correct errors for any specific lines, and then rerun the same batch. You can repeat this process until you have successfully loaded all lines in the batch.

To do this you submit the concurrent process with the same batch name. All unprocessed or errored lines are reprocessed automatically.

Logging Options

There are many logging options with Data Pump that help you find errors when running the process.

Using Data Pump

To use Data Pump, follow this sequence of tasks:

1. Decide which of the supported API modules you require for loading your external data and run the Meta-Mapper to generate interface procedures for these APIs.

   See: Running the Meta-Mapper, page 2-276.

2. Use the predefined PL/SQL routines and those created by the Meta-Mapper to transfer your external data into the Data Pump tables.

   See: Loading Data Into the Batch Tables, page 2-283.

   > **Note:** For each entity that requires a User Key you must include the value you want to use as a unique identifier. For example, the parameters P_PERSON_USER_KEY and P_ASSIGNMENT_USER_KEY for create_employee.

3. Optional. Run Data Pump in validation mode to check and correct data before it is loaded.

   See: Running the Data Pump Process, page 2-286.

4. Run Data Pump to load data from batch tables into the Oracle HRMS tables.

   > **Note:** When you load a record for the first time, Data Pump automatically inserts your user key value from the batch lines, and the unique key ID generated by the API into the HR_PUMP_BATCH_LINE_USER_KEYS table. This combination is used for all further data loads that update existing records in Oracle HRMS.
   >
   > For example, P_PERSON_USER_KEY = USER_KEY_VALUE and PERSON_ID = UNIQUE_KEY_ID.

5. Review any errors and correct causes.

   See: Finding and Fixing Errors, page 2-288.

6. If necessary, rerun Data Pump to load corrected batch lines.

   See: Rerunning the Data Pump Process, page 2-291.

   Repeat 5 and 6 until all lines are successfully loaded.

7. Optional. Purge data from the batch tables.

   See: Purging Data, page 2-291.

Running the Meta-Mapper

Based on your implementation you might decide that you do not need to use all of the predefined APIs to load external data. Run the Meta-Mapper for all APIs or for each single API that you select. The Meta-Mapper generates a specific PL/SQL package and view for each API.

> **Note:** For APIs with overloaded interfaces, the Meta-Mapper will only generate code for the latest interface. The latest interface is the interface that has the greatest number of mandatory parameters.

Use the following SQL*PLUS command to generate packages and views for a number of APIs. (Not, however, **all** APIs, as the GENERATEALL name appears to suggest):

```
sql> execute hr_pump_meta_mapper.generateall;
```

See also: APIs Supported by the GENERATEALL Command., page 2-299

Use the following SQL*PLUS command to generate packages and views for one API:

```
sql> execute hr_pump_meta_mapper.generate( <package_name>,<procedure_name>);
```

For example:

```
sql> execute hr_pump_meta_mapper.generate( 'hr_employee_api', 'cr
eate_employee' );
```

The naming convention for the view is hrdpv_<api_module_name> and the naming convention for the PL/SQL package is hrdpp_<api module name>. This applies unless the name would exceed 30 bytes, in which case the name is truncated to 30 bytes. In

the example, the name of the view is hrdpv_create_employee, and the name of the package is hrdpp_create_employee.

You can use the view to insert legacy data into the HRMS schema or the batch tables, or to update data already in the batch lines table. The PL/SQL package contains an insert_batch_lines procedure to make it easy to insert data from your external systems into the batch lines table; and a call procedure that executes the API on the rows in the batch lines table.

View Generated by the Meta-Mapper

For each API the Meta-Mapper generates a view on the HR_PUMP_BATCH_LINES table that reflects the parameters of the API. This makes it easier to examine and update row values. The name of the view reflects the API name. For example, HRDPV_CREATE_EMPLOYEE. For a full listing of this view see: Table and View Descriptions , page 2-298.

In addition to the parameters for the API, the Meta-Mapper always creates the following columns in the view:

```
Column   Description

-------------------------------------------------

BATCH_ID    Foreign key to HR_PUMP_BATCH_HEADERS

BATCH_LINE_ID  Foreign key to HR_PUMP_BATCH_LINES.

Primary key generated using the
     hr_pump_batch_lines_s sequence.

API_MODULE_ID  Foreign key to HR_API_MODULES.
     This tells Data Pump which api to
     call for each row.

LINE_STATUS   Load status of this API:

  'U' - Unprocessed.
       This must be the initial value for all
     lines

  'C' - Complete.
       The API call was successful and the
     changes have been committed.

  'E' - Error.

  'V' - Validated
     The API call was successful but the
     changes have not been committed.

USER_SEQUENCE  Used to control processing order.
     For example, to make sure that address
     for an employee is loaded after the
     employee record has been created.
```

```
LINK_VALUE   Use a unique link_value to link multiple
     rows in a single batch.
     Set this value when using parallel
     processing to make sure that related
      rows in a batch are processed together.

BUSINESS_GROUP_NAME  Alternative business group name to use
                        for a particular API call. If not null,
                        this overrides the value specified in
                        the batch header
```

Meta-Mapper also creates other columns for specific APIs. For example, some of the columns on the create employee view are:

- P_EFFECTIVE_DATE
- P_MANAGER_FLAG
- P_ASSIGNMENT_USER_KEY

Other columns are created to reflect the PL/SQL OUT values returned from the API so that you can examine these values. For example:

- P_NO_MANAGERS_WARNING

You do not need to know which columns of the batch lines table hold specific parameters for the API.

Required Columns

If you use the view to insert data to the batch lines table then remember that in addition to the data required for the insert batch line procedure you also need :

- batch_line_id

  Primary key generated using the hr_pump_batch_lines_s sequence.

- line_status

  Must be set to 'U' (unprocessed).

- api_module_id

  Foreign key to hr_api_modules.

  The following query gets the api_module_id for create employee:

```
SELECT API_MODULE_ID

FROM HR_API_MODULES

WHERE UPPER(MODULE_NAME) = 'CREATE_EMPLOYEE'

AND   UPPER(MODULE_PACKAGE) = 'HR_EMPLOYEE_API';
```

PL/SQL Package Generated by the Meta-Mapper

The Meta-Mapper also generates a separate package for each API to make it easier for you to load data to the batch lines table or to review the content of the table for specific APIs.

For example, the create_employee package hrdpp_create_employee contains two procedures:

- insert_batch_lines

- call

Insert Batch Lines Procedure

Use this procedure to simplify loading data into the batch lines table.

See also: Default and Null Values for API Parameters., page 2-285

A call to this procedure creates one row in the batch lines table, complete with all the parameters. For create employee, some of the parameters are:

p_batch_id	number	in	
p_data_pump_batch_line_id	number	in	default
p_data_pump_business_grp_name	varchar2	in	default
p_user_sequence	number	in	default
p_link_value	number	in	default
p_hire_date	date	in	
p_last_name	varchar2	in	
p_sex	varchar2	in	
p_per_comments	varchar2	in	default
p_date_employee_data_verified	date	in	default
p_date_of_birth	date	in	default
p_email_address	varchar2	in	default
p_employee_number	varchar2	in	
p_expense_check _send_to_addres	varchar2	in	default
p_first_name	varchar2	in	default
p_known_as	varchar2	in	default
p_marital_status	varchar2	in	default
p_middle_names	varchar2	in	default
p_nationality	varchar2	in	default
p_national_identifier	varchar2	in	default
p_previous_last_name	varchar2	in	default
p_registered_disabled_flag	varchar2	in	default
p_title	varchar2	in	default
p_attribute1	varchar2	in	default

```
p_attribute2                    varchar2    in     default
p_attribute3                    varchar2    in     default
p_attribute4                    varchar2    in     default
p_attribute5                    varchar2    in     default
p_attribute6                    varchar2    in     default
p_attribute7                    varchar2    in     default
p_attribute8                    varchar2    in     default
    ...
    ...
p_resume_exists                 varchar2    in     default
p_resume_last_updated           date        in     default
p_second_passport_exists        varchar2    in     default
p_student_status                varchar2    in     default
p_work_schedule                 varchar2    in     default
p_suffix                        varchar2    in     default
p_person_user_key               varchar2    in
p_assignment_user_key           varchar2    in
p_user_person_type              varchar2    in     default
p_vendor_name                   varchar2    in     default
p_correspondence_language       varchar2    in     default
```

This example does not show all the parameters as there are many more.

The optional p_data_pump_business_grp_name parameter specifies a business group name to override the name specified in the batch header.

The optional p_data_pump_batch_line_id parameter specifies the batch_line_id for the inserted row (if necessary an existing row with this batch_line_id will be deleted).

> **Note:** This procedure requires two user key values *p_person_user_key* and *p_assignment_user_key*. You must supply values for these keys. If you use Data Pump to create records in Oracle HRMS then Data Pump automatically inserts your key values and the HRMS key values generated by the APIs into the user keys table. For subsequent actions Data Pump can use these keys to match records from your external system with the Oracle HRMS records. A more detailed explanation and example is included in a later section of this document.

Call Procedure

This is the actual 'wrapper' procedure executed by the Data Pump process to call the API and pass in the appropriate parameter values. The procedure takes two arguments: *p_business_group_id* and *p_batch_line_id*.

> **Note:** Direct calls to this procedure are NOT supported. You must use the Data Pump concurrent process to execute the procedures.

Meta-Mapper Help Procedure

The Meta-Mapper package also includes a help procedure *hr_pump_meta_mapper help* that returns information on the generated PL/SQL package and view names, and the batch lines table parameter values for a given API.

The help procedure has two parameters:

- p_module_package

  The name of API PL/SQL package

- p_module_name

  The name of API PL/SQL procedure

You must set server output on before calling this procedure.

For example, use the following SQL*PLUS to get help for *hr_employee_api.create_employee:*

```
sql> set serveroutput on size 1000000;
sql> execute hr_pump_meta_mapper.help( 'hr_employee_api', 'create_
employee' );
```

The output is as follows:

```
Generated package: hrdpp_create_employee

Generated view: hrdpv_create_employee

Parameter Name              Type    In/Out Default? Lookup Type

--------------              -----   ------ -------- -----------

P_HIRE_DATE                 DATE      IN

P_LAST_NAME                 VARCHAR2  IN

P_SEX                       LOOKUP    IN   SEX

P_PER_COMMENTS              VARCHAR2  IN   DEFAULT

P_DATE_EMPLOYEE

  _DATA_VERIFIED            DATE      IN   DEFAULT

P_DATE_OF_BIRTH             DATE      IN   DEFAULT

P_EMAIL_ADDRESS             VARCHAR2  IN   DEFAULT

P_EMPLOYEE_NUMBER           VARCHAR2  IN
```

```
P_EXPENSE_CHECK
  _SEND_TO_ADDRES           LOOKUP    IN   DEFAULT  HOME_OFFICE

P_FIRST_NAME                VARCHAR2  IN   DEFAULT

P_KNOWN_AS                  VARCHAR2  IN   DEFAULT

P_MARITAL_STATUS            LOOKUP    IN   DEFAULT  MAR_STATUS

P_MIDDLE_NAMES              VARCHAR2  IN   DEFAULT

P_NATIONALITY               LOOKUP    IN   DEFAULT  NATIONALITY


P_NATIONAL_IDENTIFIER       VARCHAR2  IN   DEFAULT

P_PREVIOUS_LAST_NAME        VARCHAR2  IN   DEFAULT

P_REGISTERED_DISABLED_FLAG  LOOKUP    IN   DEFAULT  YES_NO

P_TITLE                     LOOKUP    IN   DEFAULT  TITLE

P_WORK_TELEPHONE            VARCHAR2  IN   DEFAULT

P_ATTRIBUTE_CATEGORY        VARCHAR2  IN   DEFAULT

P_ATTRIBUTE1                VARCHAR2  IN   DEFAULT

P_ATTRIBUTE2                VARCHAR2  IN   DEFAULT

P_ATTRIBUTE3                VARCHAR2  IN   DEFAULT

P_ATTRIBUTE4                VARCHAR2  IN   DEFAULT

P_ATTRIBUTE5                VARCHAR2  IN   DEFAULT

P_ATTRIBUTE6                VARCHAR2  IN   DEFAULT

...

P_ASSIGNMENT_SEQUENCE       NUMBER    OUT

P_ASSIGNMENT_NUMBER         VARCHAR2  OUT

P_NAME_COMBINATION_WARNING  BOOLEAN   OUT

P_ASSIGN_PAYROLL_WARNING    BOOLEAN   OUT

P_USER_PERSON_TYPE          VARCHAR2  IN   DEFAULT

P_VENDOR_NAME               VARCHAR2  IN   DEFAULT

P_CORRESPONDENCE_LANGUAGE   VARCHAR2  IN   DEFAULT

...
```

The following is an explanation of the help output:

- In the above example, the insert_batch_lines procedure is: hrdpp_create_employee.insert_batch_lines.
- The Parameter Name column shows the name of the parameter as it appears in the insert_batch_lines procedure and generated view.
- A parameter can have type USER_KEY which means that it is a user key (see the section User Key Values, page 2-296 for more details). For example, P_SUPERVISOR_USER_KEY USER_KEY IN DEFAULT. User key parameters are implicitly of type VARCHAR2.
- DATE parameter values are passed to the insert_batch_lines procedure as VARCHAR2 strings in YYYY/MM/DD format.

  Note: The date format used by Data Pump is YYYY/MM/DD. For dates in Oracle HRMS, the internal date format is YYYY/MM/DD HH24:MM:SS.

- BOOLEAN parameter values are passed to the insert_batch_lines procedure as VARCHAR2 strings with the values TRUE or FALSE'.
- The In/Out column has the value IN for parameters that are PL/SQL IN or IN/OUT when passed to the API, or are user key parameters. If the parameter is an API PL/SQL OUT parameter, then the In/Out column value is OUT.
- Only IN parameters are arguments to the insert_batch_lines procedure. OUT parameters appear in the generated view.
- The Default column has the value DEFAULT if the parameter's value is not required in the batch lines table. For mandatory parameters this column is empty.
- Mandatory parameter values must be passed to the insert_batch_lines procedure.
- If the parameter is a lookup parameter, the Lookup Type column contains the name of the parameter's lookup type.

Loading Data Into the Batch Tables

The Meta-Mapper generates a specific PL/SQL package and view for each API. Use these PL/SQL interface procedures and views for loading data into the batch tables, except where stated otherwise in this document.

It is particularly important that inserts are performed exclusively through the interfaces. There are two reasons for this:

- Using the PL/SQL procedure insulates you from the complexities of the underlying schema.
- Using the PL/SQL procedure insulates you from any schema changes that might be made in any future release. This is important if you intend to use Data Pump on a continuing basis.

  Tip: Test the validity of the legacy data capture code on a subset of the batch to be loaded. For example, if you plan to load details for 100000 people, test your routines to validate and load a subset of 100 representative people. This should help you to identify and resolve any obvious problems with your capture code before you attempt to load the bulk of your data.

The Batch Interface Tables

The main objective of the interface design was to keep everything as simple as possible. The result is that Data Pump only has one batch header and one batch lines table for loading data for all APIs. Views are generated by the Meta-Mapper with specific column names for each API.

Each row of the batch lines table holds the reference to an API and data values. Data Pump executes each API with the data passed in as parameters.

How to Control Processing Order

There are many instances where you need to control the order in which batch lines are loaded into the database. For example, Data Pump would generate an error if it tried to create an address for a person before it created the person.

To control the order in which operations are performed, use the *p_user_sequence* parameter to set the order manually. Choose some appropriate numeric values for this parameter when you insert the data to the batch lines table. Data Pump uses these numbers to determine processing order.

Different Approaches to Batch Loading

There are a number of approaches you can take when setting the order for processing batch lines.

One approach would be to load disparate data in separate batches. For example load personal information in one batch and address information in a second batch.

Another approach would be to create a batch containing lines with related API calls. For example, you could load person, address, and assignment information for one employee as part of one batch. In this approach, if you are using the parallel processing option, you would use the *p_link_value* parameter to make sure all the lines are processed in the same chunk. Use the default or *p_user_sequence* parameter to make sure that the different API calls are made in the correct order within the linked group.

Processing Order When Running Parallel

The Data Pump process has been optimized to take advantage of parallel processing options. If you want to run a multi-threaded process there are some special considerations for ordering batch lines.

When you run the Data Pump process in parallel, the concurrent manager generates multiple threads, each of which processes a defined number of batch lines before it commits them to the database. The number of lines is controlled by the CHUNK_SIZE payroll action parameter - see Other Parameters, page 2-287 for details.

With parallel processing and chunking of lines, in theory a transaction that includes more than one line could be split between processes. This would mean that lines might not be processed in the order set by the *p_user_sequence* parameter.

You can prevent this by using the *p_link_value* parameter. This parameter tells Data Pump that a set of batch lines must be processed in the same chunk. Use the same link value for all the lines that must be processed by the same thread - this will automatically extend the number of rows processed by a single thread when necessary.

When lines have a common link value, they must also be in consecutive user sequence in order to be processed within a single chunk.

For example, in the following table, only the lines with the user sequences 1, 2 and 5 are guaranteed to be processed in the same thread.

User Sequence	Link Value
1	1
2	1
5	1
8	2
10	1

Note: When running Data Pump in parallel you may find that performance does not scale as expected. Remember that running business process APIs in parallel may cause lock contention because of extended validation. For example, in the past, the personal payment method and element entry APIs were known to have problems in this area.

Default and Null Values for API Parameters

Specifying a Default or NULL Parameter Value

Part of the design for the APIs in Oracle HRMS is that many parameters have default values set for them. This means that they can be called directly without having to pass values for all parameters.

When you use Data Pump there is a similar mechanism that means you do not have to supply values for all parameters.

The following rules apply:

- If an insert batch lines parameter is passed NULL or is not passed a value and can be defaulted, the appropriate default value will be passed to the API module itself.
- If you want to set up an explicit NULL value for a parameter, use the special reserved string <NULL>. You may want to do this to update to a null value.

Any other value passed as a parameter will be the value inserted into the batch line and subsequently passed to the appropriate API process.

Indicator Parameters

The insert_batch_lines procedure may be generated with indicator parameters. Each indicator parameter is generated in addition to the corresponding standard parameter e.g. I_AMOUNT (indicator parameter), P_AMOUNT (standard parameter). The indicator parameters are generated to allow the special value NULL to be specified for non-mandatory number and date parameters whose default value is not NULL. If the indicator parameter = Y then the value NULL is written to the batch lines table, otherwise the standard parameter's value is used. The usual case for this is for update APIs where a number or date value needs to be updated to NULL

Assumed Default Values

Occasionally, when the value NULL is used to specify a non-mandatory parameter, the wrong default value gets passed to the API call. The usual reason for this is that the parameter in question has a non-standard default value, but the seed data has not taken

this into account. In such case, the correct default value for the parameter should be explicitly set in the batch lines row for the Data Pump API call.

The meta-mapper assumes, that unless seeded otherwise, certain default values for API parameters - this is because it is not possible to get the actual default values from the database. The default value used for a create API (e.g. create_employee) is NULL. For all other APIs, the default values used are shown in the following table:

Parameter Type	Default Value
BOOLEAN	NULL
DATE	HR_APLG_DATE
LONG	NULL
NUMBER	HR_APLG_NUMBER
VARCHAR2	HR_APLG_VARCHAR2

Default and Null Values for Mapped Parameters

A mapped parameter is one where an actual value (or a user key) is used rather than a system identifier in the Data Pump API call. The meta-mapper call procedure calls a mapping function before making the API call to resolve the system identifier value from the input value. Such a mapping function will usually have two or more parameters – an obvious name parameter e.g. P_JOB_NAME, and other parameters such as P_EFFECTIVE_DATE.

If one or more of the mapping function parameters is set to <NULL> in batch lines then the mapped parameter is passed to the API as NULL. Otherwise, if one or more of the mapping function parameters is set to NULL in batch lines and the default value is NULL or an HR_API value (e.g. HR_API.G_NUMBER) then the mapped parameter is passed to the API with its default value.

Recommendation: To use this feature, set the name parameter to <NULL> or NULL in the batch lines table. There is no need to worry about what the other mapping function parameters could be.

Running the Data Pump Process

Use the Submit Reports and Processes form to start the Data Pump Engine process. It takes these parameters:

- BATCH NAME

  The batch_name is one of the batches inserted via the create_batch_header procedure.

- VALIDATE FLAG

  Default value for this flag is No. This commits all valid lines to the database.

  If the validate flag is set to Yes, the process runs in validation mode. The APIs are called, but their results are rolled back. Use this mode to check and correct data before committing changes to the database.

- ACTION PARAMETER GROUP

The action parameter group specifies the set of action parameter values to configure this Data Pump run.

The default value for this parameter is set from the HR: Data Pump Action Parameter Group profile option.

> **Note:** Before running the Data Pump process you should decide whether to use parallel threads and whether you want to turn on any logging options.

Overview of Data Pump Action Parameters

Data Pump process running can be controlled through the action parameter value settings. A number of these action parameters (THREADS, CHUNK_SIZE, MAX_ERRORS_ALLOWED) are also used by the other processes e.g. the payroll run.

With action parameter groups it is possible to have separate action parameter values for different processes, something that is highly recommended. Another use of action parameter groups is to switch in an action parameter group for debugging e.g. so that Data Pump is run as a single thread with logging switched on.

Any action parameters not set within the specified action parameter group take their values from the default action parameter group (the null action parameter group). Furthermore, if action parameters are NULL then the Data Pump process uses default values for them.

You can set action parameter values from the Action Parameters form (navigate to Process And Reports->Action Parameters).

Running In Parallel

To enable parallel processing you set a value for the THREADS parameter in PAY_ACTION_PARAMETER_VALUES.

The threads value includes the starting process. That means that if you set a value of 2, the main engine code starts with one slave process to make a total of two concurrent processes. When running in parallel, the 'master' process may finish before the slave processes. This is normal.

> **Note:** The THREADS parameter also controls the parallel execution of the other Oracle Payroll processes. We recommend that you use action parameter groups to separate action parameters for Data Pump from normal payroll processing.

Other Parameters

There are six other payroll action parameters you can set for Data Pump.

CHUNK_SIZE

Default = 10

Controls how many batch API calls are processed at a time per thread when running in parallel. It also controls the number of API calls per commit. Note that there are certain circumstances under which the actual number can vary from this number. For example, it can be higher if the p_link_value parameter is set.

MAX_ERRORS_ALLOWED

Default = 20

Controls how many errors in calling an API will be tolerated before the entire Data Pump engine fails. This is the number of errors per parallel thread.

PUMP_DEBUG_LEVEL

Use this parameter to turn on logging for tracking errors generated by the Data Pump process. For a list of valid values for this parameter, see Logging Options, page 2-288.

DATA_PUMP_DISABLE_CONTINUOUS_CALC

Default = N

Use this parameter to turn off continuous calculation triggers. This may be desirable for performance reasons. The value Y turns off the continuous calculation triggers.

DATA_PUMP_NO_FND_AUDIT

Default = N

Use this parameter to turn off Oracle Applications auditing. This may be desirable for performance reasons. The value Y turns off the auditing.

DATA_PUMP_NO_LOOKUP_CHECKS

Default = N

Use this parameter to turn off lookup validation in the Data Pump API call. The Data Pump API call assumes that values for lookup parameters are passed in as lookup codes only. This may be desirable for performance reasons. The value Y turns off the lookup validation.

Checking Run Status

The Data Pump runs as a concurrent process. You can check process status at any time using the View Concurrent Requests window. The concurrent manager only reports failure if the entire process has failed. Usually this happens because the number of errors exceeded the value set by the MAX_ERRORS_ALLOWED parameter.

> **Note:** Even if the concurrent process completes successfully there may be some data errors encountered by the process. You should always check for batch line errors.

Finding and Fixing Errors

This section deals with the logging options available for tracking errors generated by the Data Pump process, as well as hints and tips on how to deal with these.

Logging Options

You enable logging options for Data Pump by inserting appropriate values in the PAY_ACTION_PARAMETERS_VALUES table for the PUMP_DEBUG_LEVEL parameter.

> **Note:** Turning logging on always affects the overall performance of the data pump process. You should only use logging to help track down problems when they occur. Remember also to switch logging off after you have solved your problem.

Valid values for PUMP_DEBUG_LEVEL are as follows.

> **Tip:** The first three options are likely to be the most useful to you.

Option	Description
AMD	API Module Debug (enables trace output from API)
RRP	Range Row Processing logging (logs the number of errors that occurred for each unit of work, or range)
GID	Get_id function failure information (logs failures in functions that map user values to IDs)
MSG	Output specific logging messages
ROU	Routing information (entry to and exit from procedures)
WCD	Wrapper cache debug logging
STK	Stack dump logging (trace information on failure)
EXT	Exit information (trace information on success)
RRI	Range row insert logging
BLI	Batch Line Information (output the batch line number for the batch line being processed).
CLF	Concurrent Log File (logging messages output with the MSG option go to the concurrent manager log file).

You can combine any number of these options by concatenating the values, separated by a colon. For example, the string 'MSG:RRI:RRP' combines MSG, RRI, and RRP debugging.

How to View Logging Output

When you enable logging options, output is produced for every thread that may be running. Use the PYUPIP command to view this output.

To use this command you will need to know the ID for the concurrent process you are logging. Online you can use the View My Requests window to find the Concurrent Request IDs. Alternatively, you can query from the HR_PUMP_REQUESTS table. One row is inserted for each process that is running. For example:

```
select * from hr_pump_requests;
```

Typical output would be:

```
BATCH_ID    REQUEST_ID      PROCESS_TYPE

----------- -------------- ----------------

8437        98533          MASTER

8437        98534          SLAVE
```

This tells us that there are two processes running, and the request_id values are 98533 and 98534.

Use PYUPIP to trace the output in a separate command line window. For example:

```
PYUPIP <user/password>@database REQID98533

PYUPIP <user/password>@database REQID98534
```

> **Note:** If you are running multiple threads, you should trace all the threads. If you do not choose all threads, this means that the processing comes to halt when the database trace pipe fills up. It may be advisable to run a single thread only when tracing.

How to Find Errors in Batch Lines

When an error occurs during processing, Data Pump generates one or more rows in the HR_PUMP_BATCH_EXCEPTIONS table. There will be multiple rows if the API supports multiple messaging. In this release you must use SQL*PLUS to view this information.

Additionally, you can use SQL*PLUS to query rows in HR_PUMP_BATCH_LINES where the LINE_STATUS has a value of E - error.

> **Note:** In validation mode LINE_STATUS is set to V- validated, for a successful API call. In update mode LINE_STATUS is set to to C - complete, for a successful API call.

Investigating the Cause of Errors

Investigation strategies depend on the type of error and the indications of its origin. For some errors you may need experience with the use of APIs and the Oracle HRMS application to recognize what might be wrong.

Some specific advice for Data Pump follows:

- Start with the columns of the HR_PUMP_BATCH_EXCEPTIONS table to identify which batch line has caused the error. Use this to check the parameters and values of the batch line itself.
- One common error is 'no data found'. This is most likely to happen because of an error in one of the functions called to convert user meaning to ID values. In this case, the exact cause of the error will not be obvious from looking in the exceptions table. More information can be gained from using the GID logging value. When failure occurs, the name of the function that failed, plus the argument values passed in, is displayed in the trace.
- The AMD logging value can be used to help track down problems. It activates the logging in the API modules themselves - providing copious output to examine.

- Another common cause of errors is incorrect ordering of the data load. For instance, attempting to load a person's address before the person. An associated error may occur if you are using parallel processing and do not use LINK_VALUE to associate multiple batch lines.
- When running in validation mode, ordering errors will occur if the batch is not split up into chunks that are independent of the results of other chunks. This will occur even if the validation is done with a single thread. The reason is that the results of APIs over a single chunk are rolled back to release rollback segments. This is another reason to use the *p_link_value* parameter to control the running of a load.

How to Fix Errors

The most common cause of errors is likely to be that incorrect values have been loaded via the insert_batch_lines procedure and that these need to be corrected.

Using The Views To Correct Data

Use the HRDPV_ views on HR_PUMP_BATCH_LINES to correct values in the appropriate columns. You can use normal update statements on these views and this makes fixing data problems much simpler.

> **Warning:** When using the views to make changes to problem data, you must not alter the LINE_STATUS on the HR_PUMP_BATCH_LINES table. The Data Pump engine uses this for processing.

> **Note:** Views on HR_PUMP_BATCH_LINES display rows only for the APIs for which they were generated. Any attempt to update the API_MODULE_ID column with an incorrect value will fail with an ORA-1402 error. The views are generated with a WITH CHECK OPTION on the where-clause to prevent you from using a view to generate any row that the view could not select.
>
> (The same warning applies to inserting rows into HR_PUMP_BATCH_LINES using the generated views.)

Rerunning The Data Pump Process

After you have fixed any problems you can rerun the batch by submitting the Data Pump process again using the same batch name. You can submit the process any number of times until all lines are successfully completed. Batch lines with a status of E - error; U- unprocessed; or V -validated are automatically reprocessed.

You do not have to take any action to remove rows from the exception table. Data Pump automatically deals with this.

Lines validated in previous Data Pump runs are reprocessed even if the Data Pump is run in validation mode because the results of the associated API calls would have been rolled back in the previous runs. Only lines with a status of C -complete are not reprocessed.

Purging Data

Currently there is no purge process provided with Data Pump to remove data automatically from batch tables, other than the automatic removal of rows in the exception tables. In all other instances, you should consider what data needs to be purged and when.

> **Important:** You should take extra care when purging any data from the user key values table. For example, deleting assignment and

person user keys would mean that you could not create a secondary assignment for that employee unless you first use the add_user_key procedure to recreate the purged user keys. We therefore recommend that the USER_KEYS table is only purged when Data Pump processing has been completed.

How To Purge

In all cases you should start with the following actions:

```
TRUNCATE TABLE HR_PUMP_REQUESTS;

TRUNCATE TABLE HR_PUMP_RANGES;
```

Simple Purge Of All Rows

If you want to purge all rows regardless of status then use the following:

```
TRUNCATE TABLE HR_PUMP_BATCH_EXCEPTIONS;

TRUNCATE TABLE HR_PUMP_BATCH_LINE_USER_KEYS;

TRUNCATE TABLE HR_PUMP_BATCH_LINES;

TRUNCATE TABLE HR_PUMP_BATCH_HEADERS;
```

Purge Of All Successful Rows

This is more complicated. You should purge data only when all loads have been successful. This avoids the danger of purging rows that are still needed. Perform the following actions:

- Use the HR_PUMP_BATCH_LINES.LINE_STATUS column to tell which rows have been successful, and therefore can be purged.
  - Look for a status of C. Of course, if all rows in a batch have status C then simply purge all rows in that batch.
- Remove all appropriate rows in the following tables, in the order shown below:
  - HR_PUMP_BATCH_EXCEPTIONS
  - HR_PUMP_BATCH_LINE_USER_KEYS
  - HR_PUMP_BATCH_LINES

  If all rows in HR_PUMP_BATCH_LINES have been deleted, remove the appropriate batch from the HR_PUMP_BATCH_HEADER table.

Sample Code

This section contains some sample code showing how you could call the batch lines procedures.

This example is artificial in that the data for the API calls is generated. However, it shows how we can prepare the Data Pump to create a number of batch lines that:

- Create an employee
- Create an address for the employee
- Update the default assignment criteria
- Create a secondary assignment

The example also illustrates the use of *p_link_value* to make sure that the separate transactions for each employee and assignment are processed by the same thread.

```
----------------------- start of example ----------------------
create or replace package hrdp_cre_emp as
procedure hrdp_cre_emp (p_start in number, p_end in number);
end hrdp_cre_emp;
/
create or replace package body hrdp_cre_emp as
/*
 *  Insert a number of batch lines in preparation for
 *  running the data pump engine, which will then
 *  - create an employee
 *  - create an address for the employee
 *  - update the criteria of the default assignment
 *  - create a secondary assignment
 */
procedure hrdp_cre_emp (p_start in number, p_end in number) is
   l_last_name    varchar2(40);
   l_hire_date    date;
   l_birthday     date;
   l_first_name   varchar2(40);
   l_asgno        varchar2(40);
   -- These are the 'out' values.
   l_special_ceiling_step_id     number;
   l_person_user_key             varchar2(100);
   l_address_user_key            varchar2(100);
   l_assignment_user_key         varchar2(100);
   l_assignment_user_key2        varchar2(100);
   l_link_value                  number;
   l_commit_count number;
   l_commit_limit number;
   l_emp_count    number;
   l_address_line1 varchar2(256);
begin
   l_commit_limit := 10;   -- commit after every 10 employees.
   l_commit_count := 0;
   l_first_name   := 'David';
   l_hire_date    := to_date('1997/12/01', 'YYYY/MM/DD');
   l_birthday   := to_date('1970/01/01', 'YYYY/MM/DD');
   l_link_value := 0;
   for emp_count in p_start..p_end loop
      -- Prepare to create an employee.
     l_last_name := 'DUMP' || lpad(emp_count, 5, '0');
     l_person_user_key     := l_last_name || ' : PER USER KEY';
     l_assignment_user_key := l_last_name || ' : ASG USER KEY';
     l_address_user_key := l_last_name || ' : ADDR USER KEY';
     l_address_line1 := to_char(emp_count) || ', Union Square';
     hr_utility.trace('Last Name : ' || l_last_name);
     -- Allow linking together so that these API calls process
     -- by the same thread.
     l_link_value := l_link_value + 1;
     hrdpp_create_employee.insert_batch_lines
     (
         p_batch_id             => 3,
         p_user_sequence        => null,
         p_link_value           => l_link_value,
         p_person_user_key      => l_person_user_key,
```

```
         p_assignment_user_key  => l_assignment_user_key,
         p_hire_date            => l_hire_date,
         p_last_name            => l_last_name,
         p_sex                  => 'Male',
         p_employee_number      => null,
         p_per_comments         => 'Comments for : ' || l_last_nam
e,
         p_date_of_birth        => l_birthday,
         p_email_address        => 'somebody@us.oracle.com',
         p_first_name           => l_first_name,
         p_user_person_type     => 'Employee'
      );
      -- Create an address for the person.
      hrdpp_create_us_person_address.insert_batch_lines
      (
         p_batch_id             => 3,
         p_user_sequence        => null,
         p_link_value           => l_link_value,
         p_effective_date       => l_hire_date,
         p_primary_flag         => 'Yes',
         p_date_from            => l_hire_date,
         p_address_type         => 'Home',
         p_address_line1        => l_address_line1,
         p_city                 => 'Golden Valley',
         p_county               => 'Los Angeles',
         p_state                => 'California',
         p_zip_code             => '91350',
         p_country              => 'US',
         p_person_user_key      => l_person_user_key,
         p_address_user_key     => l_address_user_key
      );
      -- Let's update some criteria.
      l_special_ceiling_step_id := hr_api.g_number;
      hrdpp_update_emp_asg_criteria.insert_batch_lines
      (
         p_batch_id                    => 3,
         p_user_sequence               => null,
         p_link_value                  => l_link_value,
         p_effective_date              => l_hire_date,
         p_datetrack_update_mode       => 'CORRECTION',
         p_assignment_user_key         => l_assignment_user_key,
         p_payroll_name                => 'Monthly',
         p_special_ceiling_step_id     => l_special_ceiling_step_
id
      );
      l_assignment_user_key2 := l_assignment_user_key || '2';
      hrdpp_create_secondary_emp_asg.insert_batch_lines
      (
         p_batch_id                 => 3,
         p_user_sequence            => null,
         p_link_value               => l_link_value,
         p_assignment_user_key      => l_assignment_user_key2,
         p_person_user_key          => l_person_user_key,
         p_effective_date           => l_hire_date,
         p_assignment_number        => l_asgno,
         p_comments                 => 'asg created by data pump'
,
         p_organization_name        => 'Setup Business Group',
```

```
         p_grade_name                => 'faz1',
         p_job_name                  => 'TEST',
         p_payroll_name              => 'Monthly'
      );
      l_hire_date    := l_hire_date + 1;
      l_commit_count := l_commit_count + 1;
      if(l_commit_count = l_commit_limit) then
         -- Commit after so many employees.
         hr_utility.trace('Commit after ' || l_commit_limit || ' e
mployees.');
         commit;
         l_commit_limit := 1;
      end if;
   end loop;
end hrdp_cre_emp;
/
```

Notes on Using The Generated Interfaces

The Meta-Mapper process generates a view and PL/SQL packages for each API. This section explains some of the factors that you should keep in mind when using them.

Finding System IDs from Names or Values

When you use APIs you must supply lookup codes and surrogate primary keys for many parameters. For example:

```
...

p_sex        => 'M',

p_payroll_id => 13456,

...
```

Without Data Pump you would need to write additional code to convert values from your external system to Oracle HRMS system IDs for each API.

However, with Data Pump you have a set of predefined procedures for each of the supported APIs that automatically convert user names or values into lookups and system IDs. For example:

```
...

p_sex          => 'Male',

p_payroll_name => 'Monthly Payroll',

...
```

> **Note:** For lookup parameters, you can use the meaning or the lookup code itself. For non-lookup type IDs you will find an alternative parameter to use.

Exceptions

There are three major exceptions to the use of names for parameter values:

- Flexfield Attribute Parameters

- PL/SQL IN/OUT Parameters
- Legislation Specific Lookup Parameters

Flexfield Attribute Parameters

Most of the API processes include flexfield attribute parameters with names like P_SEGMENT18 or P_ATTRIBUTE20. Data Pump cannot know what the mappings of these values are in your specific implementation and therefore value conversion is not supported.

This means that you must take responsibility for passing the correct lookup code or other value as appropriate.

PL/SQL IN/OUT Parameters

When an API performs a combination of different actions then you need to provide the appropriate ID or code values for the parameters rather than the user meanings. This should not be a great problem where the values for these items can be derived before the Data Pump run.

For example, in hr_assignment_api.update_emp_asg , p_special_ceiling_step_id must be passed in as an ID, even though other APIs require it to be a user key.

> **Note:** You cannot provide user keys for PL/SQL IN/OUT parameters of the API because the Data Pump code that calls the specific API has no way to determine whether the user key existed before the API call and therefore whether it is to be created or its ID value updated after the API call.

Many APIs generate a comment_id as an output parameter. However, you are not required to supply a user key value for the comment_id. This avoids the generation of a lot of meaningless user keys.

> **Note:** A comment_id user key is required for the comment_id parameters to the element entry creation and update APIs. You must add these user keys if you require them for the element entry API calls.

Legislation Specific Lookup Parameters

A similar situation arises with legislation-specific business process API calls where a specific lookup in the legislation-specific API call corresponds to a generic parameter in the generic business process API call.

For example, the *p_region_1* parameter in the hr_person_address_api. create_person_address API corresponds to *p_county lookup* parameter in the hr_person_address_api.create_gb_person_address API.

When calling hr_person_address_api.create_person_address for a GB address via Data Pump, you would have to pass the 'GB_COUNTY' lookup code for the *p_region_1* parameter. Alternatively you could use the 'GB_COUNTY' lookup meaning if you used hr_person_address_api.create_gb_person_address.

> **Note:** You should use legislation-specific APIs where these are available.

User Key Values

When you are mapping data from your external system to Oracle HRMS you will find that there are some cases where an ID value for an Oracle entity cannot be derived from a logical unique key or name. Examples of this are Person, Assignment

and Address. Consider the unique identifier for a person. It is very difficult, if not impossible, to identify a person uniquely. In theory different people may share the same first and last names, gender, birth date, marital status, and so forth.

There are similar problems if an entity does not have a logical key, and its surrogate ID cannot be derived easily from the names of any of its component entities. For example, it isn't easy to identify a unique Element Link by looking simply at names of its components - Payroll, Job, Position etc.

Or, the entity may be an abstract entity specific to the Oracle Applications products and is only identifiable using an ID value. For example an ID_FLEX_NUM.

The solution provided by Data Pump is to enable you to set a 'User Key' value. This value must be a unique character string. It could be a unique ID taken from your external system or it could be a concatenation of multiple values. For example a user key for a person could be the person's name concatenated with the existing employee number from your legacy system. An illustration would be:

```
p_person_user_key => 'Joe Bloggs' || '2345',  -- name + emp no
```

You must define user key values for any parameters with a name that ends 'user_key'. Data Pump uses these user key values to identify IDs for the records in the Oracle HRMS system.

> **Note:** User key values must be unique across all entities. For example, it is not possible to have a Person user key value of 'SMITH1001', and an Assignment user key value also of 'SMITH1001'.

In most cases you will have one user key value for each system ID. However, with Data Pump you can define many different user keys for the same system ID. This is important if you are loading data from different external systems and the unique keys do not match.

User keys are held as rows in the HR_PUMP_BATCH_LINE_USER_KEYS table.

Creating User Key Values

User keys are created in one of two ways:

- Data Pump inserts new user keys

  Using Data Pump you must specify user keys for several API parameters. After a successful call to an API that creates a new record, Data Pump inserts a new row in the user keys table with the name you specified and the system ID value returned from the API. The returned ID value is a PL/SQL OUT parameter to the API.

- Manually insert a new user key

  If you have already loaded data from an external system, or you want to create multiple user keys for the same system ID you can manually insert rows into HR_PUMP_BATCH_LINE_USER_KEYS using the *add_user_key* utility procedure.

Once the user keys have been created you can use the same key with other APIs to update an existing entity, or to specify another entity. For example, two person user keys can be used to specify a contact relationship.

Utility Procedures Available With Data Pump

This section lists the utility procedures that are provided with the Data Pump.

All the procedures are in the HR_PUMP_UTILS package.

create_batch_header

```
Parameters :
    p_batch_name            : unique batch name.
    p_business_group_name   : name of business group (optional)
    p_reference             : user reference value (optional)
Returns
    The hr_pump_batch_headers.batch_id.
Description :
    Creates a batch header row.  This should be used to create
    the row rather than direct insert.
```

An example of a call to this procedure is:

```
  declare
     l_batch_id number;
  begin
     l_batch_id := hr_pump_utils.create_batch_header
                 ('Employees for Dept 071', 'AKA Enterprises');
  end;
```

add_user_key

```
Procedure  : add_user_key
Parameters :
    p_user_key_value        : unique user key value.
    p_unique_key_id         : ID associated with the user key.
Description :
  Creates a user key for use with Data Pump API calls.
  add_user_key is used to add a user key when the object
  referred to by the ID value has not been created by Data
  Pump. This may happen when the object has no creation API but
  is required as a user key parameter to an API called by Data
  Pump, or if the object was created before Data Pump was
  available.
```

modify_user_key

```
Procedure  : modify_user_key
Parameters :
    p_user_key_value        : unique user key value identifying
                              the user key to be changed.
    p_new_user_key_value    : new unique user key value.
    p_unique_key_id         : new ID associated with the user
                              key.
Description :
    The main purpose of modify_user_key is to fix an incorrect
    user key created by add_user_key. If either
    p_new_user_key_value or p_unique_key_id are null then the
    corresponding column is not updated for the user key.
```

Table and View Descriptions

The following section provides more detailed descriptions of the specific tables and views you use with Data Pump.

APIs Supported by the GENERATEALL Command

Package Name	Business Process
HR_APPLICANT_API	CREATE_APPLICANT
	CREATE_GB_APPLICANT
	CREATE_US_APPLICANT
HR_ASSIGNMENT_API	ACTIVATE_EMP_ASG
	ACTUAL_TERMINATION_EMP_ASG
	CREATE_SECONDARY_EMP_ASG
	CREATE_GB_SECONDARY_EMP_ASG
	CREATE_US_SECONDARY_EMP_ASG
	SUSPEND_EMP_ASG
	UPDATE_EMP_ASG
	UPDATE_EMP_ASG_CRITERIA
	UPDATE_GB_EMP_ASG
	UPDATE_US_EMP_ASG
HR_CONTACT_API	CREATE_PERSON
HR_CONTACT_REL_ API	CREATE_CONTACT
HR_EMPLOYEE_API	CREATE_EMPLOYEE
	CREATE_GB_EMPLOYEE
	CREATE_US_EMPLOYEE
HR_EX_ EMPLOYEE_API	ACTUAL_TERMINATION_EMP
	FINAL_PROCESS_EMP
HR_JOB_API	CREATE_JOB
HR_JOB_REQUIREMENT_API	CREATE_JOB_REQUIREMENT
HR_PERSONAL_PAY_METHOD_API	CREATE_GB_PERSONAL_PAY_METHOD
	CREATE_PERSONAL_PAY_METHOD
	CREATE_US_PERSONAL_PAY_METHOD
	DELETE_PERSONAL_PAY_METHOD

Package Name	Business Process
	UPDATE_PERSONAL_PAY_METHOD
	UPDATE_GB_ PERSONAL_PAY_METHOD
	UPDATE_US_ PERSONAL_PAY_METHOD
HR_PERSON_ADDRESS_API	CREATE_GB_PERSON_ADDRESS
	CREATE_PERSON_ADDRESS
	CREATE_US_PERSON_ADDRESS
	UPDATE_PERSON_ADDRESS
	UPDATE_GB_PERSON_ADDRESS
	UPDATE_US_PERSON_ADDRESS
HR_PERSON_API	UPDATE_PERSON
	UPDATE_GB_PERSON
	UPDATE_US_PERSON
HR_POSITION_API	CREATE_POSITION
	UPDATE_POSITION
HR_POSITION_REQUIREMENT_API	CREATE_POSITION_REQUIREMENT
HR_SIT_API	CREATE_SIT
HR_VALID_GRADE_API	CREATE_VALID_GRADE
PY_ELEMENT_ENTRY_API	CREATE_ELEMENT_ENTRY
	DELETE_ELEMENT_ENTRY
	UPDATE_ELEMENT_ENTRY

Using Data Pump with Unsupported APIs

Sometimes the necessary seed data for a Data Pump call to a particular API is not present. The usual problem when running the meta-mapper generate is the lack of mapping functions to resolve system identifiers from user values, for example:

ORA-2001: Seed data error: Mapping function get_set_of_books_id does not exist. Please contact your support representative.

This type of error is usually caused by API parameters with names ending in _ID, for example, P_JOB_ID.

You can call the meta-mapper in an alternative generate mode that essentially generates a direct call to the API rather than processing parameter values beforehand to get

system values. Making a Data Pump call with this generate mode requires a better understanding of the API itself than is required when using the standard generate mode.

Use this SQL*PLUS command to generate packages and views for an API:

sql > execute hr_pump_meta_mapper.generate (<package_name>, <procedure_name>, false) ;

Use these SQL*PLUS commands to display the help text for the API:

sql > set serveroutput on size 1000000;

sql > execute hr_pump_meta_mapper.generate (<package_name>, <procedure_name>, false) ;

The view and package generated are the same as in the standard generation mode discussed earlier in this essay. They can be used as described in this essay. However, when using this generate mode you should note that:

- There must be a row for the API with API_MODULE_TYPE A1 or BP in HR_API_MODULES. Note that Oracle does not support customer creation of rows in HR_API_MODULES. This is because problems can occur if the data is delivered in future patches.
- You must explicitly set the correct default values for API parameters when you make the Data Pump API call. This is because API parameter default values are not predefined and the meta-mapper makes assumptions about the default parameter values. For details about these assumptions, see Default and NULL Values for API Parameters, page 2-285 (Assumed Default Values).
- You will have to resolve the system values when you set up the data for each individual API call. This is because the generated Data Pump API does not have user keys, or names to identify the system values. This also restricts the mix of API calls within a batch because you cannot pass system identifiers implicitly between API calls. The same restriction applies to the object version number where an API call creates or updates an object.

Table and View Descriptions

The following section provides more details of the specific tables and views that you use with Oracle HRMS Data Pump

HR_API_MODULES

API modules supported by Data Pump

Name	Description
API_MODULE_ID	Sequence generated unique ID.
API_MODULE_TYPE	Type of the API represented by: 'RH' - Row Handler (not of interest to Data Pump). 'BP' - Business Process API.

```
                          'AI' - Alternative Interface API.

MODULE_NAME               API procedure name.

MODULE_PACKAGE            API package name when the

                          module type is 'BP' or 'AI'.
```

HR_PUMP_BATCH_LINE_USER_KEYS

This table holds key mappings between your external system and the Oracle HRMS system. These keys are required for specific entities where it may be difficult to identify the record uniquely in Oracle HRMS from a single field in the batch line table. For example, you might want to use Name||National Identifier from the external system to map to Person ID in Oracle HRMS.

This table is populated automatically by the Data Pump process when you create new records in Oracle HRMS. For example when you load your legacy data. You can insert new lines to this table if you have already loaded your legacy data.

You can have multiple external key mappings to the same unique_key_id in Oracle HRMS. For example, if you want to interface data from an external payroll system and an external benefits system to Oracle HR where the unique IDs are different.

```
Name                          Null?    Type          Description

----------------------------- -------- ----          -------------

USER_KEY_ID                   NOT NULL NUMBER(9)

BATCH_LINE_ID                          NUMBER(9)

USER_KEY_VALUE                NOT NULL VARCHAR2(240) User Defined

                                                     key to identif
y

                                                     a record.

UNIQUE_KEY_ID                 NOT NULL NUMBER(15)    Unique Key in

                                                     Oracle HRMS

LAST_UPDATE_DATE                       DATE

LAST_UPDATED_BY                        NUMBER(15)

LAST_UPDATE_LOGIN                      NUMBER(15)

CREATED_BY                             NUMBER(15)

CREATION_DATE                          DATE
```

HR_PUMP_BATCH_HEADERS

This table holds batch header information for Data Pump. BATCH_NAME is a parameter for the Data Pump concurrent process.

```
 Name                     Null?    Type         Description
```

```
---------------------- -------- -----       ------------

 BATCH_ID               NOT NULL NUMBER(9)

 BATCH_NAME             NOT NULL VARCHAR2(80) Unique name for

                                              the batch

 BATCH_STATUS           NOT NULL VARCHAR2(30) Status can be
                                                decoded using
                                                 'ACTION STATUS'
                                                  lookup type

 REFERENCE                       VARCHAR2(80)

 BUSINESS_GROUP_NAME             VARCHAR2(80)

 LAST_UPDATE_DATE                DATE

 LAST_UPDATE_LOGIN               NUMBER(15)

 LAST_UPDATED_BY                 NUMBER(15)

 CREATED_BY                      NUMBER(15)

 CREATION_DATE                   DATE
```

HR_PUMP_BATCH_LINES

This table holds the individual batch lines that will be loaded by Data Pump

```
Name                 Null?    Type         Description

------------------ -------- ----         -------------

BATCH_LINE_ID       NOT NULL NUMBER(9)    Sequence generated ID

BATCH_ID            NOT NULL NUMBER(9)    Foreign key to

                                          HR_PUMP_BATCH_HEADERS

API_MODULE_ID       NOT NULL NUMBER(9)    Foreign key to

                                          HR_API_MODULES

LINE_STATUS         NOT NULL VARCHAR2(1)  Load status of this API

                                          'U' Unprocessed (initia
l

                                           value)

                                          'V' - Validated but

                                          record not committed

                                          'C' - Complete and
                                            record
```

	committed 'E' - Error
PROCESS_SEQUENCE	NUMBER(9)
USER_SEQUENCE	NUMBER(9)
LINK_VALUE	NUMBER
PVAL001	VARCHAR2(2000)
PVAL002	VARCHAR2(2000)
PVAL003	VARCHAR2(2000)
PVAL004	VARCHAR2(2000)
PVAL005	VARCHAR2(2000)
PVAL006	VARCHAR2(2000)
PVAL007	VARCHAR2(2000)
PVAL008	VARCHAR2(2000)
PVAL009	VARCHAR2(2000)
PVAL010	VARCHAR2(2000)
PVAL230	VARCHAR2(2000)
PLONGVAL	LONG
BUSINESS_GROUP_NAME	VARCHAR2(240)

HR_PUMP_BATCH_EXCEPTIONS

Holds exception information.

Name	Description
EXCEPTION_SEQUENCE	Sequence generated unique ID.
EXCEPTION_LEVEL	Decode using 'MESSAGE_LEVEL' lookup.
SOURCE_ID	BATCH_ID or BATCH_LINE_ID.
SOURCE_TYPE	Indicates what SOURCE_ID holds: 'BATCH_HEADER' : BATCH_ID 'BATCH_LINE' : BATCH_LINE_ID
EXCEPTION_TEXT	Text of exception.

```
HRDPV_CREATE_EMPLOYEE
Name                           Null?    Type

---------------------------------- ----

BATCH_ID                       NOT NULL NUMBER(9)

BATCH_LINE_ID                  NOT NULL NUMBER(9)

API_MODULE_ID                  NOT NULL NUMBER(9)

LINE_STATUS                    NOT NULL VARCHAR2(1)

USER_SEQUENCE                           NUMBER(9)

LINK_VALUE                              NUMBER

BUSINESS_GROUP_NAME              VARCHAR2(240)

P_HIRE_DATE                             VARCHAR2(2000)

P_LAST_NAME                             VARCHAR2(2000)

P_SEX                                   VARCHAR2(2000)

P_PER_COMMENTS                          VARCHAR2(2000)

P_DATE_EMPLOYEE_DATA_VERIFIED           VARCHAR2(2000)

P_DATE_OF_BIRTH                         VARCHAR2(2000)

P_EMAIL_ADDRESS                         VARCHAR2(2000)

P_EMPLOYEE_NUMBER                       VARCHAR2(2000)

P_EXPENSE_CHECK_SEND_TO_ADDRES          VARCHAR2(2000)

P_FIRST_NAME                            VARCHAR2(2000)

P_KNOWN_AS                              VARCHAR2(2000)

P_MARITAL_STATUS                        VARCHAR2(2000)

P_MIDDLE_NAMES                          VARCHAR2(2000)

P_NATIONALITY                           VARCHAR2(2000)

P_NATIONAL_IDENTIFIER                   VARCHAR2(2000)

P_PREVIOUS_LAST_NAME                    VARCHAR2(2000)

P_REGISTERED_DISABLED_FLAG              VARCHAR2(2000)

P_TITLE                                 VARCHAR2(2000)

P_WORK_TELEPHONE                        VARCHAR2(2000)
```

```
P_ATTRIBUTE_CATEGORY                VARCHAR2(2000)
P_ATTRIBUTE1                        VARCHAR2(2000)
P_ATTRIBUTE2                        VARCHAR2(2000)
P_ATTRIBUTE3                        VARCHAR2(2000)
...
P_ATTRIBUTE30                       VARCHAR2(2000)
P_PER_INFORMATION_CATEGORY          VARCHAR2(2000)
P_PER_INFORMATION1                  VARCHAR2(2000)
P_PER_INFORMATION2                  VARCHAR2(2000)
P_PER_INFORMATION3                  VARCHAR2(2000)
...
P_PER_INFORMATION30                 VARCHAR2(2000)
P_BACKGROUND_CHECK_STATUS           VARCHAR2(2000)
P_BACKGROUND_DATE_CHECK             VARCHAR2(2000)
P_BLOOD_TYPE                        VARCHAR2(2000)
P_FAST_PATH_EMPLOYEE                VARCHAR2(2000)
P_FTE_CAPACITY                      VARCHAR2(2000)
P_HONORS                            VARCHAR2(2000)
P_INTERNAL_LOCATION                 VARCHAR2(2000)
P_LAST_MEDICAL_TEST_BY              VARCHAR2(2000)
P_LAST_MEDICAL_TEST_DATE            VARCHAR2(2000)
P_MAILSTOP                          VARCHAR2(2000)
P_OFFICE_NUMBER                     VARCHAR2(2000)
P_ON_MILITARY_SERVICE               VARCHAR2(2000)
P_PRE_NAME_ADJUNCT                  VARCHAR2(2000)
P_PROJECTED_START_DATE              VARCHAR2(2000)
P_RESUME_EXISTS                     VARCHAR2(2000)
P_RESUME_LAST_UPDATED               VARCHAR2(2000)
P_SECOND_PASSPORT_EXISTS            VARCHAR2(2000)
```

```
P_STUDENT_STATUS                        VARCHAR2(2000)

P_WORK_SCHEDULE                         VARCHAR2(2000)

P_SUFFIX                                VARCHAR2(2000)

P_PERSON_USER_KEY                       VARCHAR2(2000)

P_ASSIGNMENT_USER_KEY                   VARCHAR2(2000)

P_PER_OBJECT_VERSION_NUMBER             VARCHAR2(2000)

P_ASG_OBJECT_VERSION_NUMBER             VARCHAR2(2000)

P_PER_EFFECTIVE_START_DATE              VARCHAR2(2000)

P_PER_EFFECTIVE_END_DATE                VARCHAR2(2000)

P_FULL_NAME                             VARCHAR2(2000)

P_PER_COMMENT_ID                        VARCHAR2(2000)

P_ASSIGNMENT_SEQUENCE                   VARCHAR2(2000)

P_ASSIGNMENT_NUMBER                     VARCHAR2(2000)

P_NAME_COMBINATION_WARNING              VARCHAR2(2000)

P_ASSIGN_PAYROLL_WARNING                VARCHAR2(2000)

P_USER_PERSON_TYPE                      VARCHAR2(2000)

P_VENDOR_NAME                           VARCHAR2(2000)

P_CORRESPONDENCE_LANGUAGE               VARCHAR2(2000)
```

PAY_ACTION_PARAMETER_GROUPS

```
Name                          Null?      Type

-------------------------------------    ----

ACTION_PARAMETER_GROUP_ID     NOT NULL NUMBER(9)

ACTION_PARAMETER_GROUP_NAME NOT NULL VARCHAR2(30)
```

PAY_ACTION_PARAMETER_VALUES

```
Name                          Null?      Type

-------------------------------------    ----

PARAMETER_NAME                NOT NULL    VARCHAR2(30)

PARAMETER_VALUE               NOT NULL    VARCHAR2(80)

ACTION_PARAMETER_GROUP_ID                 NUMBER(9)
```

```
Note: The PAY_ACTION_PARAMETERS view just returns those rows from
 PAY_ACTION_PARAMETER_VALUES that have a NULL_ACTION_PARAMETER_GRO
UP_ID
```

SQL Trace

SQL Trace

The SQL trace facility provides you with performance information on individual SQL statements. You can enable the trace facility for either a session or an instance.

For each SQL statement traced, the following performance information is generated:

- SQL statement text
- Parse, Execute and Fetch count, CPU/elapsed times, physical/logical reads and rows processed
- The optimized goal
- Misses in the library cache during parse
- The Explain Plan at time of SQL execution (Oracle 8.1.6+)
- User for which the parse occurred
- Recursive SQL depth

When you enable the trace facility, the performance information for executed SQL statements is written out to a trace file until the SQL trace facility is disabled.

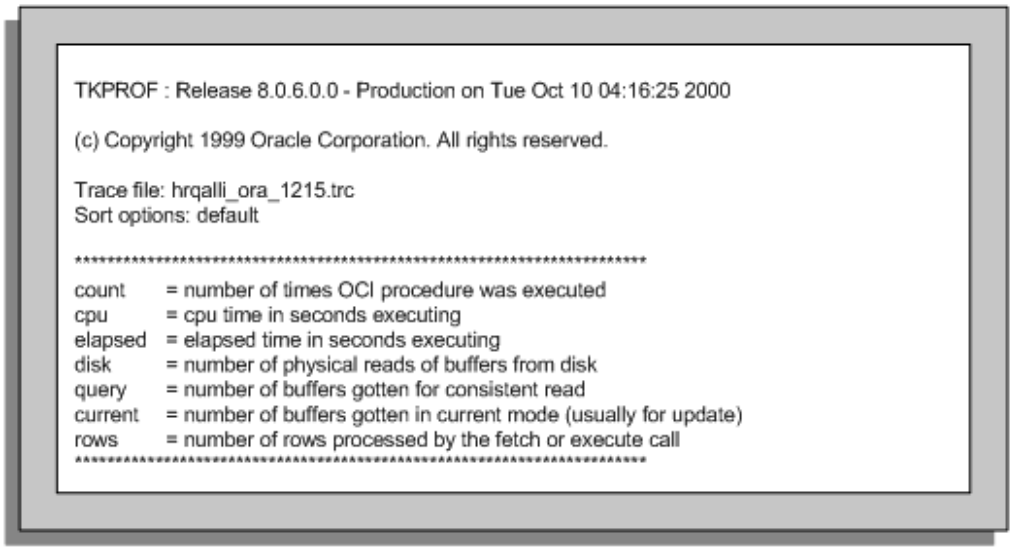
> **Note:** You need Oracle 8.1.6 and Oracle Applications Release 11*i* to be able to use SQL Trace.

Using SQL Trace

To use SQL Trace, first enable it, then the desired SQL application/process/statement(s) are executed. When all the SQL statements have been executed, SQL Trace is disabled.

Viewing the Content of the Trace

Once you have generated the Trace file, you can convert it into a user-friendly report using the Oracle reporting program TKPROF. Alternatively, you can view the generated raw trace file directly.



```
TKPROF : Release 8.0.6.0.0 - Production on Tue Oct 10 04:16:25 2000

(c) Copyright 1999 Oracle Corporation. All rights reserved.

Trace file: hrqalli_ora_1215.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************
```

Note: If you enable SQL Trace, an additional processing overhead is incurred, although the impact on performance is minor.

Enabling SQL Trace

You enable and disable SQL Trace through the init.ora parameter sql_trace. The parameter accepts a Boolean value of TRUE or FALSE. The parameter is set at the system level in the init.ora file. Alternatively, you can set it dynamically for a session using the SQL command ALTER SESSION, or PL/SQL dbms_session.set_sql_trace, dbms_system.set_sql_trace_in_session.

These are Oracle supplied packaged procedures.

Related Trace init.ora Parameters

The following table details parameters that enable timings, directory location, maximum trace file size and trace file access protections to be specified and adhered to when SQL Trace is enabled.

Related Trace init.ora Parameters

Parameter	Meaning
timed_statistics	Specifies if time statistics are to be collected or not. Valid values are TRUE or FALSE. The timing has a resolution of 1/100th of a second. Any operation that is less than this may not be timed accurately. If this parameter is FALSE, timings are not recorded and are shown as 0 in the trace file. For tkprof the 'cpu' and 'elapsed' times will be 0.
max_dump_file_size	Specifies the maximum SQL Trace file size in O/S blocks if just a number, bytes if K or M is specified or unlimited if UNLIMITED is specified. If the size of the trace exceeds the size of max_dump_file_size then the *** Trace file full *** message appears at the end of the file.
user_dump_dest	Specifies the directory where the SQL Trace is to be placed. If the values is ?/log then ? means $ORACLE_HOME because the DBA has not changed the default trace file destination.
_trace_files_public	Specifies if a trace file is written out with public access settings. Valid values are TRUE or FALSE.

init.ora Parameters

You can view these init.ora parameters from an Oracle Session by examining the v$parameters table below:

```
SELECT name

,value

FROM v$parameter

WHERE name IN

('timed_statistics'

,'max_dump_file_size'

,'user_dump_dest'

,'_trace_file_public');

NAM      VALUE

----------------   ------

timed_statistics   TRUE

user_dump_dest   usr/oracleHR/log
```

```
max_dump_file_size  204800

_trace_file_public  TRUE
```

Selecting SQL Trace init.ora Parameters

You can set the timed_statistics and max_dump_file_size dynamically at either the session or system level, using the ALTER SESSION or ALTER SYSTEM commands.

```
ALTER SESSION SET timed_statistics=TRUE;

ALTER SYSTEM SET timed_statistics=FALSE;

ALTER SESSION SET max_dump_file_size=204800;

ALTER SYSTEM SET max_dump_file_size=204800;
```

You can only set the user_dump_dest parameter dynamically at the system level. You can only set the _trace_file_public parameter in the init.ora file.

Tracing Oracle Payroll Processes and Reports

When SQL Trace is enabled for Oracle Payroll processes, each process produces a trace file for the session in which the Trace is executed. If the process is run in parallel, for example, the Payroll Run, a trace file is produced for each thread.

You can enable and disable Trace for Oracle Payroll processes and reports by setting the parameter TRACE in the PAY_ACTION_PARAMETERS table. You can do this by one of two methods, using SQL *Plus, or the Action Parameters window.

Method 1: Using SQL *Plus

You can set the parameter to **Y** (enable trace), or to **N** (disable trace). For example:

```
/* To enable SQL Trace */

UPDATE pay_action_parameters

SET  parameter_value = 'Y'

WHERE  parameter_name = 'TRACE';

COMMIT;

/* To disable SQL Trace */

UPDATE pay_action_parameters

SET  parameter_value = 'N'

WHERE  parameter_name = 'TRACE';

COMMIT;
```

Method 2: Using the Action Parameters Window

Alternatively, you can enable Trace using the Action Parameters window.

1. Select Trace as the parameter name.
2. Enter Y to enable trace, or N to disable trace.

Tracing HRMS Application Forms

You can trace HRMS Application forms if the system administrator has granted access to the 'HR Debug Tools' facility.

1. Select Help->Diagnostics->Trace from the menu option.
2. Check the Trace check box.

   Uncheck the Trace check box if you want to disable the utility.

Dynamically Tracing from SQL *Plus

You can use either the ALTER SESSION or PL/SQL packaged procedure dbms_session.set_sql_trace to trace from SQL *Plus. Whichever method you use, SQL_TRACE is enabled, the SQL statements are executed and SQL_TRACE is disabled to stop the trace.

```
SQL> ALTER SESSION SET SQL_TRACE=TRUE;

SQL> Execute SQL statements

SQL> ALTER SESSION SET SQL_TRACE=FALSE;

Or

SQL> EXECUTE dbms_session.set_sql_trace(TRUE);

SQL> Execute SQL statements

SQL> EXECUTE dbms_session.set_sql_trace(FALSE);
```

You can run the SQL Trace facility in any current active Oracle Session by using the dbms_system.set_sql_trace_in_session packaged procedure. This procedure accepts the three following arguments:

- SID
- SERIAL#
- SQL_TRACE

You can determine the SID and SERIAL# values from the v$session table. Further filtration on the v$session columns osuser name, username, and such, can help identify the SID/SERIAL# values. For example:

```
SELECT s.sid,

s.serial#

FROM  v$session s

WHERE  s.osuser = 'afergusson'

AND  s.username = 'APPS'

SID  SERIAL#

---  --------

15  19201
```

```
execute

dbms_system.set_sql_trace_in_session(15,19201, TRUE);
```

The SQL_TRACE argument is Boolean and accepts TRUE or FALSE values.

Locating the Trace File

You specify the location of the Trace file using the user_dump_dest parameter. The Trace file name is dependent on the operating system:

- On UNIX, the trace file name is SID_ora_PID.trc
- On NT, the trace file name is oraPID.TRC

SID is the Oracle System Identifier, and PID is the operating system Process Identifier. The PID is determined by interrogating the v$process and v$session dynamic tables for a specific active session ID.

The example below illustrates selecting a PID for a specific active session.

```
SELECT p.spid,

FROM  v$session s, v$process p

WHERE  s.audsid = &sessionid

AND  p.addr = s.paddr:

SPID

----------

89012
```

What is TKPROF?

TKPROF is a program that formats a raw SQL Trace file into a user-friendly file. The program reads the Trace file and creates a file that has the following section:

- Header
- Body
- Summary.

The header section contains the version of TKPROF, any sort options used and a glossary of terms. The body section displays the following information for each user level SQL statement traced:

- SQL statement text
- Tabulated Parse
- Execute and Fetch statistics
- Number of library cache misses during Parse
- Parsing user id

If specified, TKPROF also:

- Shows the explain plans when the SQL Trace was executed and when TKPROF was run
- Creates a SQL script that creates a table and inserts a row of statistics for each SQL statement

The power of TKPROF is the ability to sort the SQL statements. The sorting helps identify and sequence statements that are using the most resources. At the end of the report, a tabular summary for all the user level and recursive SQL statements is provided.

Formatting a Trace File using TKPROF

You execute TKPROF from the command line. Although TKPROF has many arguments, generally only two mandatory and three optional arguments are used. The execution syntax and arguments are as follows:

tkprof infile outfile sort=options explain=username/password@db print=integer

The tkprof arguments are:

Table of tkprof Arguments

Argument	Meaning
infile	Specifies the raw SQL Trace file
outfile	Specifies the file that TKPROF will create the report in
explain	Optionally specifies the Oracle username, password and DB connector where each SQL statement is to be explained. For Oracle 8.1.6, if explain is specified, then two plans are provided. The first plan is for when the SQL Trace was generated (and is always present regardless of the explain option setting). The second plan is generated when TKPROF is executed
print	Limits the number of SQL statements to be included in the report. The argument accepts an integer value. This is particularly useful is you have a large SQL Trace file. For example, you may want to examine the worst 25 SQL statements only and would use print=25
sort	Optionally specifies a sort order in descending order. The order comprises of one or more options. If the sort argument is omitted, the SQL statements are ordered in the order they are located in the trace file. More than one option can be specified provided a space separates them. The options available are shown in the following table.

Table of Sort Options

Sort Options	Meaning
PRSCNT	Number of times parsed
PRSCPU	CPU time spent parsing
PRSELA	Elapsed time spent parsing
PRSDSK	Number of physical reads from disk during parse
PRSQRY	Number of consistent mode block reads during parse
PRSCU	Number of current mode block reads during pars
PRSMIS	Number of library cache misses during parse
EXECNT	Number of executes
EXECPU	CPU time spent executing
EXEELA	Elapsed time spent executing
EXEDSK	Number of physical reads from disk during execute
EXEQRY	Number of consistent mode block reads during execute
EXECU	Number of current mode block reads during execute
EXEROW	Number of rows processed during execute
EXEMIS	Number of library cache misses during execute
FCHCNT	Number of fetches
FCHCPU	CPU time spent fetching
FCHELA	Elapsed time spent fetching
FCHDSK	Number of physical reads from disk during fetch
FCHQRY	Number of consistent mode block reads during fetch
FCHCU	Number of current mode block reads during fetch
FCHROW	Number of rows fetched

Typical TKPROF Execution Examples:

- Standard report

  tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp

- Report with Explain option

  tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp explain=apps/apps@hrdb

- Report with explain, sorted by execute/fetch elapsed time for the worst 25 statements

  tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp explain=apps/apps@hrdb sort= exeela fchela print=25

TKPROF Sort Options

TKPROF provides a number of sort options which can be specified to sort the traced SQL statements. Some recommended sort options are listed below:

- Sort by logical IO

  tkprof infile outfile sort=exeqry execu fchqry fchcu

- Sort by physical IO

  tkprof infile outfile sort=exeqry execu fchqry fchcu

- Sort by CPU time (only if the timed_statistics is enabled)

  tkprof infile outfile sort=execpu fchcpu prscpu

- Sort by elapsed time (only if the timed_statistics is enabled) tkprof infile outfile sort=exeela fchela prsela

- Sort by library cache misses

  tkprof infile outfile sort=prsmis

HRMS Development prefers both a Raw SQL Trace file and a TKPROF report sorting by execute elapsed (exeela) time and fetch elapsed (fchela) time providing timed_statistics is set to TRUE. If timed_statistics is FALSE, then the execute disk (exedsk), execute query (exeqry) and execute cpu (execu) sort options should be used.

Understanding a TKPROF Report

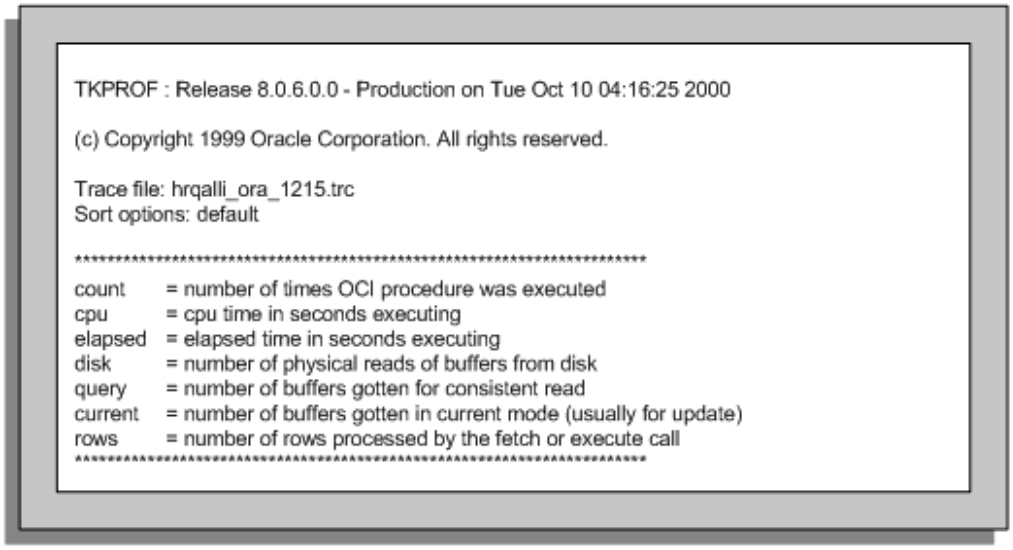
After running tkprof, the resulting file contains a report which is divided into three sections:

- Header
- Body
- Summary

TKPROF Header

The header shows the TKPROF version, date of run, the SQL Trace infile, any sort options (default if no options specified) and a glossary for terms used in the statistic table.

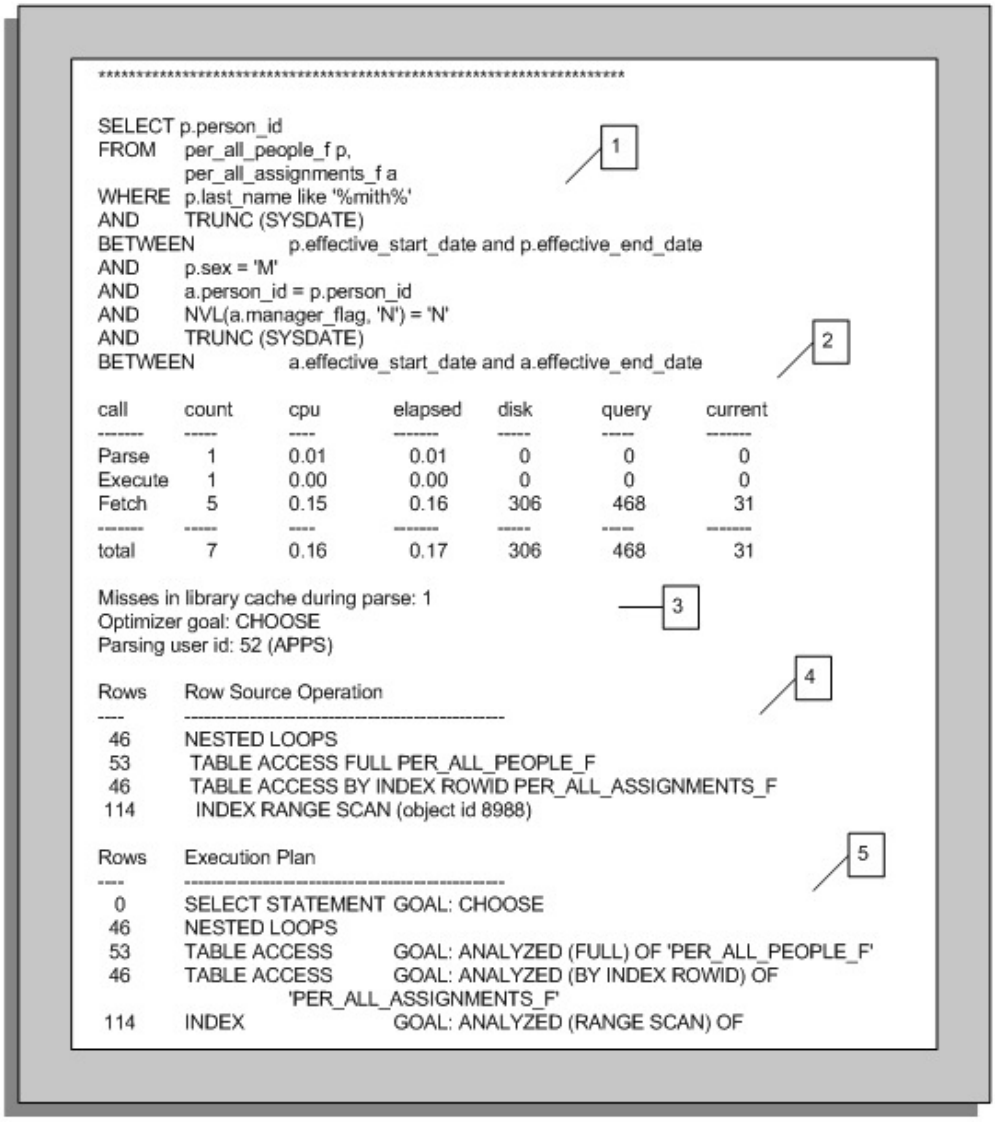
TKPROF header



TKPPROF Body

The body contains all the SQL statements which have been traced. Each SQL statement is shown with it statistics and explain plan in sorted order.

TKPROF body



```
********************************************************************************

SELECT p.person_id                                                    1
FROM    per_all_people_f p,
        per_all_assignments_f a
WHERE   p.last_name like '%mith%'
AND     TRUNC (SYSDATE)
BETWEEN             p.effective_start_date and p.effective_end_date
AND     p.sex = 'M'
AND     a.person_id = p.person_id
AND     NVL(a.manager_flag, 'N') = 'N'
AND     TRUNC (SYSDATE)                                               2
BETWEEN             a.effective_start_date and a.effective_end_date

call     count       cpu    elapsed       disk      query    current
-------  ------  --------  ----------  ----------  ----------  ----------
Parse        1      0.01       0.01          0          0          0
Execute      1      0.00       0.00          0          0          0
Fetch        5      0.15       0.16        306        468         31
-------  ------  --------  ----------  ----------  ----------  ----------
total        7      0.16       0.17        306        468         31

Misses in library cache during parse: 1                               3
Optimizer goal: CHOOSE
Parsing user id: 52 (APPS)

Rows     Row Source Operation                                         4
-------  ---------------------------------------------------
     46  NESTED LOOPS
     53   TABLE ACCESS FULL PER_ALL_PEOPLE_F
     46   TABLE ACCESS BY INDEX ROWID PER_ALL_ASSIGNMENTS_F
    114    INDEX RANGE SCAN (object id 8988)

Rows     Execution Plan                                               5
-------  ---------------------------------------------------
      0  SELECT STATEMENT GOAL: CHOOSE
     46  NESTED LOOPS
     53  TABLE ACCESS        GOAL: ANALYZED (FULL) OF 'PER_ALL_PEOPLE_F'
     46  TABLE ACCESS        GOAL: ANALYZED (BY INDEX ROWID) OF
                 'PER_ALL_ASSIGNMENTS_F'
    114  INDEX                GOAL: ANALYZED (RANGE SCAN) OF
```

1.Illustrates the SQL Statement Being Traced

The SQL statement being processed is shown, together with any bind variables without truncation. Only the following SQL statements are truncated to 25 characters:

`SET ROLE, GRANT, ALTER USER, ALTER ROLE, CREATE USER, CREATE ROLE`

2 Illustrates the Parse, Execute and Fetch Tabular Statistics for the SQL Statement

The tabular statistics table below is the most important information to examine for each parse, execute and fetch call.

Tabular Statistics

Call	Purpose
parse	The parse call is responsible for syntax/ semantic checking, type checking, execution plan generation and the building of a shared cursor. Depending on the SQL statement being parsed, either a hard or soft parse will be performed. If the SQL statement was not found in the shared cached then a hard parse if performed. A hard parse will perform all of the parsing steps required and is the most expensive parse operation. If the SQL statement does exist in the shared cache then a complete parse operation does not need to be performed because the shared cursor definition can be used, this is known as a soft parse
execute	Will execute the SQL statement or in the event of a SELECT prepare for fetching.
fetch	Fetches rows which are returned from a SELECT SQL statement. For a SELECT that contains an ORDER BY or a FOR UPDATE clause, rows may be accessed during execute.
count	The number of calls for each call type.
cpu	CPU time in seconds (always zero if timed_statistics is FALSE). For parse, if a statement was found in the shared pool (i.e. no library cache misses) then this will be 0.
elapsed	Elapsed time in seconds (always zero if timed_statistics is FALSE).
disk	Number of physical reads of buffers from the database files. (Physical I/O).
query	Number of buffers gets in a consistent (query) mode from memory. (Logical I/O). This column usually reflects the processing of a SELECT statement.
current	Number of buffers read in current mode from memory. This column usually reflects the processing of a DML INSERT, UPDATE or DELETE statement.
rows	The number of rows processed by each type. For SELECT statements, the number of rows will be in the fetch column. For INSERT, UPDATE and DELETE statements, the number of rows will in the execute column.

The statistics can be useful in determining other statistical values and pointers to where particular problems are occurring.

For example:

Total logical IO buffer gets

total logical IO = query total + current total

This statistic provides the total number of data buffers retrieved from memory.

Logical IO per row

logical IO per row = total logical IO / total rows

This statistic will provides the total number of data buffers retrieved from memory for each row processed. The greater the number of logical IOs performed the greater the row cost. Ideally this ratio should be as low as possible.

Logical IO per execute

logical IO per execute = total logical IO / execute count

This statistic is similar to 'logical IO per row' but is based on per execute.

Parses per execute

parses per execute = parse count / execute count

This statistic determines the number of parses per execute. If this value is close to or is 1 (providing more than 1 execute has taken place) then a parse is being performed for each execute and the cursor is not being re-used. The shared pool size may not be large enough and may need increasing.

Buffer cache miss rate

buffer cache miss rate = disk total / total logical IO

This statistic provides the miss rate for data not being cached in memory. Ideally this figure should be less than 10%.

Average time per execution

avg. time per execute = elapsed total/execute count

This statistic provides the average time it takes to execute the statement. The figure is really a guideline to determine if it is acceptable by the end user.

Average rows per fetch

avg. rows per fetch = fetch rows/fetch count

This statistic will provide the average number of rows fetched per fetch call. This is particularly useful in determining if array fetching is being used.

3. Illustrates the Number of Misses in the Library Cache During Parse, the Optimizer Mode Used and the Parsing User Id

The statistic 'Misses in library cache during parse indicates if the SQL statement was hard or soft parsed. If a miss has occurred (i.e. > 0) then the SQL statement was not found in the shared cursor cache and was hard parsed. If a miss did not occur (i.e. = 0) the SQL was found in the shared cursor cache and was soft parsed. If this statistic is consistently being set (e.g. > 0) then investigation will be required to determine why the SQL is not being shared.

The statistic 'Optimizer goal' shows the goal used by the Optimizer to process the SQL statement. The goal will be one of the following values:

CHOOSE, FIRST_ROWS, ALL_ROWS or RULE

The 'Parsing user id' shows the user who issued the SQL command.

4. The Explain Plan Generated when the SQL Statement was Traced

The runtime explain plan is generated when the SQL statement was executed. This explain plan is always present regardless if the explain option is specified as a tkprof argument (although sometimes it is not shown if the user does not have access to all the underlying objects). Additionally, the plan contains object ids instead of names for referenced objects. These object ids map directly onto the all/dba/user_objects tables where the object_name can be retrieved.

5. The Explain Plan Generated when the SQL Statement was Processed by TKPROF Provided the Explain Argument was Specified

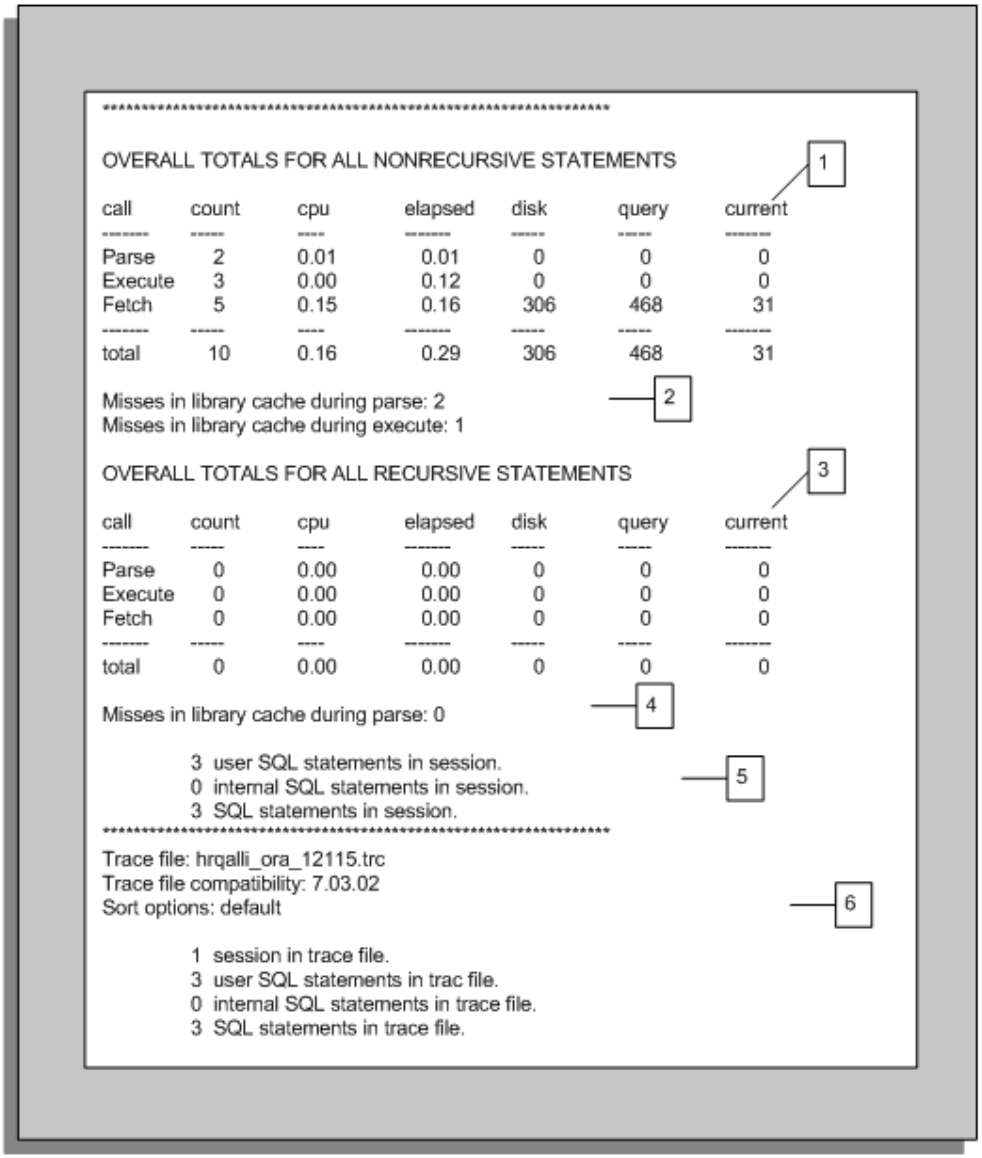
By providing TKRPOF with the explain argument, each SQL statement will be explained during the TKRPOF processing. The fundamental difference between this and the explain plan generated at SQL Trace execution is they can be different if any of the underlying objects or corresponding database statistics have changed (if using the Cost Based Optimizer). Also, all object names are displayed instead of object ids.

The 'Rows' column shows the number of rows processed by each operation.

TKPROF Summary

The summary is located at the end of the TKPROF file after all the traced SQL statements.

TKPROF summary



```
********************************************************************************

OVERALL TOTALS FOR ALL NONRECURSIVE STATEMENTS                              1

call     count       cpu    elapsed       disk      query    current
-------  ------  --------  ----------  ----------  ----------  ----------
Parse        2      0.01       0.01          0           0           0
Execute      3      0.00       0.12          0           0           0
Fetch        5      0.15       0.16        306         468          31
-------  ------  --------  ----------  ----------  ----------  ----------
total       10      0.16       0.29        306         468          31

Misses in library cache during parse: 2                                     2
Misses in library cache during execute: 1

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS                                 3

call     count       cpu    elapsed       disk      query    current
-------  ------  --------  ----------  ----------  ----------  ----------
Parse        0      0.00       0.00          0           0           0
Execute      0      0.00       0.00          0           0           0
Fetch        0      0.00       0.00          0           0           0
-------  ------  --------  ----------  ----------  ----------  ----------
total        0      0.00       0.00          0           0           0

Misses in library cache during parse: 0                                     4

    3  user SQL statements in session.                                      5
    0  internal SQL statements in session.
    3  SQL statements in session.
********************************************************************************
Trace file: hrqalli_ora_12115.trc
Trace file compatibility: 7.03.02                                           6
Sort options: default

    1  session in trace file.
    3  user SQL statements in trac file.
    0  internal SQL statements in trace file.
    3  SQL statements in trace file.
```

1. Illustrates Overall Totals for Non-Recursive SQL Statements

Non-recursive SQL statements are user level SQL statements, such as SQL written by developers.

The 'OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS' tabular table contains the sum of all user issued statements not including an Recursive SQL issued (see number 3 below for Recursive SQL description).

2. Illustrates the Library Cache Misses During Execute and Parse

As mentioned previous in the body section, the library cache misses indicates the number of Non-recursive SQL Statements not being shared, for example, user.

3. Illustrates the Overall Totals for Recursive SQL Statements

Recursive SQL are internal statements issued by Oracle in to complete a user SQL statement. Typical examples are dynamic space management, getting missing data dictionary information, and so on.

Statistics for Recursive SQL are not included in the statistics for the SQL statements which issued the calls. Therefore, the total resources/cost for a SQL statement is the SQL statement plus all corresponding Recursive SQL values.

The 'OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS' tabular table contains the sum of all the Recursive SQL in the SQL Trace file. These figures are important to determine how much extra work is being performed internally by Oracle in order to satisfy the user SQL statements.

4. Illustrates the Library Cache Misses During Parse

As mentioned previous in the body section, the library cache misses indicates the number of Recursive SQL Statements not being shared.

5. Illustrates the Summary of SQL Statements Processed

Provides a quick summation of the number of 'user SQL statements in session' (Non-recursive), '0 internal SQL statements in session' (Recursive) and 'SQL statements in session' (total of Non-recursive + Recursive SQL statements).

6. Illustrates the TKRPOF Compatibility and Processing Statistics

Lists the SQL Trace file which has been processed, the trace file compatibility and sort options. Additionally the number of sessions, unique SQL statements and number of lines in the SQL Trace file are provided.

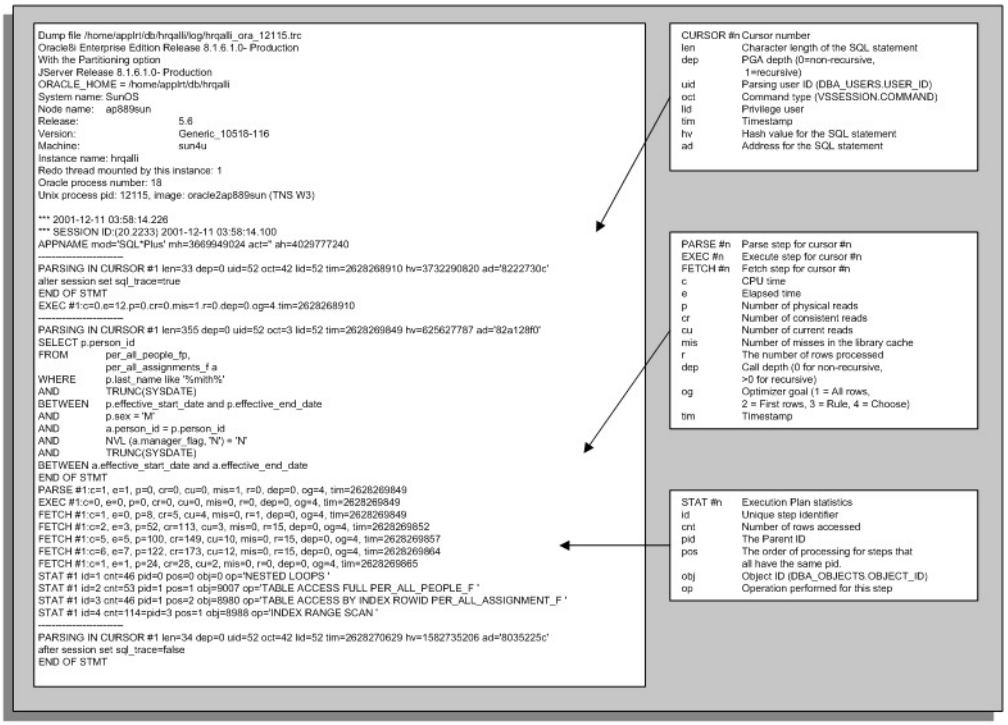
Raw SQL Trace File Example

The following example illustrates a simple, raw SQL Trace file produced for three SQL statements:

- ALTER SESSION SET SQL_TRACE=TRUE
- SELECT
- ALTER SESSION SET SQL_TRACE=FALSE.

The Trace file is more difficult to read than the TKRPOF report, and is not in any sorted order.

Example Trace file



Advanced SQL Tracing Using Event 10046

The 10046 Event enables extra information on bind variables and waits to be reported in the Raw SQL Trace file. This extra information is determined by setting the event level. The Event has four level settings which are described in the table below:

Event Level Settings

Level	Setting
1	Default SQL Trace
4	Include bind variable information
8	Include wait event information
12	Include bind variable and wait event information

By default, each SQL Trace is set to level 1. To enable extra information to be reported, the 10046 Event is set to the desired reporting level using the ALTER SESSION command.

```
ALTER SESSION SET EVENTS ’10046 trace name context forever, level
 1’;
```

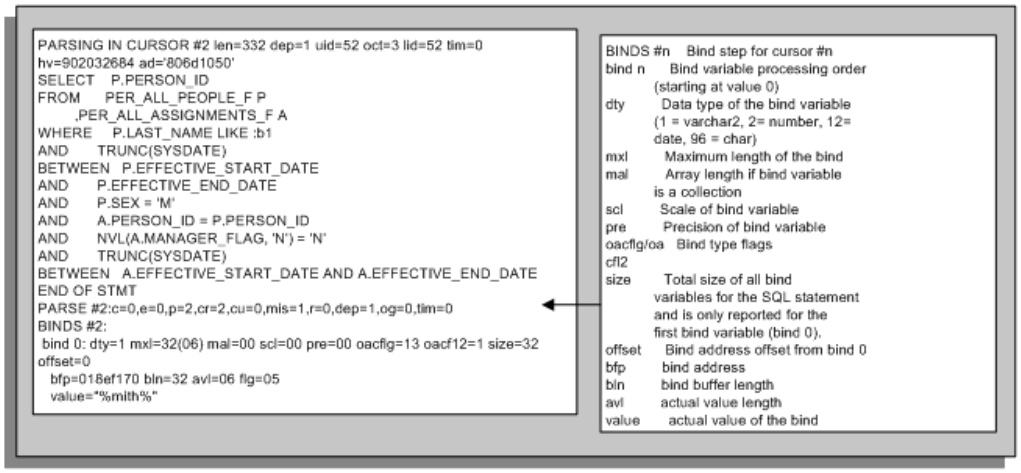
```
ALTER SESSION SET EVENTS ’10046 trace name context forever, level
 4’;

ALTER SESSION SET EVENTS ’10046 trace name context forever, level
 8’;

ALTER SESSION SET EVENTS ’10046 trace name context forever, level
 12’;
```

By setting the event level to either 4, 8 or 12, the extra information is reported in the Raw SQL Trace file if SQL Trace is enabled. It is important to note that TKPROF ignores any extra information reported from setting events.

Event 10046 Bind Variable information

When the 10046 is set to level 4 or 12 bind variable information is provided if the traced SQL statement contains bind variables. This is particularly useful if you need to review the bind variable values being used.

Event 10046 Bind Variable Information



Event 10046 Wait Event information

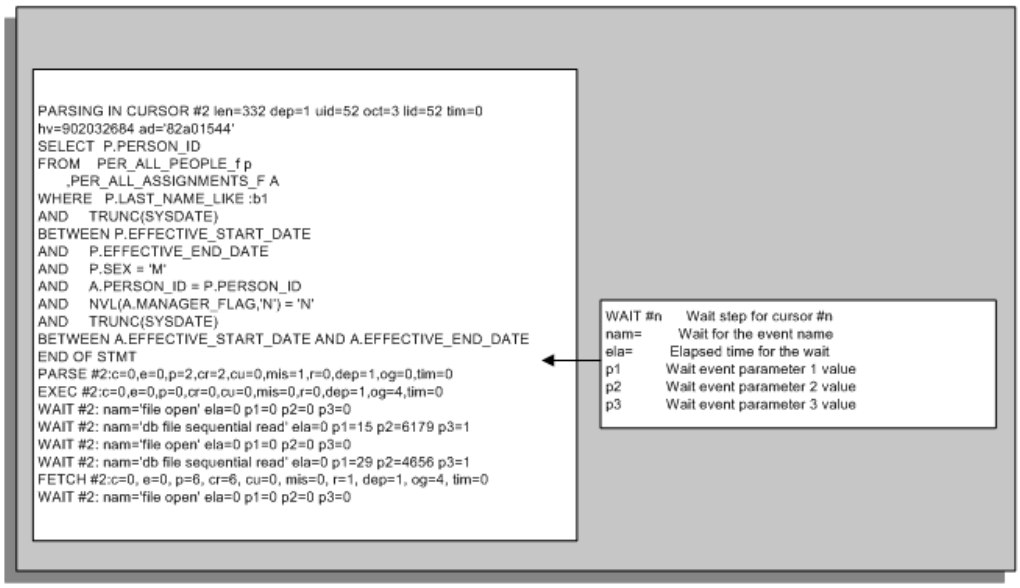
When the 10046 is set to level 8 or 12, wait event information is provided if the traced SQL statement contains waits. The wait event names are the same events which are from V$SYSTEM_EVENT. Each event has three parameters:

- p1
- p2
- p3

These are the same as the parameters in V$SESSION_WAIT. For a full event and parameter description please refer to the Oracle 8i Reference Release 2 (8.1.6) Part Number A76961-01, Appendix A - Oracle Wait Events.

Tracing for Wait Events can be very useful in identifying why the elapsed time of a SQL statement is higher than expected. For example, the session may be waiting on a latch, I/O, SQL*Net, and so on.

Event 10046 Wait Event Information



Backfeed

Oracle Generic Third Party Payroll Backfeed

This essay provides the information that you need to understand and use the Oracle Generic Third Party Payroll Backfeed. To understand this information you should already have a good functional and technical knowledge of the Oracle HRMS product architecture, including:

- The data model for Oracle HRMS.
- The API strategy and how to call APIs directly.
- How to code PL/SQL.
- The HRMS parameters that control the running of concurrent processes.
- How to use and configure Data Pump.

Contents

This essay contains the following sections:

- Overview, page 2-327

  Provides an overview of Oracle Generic Third Party Payroll Backfeed

- Setting Up Oracle Generic Third Party Payroll Backfeed, page 2-327

  Describes the steps for setting up Third Party Payroll Backfeed at a high level. Each step is explained in more detail in the following sections:

  - Installing the Generic Payroll Backfeed, page 2-328
  - Payment Information, page 2-329
  - Balance Types, page 2-329
  - Setting Up Data Pump, page 2-331

- Setting Up the Data Uploader, page 2-332

- Using Backfeed to Upload Payroll Run Results, page 2-335

  Describes the steps for using Third Party Payroll Backfeed at a high level. Each step is explained in more detail in the following sections:

  - Using the Load Sheets Macro, page 2-337
  - Using the Save Sheets Macro, page 2-337
  - Running Data Uploader, page 2-338
  - Running Data Pump, page 2-338

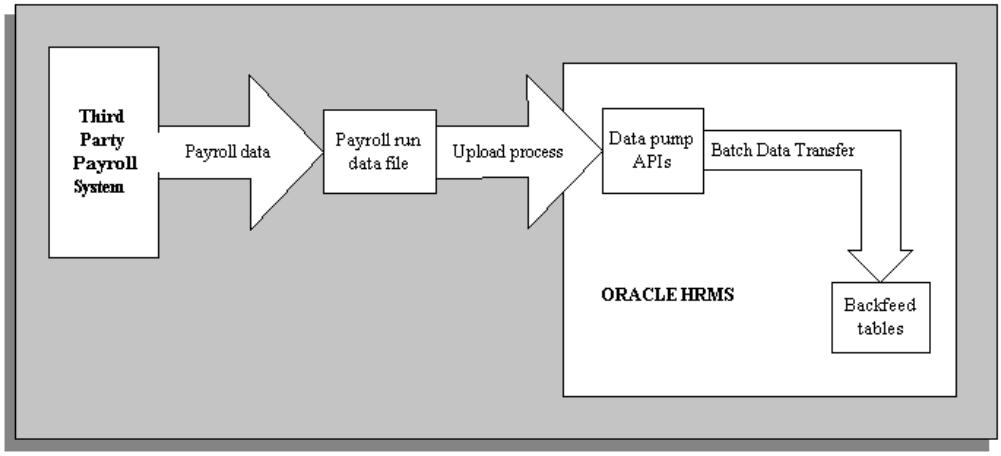
- Viewing Third Party Payroll Run Results in Oracle HRMS, page 2-339

  Describes how you view the payroll run results in Oracle HRMS windows.

Overview

If you use a third party payroll system, Oracle Generic Third Party Payroll Backfeed enables you to upload information supplied by your payroll system for a payroll run into the Oracle HRMS tables. This information can include payment information and balance details calculated by your third party payroll system. You can then view this information using Oracle HRMS windows and generate reports based on this information.

Backfeed Process



The payroll results data that is uploaded using Backfeed is held in specific Backfeed tables, not tables belonging to Oracle Payroll. This means that if you are using Oracle Payroll and a third party payroll system, your Oracle Payroll implementation is not impacted by Backfeed.

This generic version of Oracle Third Party Payroll Backfeed is vendor independent. It can be configured during implementation to fit the requirements of your third party payroll system and your HRMS implementation.

Setting Up the Generic Payroll Backfeed

To set up the Generic Payroll Backfeed, follow this sequence of tasks:

1. Install the Generic Payroll Backfeed

See: Installing the Oracle Generic Third Party Payroll Backfeed, page 2-328

2. Ensure that payment information is set up for Oracle HRMS if you intend to upload payment information using Backfeed.

   See: Payment Information, page 2-329

3. Enter the names of the balance types that will be uploaded into Oracle HRMS from your third party payroll system.

   See: Balance Types, page 2-329

4. Decide which upload option to use.

   See: Deciding Which Upload Option to Use, page 2-332

5. Set Up Data Pump.

   See: Setting Up Data Pump, page 2-331

6. Run Data Pump Meta-Mapper.

   See the *Oracle HRMS Data Pump* technical essay for further details.

7. Set up Data Uploader

   See: Setting Up Data Uploader, page 2-332

8. Add the View Third Party Payroll Employee Run Results, View Third Party Payroll Organization Run Results and the Enter Third Party Payroll Balance Types form functions to your menus. Use the Menus window.

   See: *Oracle Applications System Administrator's Guide*

9. Create new folder definitions in the Third Party Payroll Run Employee Results window and the Third Party Payroll Run Organization Results, if required, so information relevant to your enterprise is displayed.

Installing the Oracle Generic Third Party Payroll Backfeed

Release 11*i*

If you are using Oracle HRMS 11*i* you should apply the patches listed below. You can obtain these patches from Oracle Support or Metalink.

> **Note:** These patches are subject to change. Please contact Oracle Support for the latest information.

Install the Backfeed tables - Patch Number 1287911

This patch installs the Third Party Payroll Backfeed tables, APIs, forms, and views.

Install Data Pump Configuration Data - Patch Number 1313097

This patch delivers some Data Pump configuration data that enables Data Pump to call the Backfeed APIs. Also included are some PL/SQL functions that resolve the Oracle HR system ids. These functions make certain assumptions about your Oracle HRMS implementation. The functions are documented in the Reference Information section of this document. If the assumptions are not valid for your implementation you will have to configure some of the scripts that are delivered by patch 1313097.

Install Data Uploader - Patch Numbers 1164750 and 1316578

These patches deliver the Data Uploader and seed data to enable you to use the Data Uploader functionality as part of your Third Party Payroll Backfeed. If you have changed

the PL/SQL functions that are delivered in patch 1313097, you may need to change the seed data delivered by patch 1316578.

Install Enhancement to support Descriptive Flexfields - Patch Number 1928571

This patch delivers the updates to tables, views, business views, forms, APIs, row handlers, messages and datapump scripts required to support descriptive flexfields. You should install this patch if you wish to hold additional information against a balance amount, payment detail, payroll run or processed assignment, or if you want to use the business views; for example to create Oracle Discover reports.

Release 11.0

If you are using Oracle HRMS 11.0.x you should apply the patches listed below. You can obtain these patches from Oracle Support or Metalink.

> **Note:** These patches are subject to change. Please contact Oracle Support for the latest information.

Install the Backfeed tables - Patch Number 1198005

This patch installs the Third Party Payroll Backfeed tables, APIs, forms, and views.

Install Data Pump - Patch Numbers 1053696 and 1077660

These patches deliver enhancements to Data Pump and some Data Pump configuration data that enables Data Pump to call the Backfeed APIs. Also included are some PL/SQL functions that resolve the Oracle HR system ids. These functions make certain assumptions about your Oracle HRMS implementation. The functions are documented in the Reference Information section of this document. If the assumptions are not valid for your implementation you will have to configure some of the scripts that are delivered by patch 1077660.

Install Data Uploader - Patch Numbers 1325570 and 1176584

These patches deliver the Data Uploader and seed data to enable you to use the Data Uploader functionality as part of your Third Party Payroll Backfeed. If you have changed the PL/SQL functions that are delivered in patch 1077660, you may need to change the seed data delivered by patch 1176584.

Install the Business Views - Patch Number 1198041

This patch delivers the business views for the Oracle Generic Third Party Payroll Backfeed. You should install this if you want to use the business views, for example to create Oracle Discoverer reports.

Payment Information

All employees for whom payments information is to be loaded using the Backfeed must have personal payment methods set up in Oracle HRMS before the Backfeed is run.

This information should be entered using the Organizational Payment Method, and the Personal Payment Method windows.

See: Payrolls and Other Employment Groups, and Employment Information, *Oracle Human Resources User's Guide*

While uploading payment details a currency code must be provided. This currency code must match the currency of the payment method.

Balance Types

Balances that are maintained by your third party payroll system can be loaded into the Backfeed tables. Each third party payroll balance that you want to hold in the

Backfeed tables must be defined as a Backfeed balance type in Oracle HRMS before you run the Backfeed.

> **Note:** Backfeed balance types are not the same as Oracle Payroll balance types.

Balance dimensions can be held for any of the balance types you create. The balance dimensions that can be held for each balance type are:

- Year-to-date balance
- Fiscal year-to-date balance
- Period-to-date balance
- Month-to-date balance
- Quarter-to-date balance
- Run amount

You must set up the balance types required by your enterprise before you upload any payroll run data to the HRMS system. When setting up your balance types you can link them to any user defined element input value. This enables you to easily generate reports that can link the balance types to their associated elements.

When uploading monetary balance amounts a currency code must be provided. This currency code must match the currency of the balance or its associated element, as appropriate. One of the following checks is done to ensure the currency of the balance details being loaded is the same as those defined for the balance type:

- If the balance type for the amount being uploaded is associated with an element, a check is done to ensure that the amount being uploaded is in the same currency as the input currency for the associated element.
- If the balance type for the amount being loaded is not associated with an element, a check is done to ensure that the amount being uploaded is in the same currency entered for the balance type.

Balance types must be set up using the Third Party Payroll Balance Types window.

To set up balance types:

1. Enter a display name for the balance type and enter a valid from date. If required, you can also enter a valid to date. The balance type will not be available after this date.
2. Enter an internal name. This is used to identify the balance type internally and must be unique within the Business Group.
3. Enter a category if required. This can be used to group balance types for reporting purposes. For example, you could group together all balance types relating to employee holidays in a category called Holidays.
4. Do one of the following:
   - Select a user defined element and an input value to link to the balance type. The Currency and Unit fields will be populated according to the element and input values you have selected.
   - Select a unit for the balance type and, if required, a currency.

5. The In Use check box indicates whether a balance type has any balance amounts recorded against it. If it does you are not permitted to change the balance type's currency, units element name or internal name.
6. Save your changes.

APIs

Data is maintained in the Backfeed tables using business process APIs. These are interfaces that enable you to create, update and delete information from the Oracle tables. These APIs call one or more row handlers. Row handlers maintain the data in a single table by validating the data being passed in before allowing it to be created, updated, or deleted. Row handlers should not be called directly.

See the *APIs in Oracle HRMS* technical essay for further details.

We recommend you use Data Pump to upload your third party payroll run data into the Oracle HRMS Backfeed tables. You launch Data Pump as a concurrent program from the Run Reports and Process window. Data Pump will automatically call the appropriate Backfeed APIs.

Setting Up Data Pump

One of the features of Data Pump is the ability to resolve internal id values using other information that has been passed in. Functions have to be created when implementing a Data Pump front end to resolve these ids. These functions will differ for each implementation as each enterprise maps the data in different ways depending on how they have implemented Oracle HRMS.

See the *Oracle HRMS Data Pump* technical essay before you attempt to configure Data Pump.

Configuring the Data Pump Front End

The Generic Payroll Backfeed uses a package called PER_BF_GEN_DATA_PUMP. This contains some functions that are used to resolve the internal system ids, such as payroll_id (the function for this is called get_payroll_id).

The function definitions are delivered in two scripts; pebgendp.pkh and pebgendp.pkb. If you are using Oracle HRMS 11.0 they are located in $PER_TOP/patch/110/sql. If you are using Oracle HRMS 11*i* they are located in $PER_TOP/patch/115/sql.

If the assumptions made by the supplied functions are not appropriate to your enterprise you will have to modify the functions to reflect the way in which you have implemented Oracle HRMS. We recommend that you make a copy of the package and make your changes to the copy.

If you do not need to alter any of the parameters in the generic functions, but need to change the body of the function, you can do this and run your amended version against your database. To do this you must navigate to the directory containing your configured script and enter the following:

```
sqlplus <apps_username>/<apps_pwd>@<database_name> @<package_body
_name.pkb>
```

If, however, you need to change the parameters in the functions, or add new functions, as well as altering the package, you will have to run both scripts against the database. To do this navigate to the directory containing your configured scripts and enter the following:

```
sqlplus <apps_username>/<apps_pwd>@<database_name> @<package_head
er_name.pkh>

sqlplus <apps_username>/<apps_pwd>@<database_name> @<package_body
_name.pkb>
```

You must also run the Data Pump Meta-Mapper. This regenerates the Data Pump APIs and views specific to the Third Party Payroll Backfeed interface. For more information on how to do this, and other Data Pump functionality that you may want to use, please refer to the *Oracle HRMS Data Pump* technical essay.

If you do make any changes to the parameters in the supplied generic functions, or add any new functions, you will also need to configure the Data Uploader front end.

See: Configuring the Data Uploader Front End, page 2-335

Deciding Which Upload Option to Use

In order to use Data Pump to upload the third party payroll run data into the Backfeed tables you must first get this data into the Data Pump batch tables. There are two alternative approaches to achieving this:

- Use APIs generated by the Data Pump Meta-Mapper

  If you decide to use this option you will need to write a PL/SQL program to read your payroll results data and insert it into the Data Pump batch tables using the Data Pump APIs.

- Use Data Uploader

  If you decide to use this option you will need to format your payroll run results data file into a flat file in a format that is readable by the Data Uploader.

You must decide which is the best approach for you based on your technical resources and the source of your payroll results data.

Setting Up Data Uploader

Data Uploader takes data held in tab delimited text files and uploads it to the Data Pump batch tables using the packages and views created when Data-Pump Meta-Mapper is run. To use Data Uploader you must get your payroll run data into tab delimited files of the format required by Data Uploader. To help you format your payroll run data files, a Microsoft Excel workbook called bfexampl.xls has been supplied. This shows how your data must be set out. Once formatted you can use the Save Sheets macro to export the data held in the Excel worksheets into the tab delimited text files used by Data Uploader. This, and the Load Sheets macro are supplied in the bfmacros.xls file.

Using Excel to Create Files

Although you can use the Excel macros during the early stages of a Backfeed implementation to create files that can be read by Data Uploader, you should stop using Excel once you are using Backfeed in a production environment. We suggest that you automate the creation of the tab delimited Data Uploader files, instead.

You can continue to use Excel for debugging purposes, if the files are small enough for Excel to handle, if problems occur when running the Data Uploader part of Backfeed.

Example Files

The example files consist of:

bfexampl.xls

- **Header Sheet**. This contains basic information for the workbook such as the individual worksheet names.
- **Payroll Run Sheet**. This holds details relating to the entire run such as the processing date. This contains data to be used by the the create_payroll_run API.
- **Balance Amounts Sheet**. This holds the employee balance details for the run defined in the Payroll Run worksheet. This contains data to be used by the the create_balance_amount API.
- **Payment Details Sheet**. This holds the employee payment details for the run defined in the Payroll Run worksheet. This contains data to be used by the create_payment_details API.
- **Processed Assignments Sheet**. This holds the processed assignment details for a particular employee assignment relating to the run defined in the Payroll Run worksheet. This contains data to be used by the create_processed_assignment API.

bfmacros.xls

- **Save Sheets Macro**. This is a macro that saves the individual sheets in the workbook as individual tab delimited text files
- **Load Sheets Macro**. This is a macro that loads the individual text files based on the Header file.

Header Sheet

The Header Sheet contains information about the complete set of data that is to be uploaded. It defines standard information such as batch name and date, and also specifies the files that are to be used in this upload.

You must enter a batch name that will uniquely identify this upload. You will be asked for this batch name when you run the Data Pump process.

The text between the Files Start and Files End rows are the file names for the individual sheets. The first column contains the name of the sheet, and the second column contains the name of the text file. This is the name that the related sheet will be saved as, or uploaded from if you use the macros.

Payroll Run Sheet

Every payroll run has information that relates to the entire run such as processing date, periods start and end dates, and a unique identifier for the run. This worksheet contains this type of information.

At the top of the sheet, between the Descriptor Start and Descriptor End columns, the details relating to the run are held. It is likely that these will remain the same for all your data uploads.

The User Key row contains an entry that allows the Data Uploader and Data Pump functionality to uniquely refer to the payroll run that is being inserted from other sheets that need this reference, such as the Balance Amounts Sheet and the Payment Details Sheet. The default entry for this is %$Business Group%:payroll_identifier. You should not need to change this as the combination of Business Group ID and the payroll identifier should always uniquely identify a payroll run.

The ID column is the way the Data Uploader identifies a row in the spreadsheet and can be used by other sheets in the same workbook to refer to a particular row. In this case, both the Balance Amount Sheet and the Payment Details Sheet have a column

called Payroll_run_id that will refer to the row in this sheet. Each row of your data should have a different, sequential number in the ID column.

Balance Amounts Sheet

The Balance Amounts worksheet holds the balance information relating to each employee for a particular payroll run.

The row beneath the the Data Start row contains the column titles of the API. Your payroll run balance amount details for each employee need to go between this row and the Data End row. A currency code must be provided for all monetary amounts.

The ID column needs to be populated with sequential numbers starting from 1.

The column named Payroll_Run_id refers to the ID column in the Payroll Run worksheet. This number will be the same for all the rows in the payroll run.

Payment Details Sheet

The Payment Amount Sheet holds the payment details for each employee processed in a payroll run.

The row beneath the the Data Start row contains the column titles of the API. Your payment details for a particular run need to go between this row and the Data End rows. You must provide a currency code for all monetary amounts.

The ID column needs to be populated with sequential numbers starting from 1.

The column named Payroll_Run_id refers to the ID column in the Payroll Run worksheet. This number will be the same for all the rows in the payroll run.

Processed Assignments Sheet

The Processed Assignment Sheet holds the assignment details for each employee processed on a payroll run.

The row beneath the Data Start row contains the column titles of the API. Your processed assignment details for a particular employee and payroll run need to go between this row and the Data End row.

The ID column needs to be populated with sequential numbers starting from 1.

The column named Payroll_Run_id refers to the ID column in the Payroll Run Worksheet. This number will be the same for all the rows in the payroll run.

This worksheet is only required if additional information is held within the processed assignment descriptive flexfield. If there is no additional information then the processed assignment will be created by the balance amount api or payment detail api.

If this worksheet is not required (for reasons mentioned above) then the name and text file for processed assignment must be removed from the header sheet.

Save Sheets Macro

This Excel macro saves the individual Worksheets as tab delimited text files. The name of each text file, with the exception of the Header Sheet, is held in the Header Sheet. You are prompted to enter a name for the Header Sheet when you run the macro.

Load Sheets Macro

To use this macro you must have a tab delimited text file of your Header Sheet. This macro loads the text files specified in the Header Sheet as worksheets into workbook from which the macro was run. The text files to be loaded must be in the same directory as the selected Header Sheet text file.

Specifying the Upload Directories for Data Uploader

You must specify the location in which files to be imported using the Data Uploader must be placed. The following steps describe the tasks that must be completed to do this:

1. In the initialization file for the database, your Database Administrator must specify the directory that will hold the files to be uploaded. This is done by including the path of the required directory in the UTL_FILE_DIR parameter.
2. Your System Administrator must enter the full path to this directory in the HR: Data Exchange Directory user profile option. Use the System Profile Values window. You can set this profile option at site, application and responsibility level, depending on the security you want to impose.

Configuring the Data Uploader Front End

The generic Data Uploader parameters are defined in a script called pedugens.sql. It is separated into different sections for creating parameters for Payroll Run, Balance Amounts etc.

If you are using Oracle HRMS 11.0 this script is located in $PER_TOP/patch/110/sql. If you are using Oracle HRMS 11*i* it is located in $PER_TOP/patch/115/sql.

If you have changed the parameters in the Data Pump functions to resolve the system ids, or added new functions and used Meta-Mapper to regenerate the Data Pump APIs, you must include a column containing the data specified in the new parameters in the appropriate sheet of your Excel upload workbook. See: Creating an Upload Workbook, page 2-336.

You must then amend the pedugens.sql script to map the new data in the Excel column to the API used by the Data Uploader.

The following is an example of code that is used to create the Data Uploader mapping details for the create_balance_amount API:

```
HRDU_DO_API_TRANSLATE.hrdu_insert_mapping(
          p_api_module        => 'create_balance_amount',
          p_column_name       => 'balance_type_name',
          p_mapped_to_name    => 'p_balance_type_name');
```

The p_api_module parameter identifies which Microsoft Excel worksheet holds the data that will be uploaded using this api. In this case it is create_balance_amount. The p_column_name parameter passes in the associated Excel worksheet column name, in this case, balance_type_name. The p_mapped_to_name parameter passes the Data Pump view column that is to be associated with the Excel worksheet, in this case p_balance_type_name.

You will need to add an insert statement for any new columns that you have added to the upload workbook, whether they are in existing or new functions.

Using Backfeed to Upload Payroll Run Results

To upload payroll run results using Backfeed, follow this sequence of tasks:

1. Save the payroll run results from your third party payroll system into a text file.
2. Create an upload workbook.

   See: Creating an Upload Workbook, page 2-336
3. Format the payroll run data into the format required by Data Uploader.

See: Formatting the Payroll Run Data into the Format Required by Data Uploader, page 2-336

4. Use Data Uploader concurrent process to load the information from the text file into the Data Pump batch tables.

   See: Running Data Uploader, page 2-338

   Note: If you decided not to use Data Uploader to load the payroll run data into the Data Pump Batch table, but to write a PL/SQL program that uploads the data using the APIs generated by the Data Pump Meta-Mapper, you should ignore steps 3 and 4.

5. Run the Data Pump concurrent process to upload the data from the Data Pump batch tables into the Backfeed tables.

   See: Running Data Pump, page 2-338

Creating an Upload Workbook

You must create an upload workbook based on the bfexampl.xls file that meets the need of your enterprise before you use Data Uploader.

You can change the names of the files specified in the Header Sheets to whatever you would like the files saved as. For example, if you want to keep a file record of all the payroll runs you have uploaded into the Backfeed tables, you may want to prefix the files with the payroll identifier for the run they relate to.

You can amend the layout of the worksheet and remove any unnecessary worksheets as detailed below:

If you are only using the balance detail functionality and not the payment detail functionality, you can remove the line from the Header Sheet detailing the Payment Detail sheet and delete the Payment Detail Sheet. You can also remove the Balance Details functionality in the same way if you do not want to use it.

If there are any non-essential columns, such as check_type or ftd_amount, that you are not using, you can remove them from the worksheet. Ensure that you do not remove any columns that will prevent the data being loaded via Data Pump. For example, you cannot remove the ID or payroll identifier columns because these are essential to the operation of both Data Pump and Data Uploader.

As well as this, you can change the order of the data columns (with the exception of the ID column) to suit your preference. You must also add any new columns required by changes you have made to your Data Pump front end.

See: Configuring the Data Pump Front End, page 2-331

Formatting the Payroll Run Data into the Format Required by Data Uploader

There are a number of methods that you can use to format the payroll run data into the format required by Data Uploader. You can choose the method that suits the working practices of your enterprise.

One method would be to format your payroll run data using your operating system tools and load it into another spreadsheet. You can then cut and paste it into position in the upload workbook and use the Save Sheets macro to save the worksheets into individual tab delimited text files.

Alternatively, you could save the upload worksheets without any data in using the Save Sheets macro, and use operating system tools to put the data into the correct position. To

ensure that the data is correctly formatted you could use the Load Sheets macro to reload the data into Excel so that you can view it. Reloading the data into Excel to check it is not necessary for correct operation of the Data Uploader tool, but it is recommended.

For worksheets with minimal data, another method would be to enter the data manually into Excel and then save it using the Save Sheets macro.

Using the Load Sheets Macro

The Load Sheets macro enables you to load the text files specified in a tab delimited text file version of your Header Sheet into a workbook. The files are loaded from the same directory in which the header text file is stored

To run the Load Sheets macro

1. Ensure you have a version of your Header Sheet, in the same format as the first worksheet in bfexampl.xls, saved as a tab delimited text file. This defines the text files you want to load and the names of the Excel worksheets that should be created when they are loaded.
2. Ensure that the text files you want to upload are stored in the same directory as the Header Sheet text file.
3. Open the workbook into which you want to load the files. If this workbook does not contain the Load Sheets macro you must copy it in from another workbook.
4. Choose Macro from the Tools menu and select the Load Sheets macro in the displayed Macros window.
5. Enter the path of the directory that contains the Header Sheet text file and choose OK.

   Note: The last character you must enter in this path must be a "\", for example C:\upload\.

6. Enter the name of the Header Sheet text file and choose OK. The files are loaded into the workbook.

   Note: When the files are loaded into the workbook the name of the worksheet containing the header information, i.e. the first worksheet, will always be header_sheet.

Using the Save Sheets Macro

The Save Sheets macro enables you to save a multiple sheet Excel workbook into corresponding tab delimited text files. Each text file will be given the name specified in the Header Sheet and will be saved in the specified directory. The first worksheet in the workbook, the Header Sheet, will create the header file that will be used by Data Uploader.

To run the Save Sheets macro:

1. Ensure that the required Excel workbook is open. If this workbook does not contain the Save Sheets macro you must copy it in from another workbook.
2. Ensure the worksheet containing the Header information is called header_sheet. If it is not you must rename this worksheet or the macro will fail.
3. Choose Macro from the Tools menu and select the Save Sheets macro in the displayed Macros window.

4. Enter the path of the directory in which you want to save the text files. This should be the directory defined by your System Administrator during the set up of Backfeed. Choose OK.

   Note: The last character you must enter in this path must be a "\", for example C:\upload\.

5. Enter a name for the header file. This will default to the name of the first worksheet in the workbook. You will need to specify this file when you run the Data Uploader process. Choose OK.

Running Data Uploader

The Data Uploader takes the information held in the text files you have created and loads them into the Data Pump batch tables. The files that are used in each upload are defined by the header file you select when running the HR Data Uploader concurrent process.

Note: You can load the payroll run data into the Data Pump tables using another method if you desire.

Once the setup tasks have been completed you run the Data Uploader in the Submit Requests window.

To run the Data Uploader process:

1. Ensure that the files you want to upload are in the directory specified during the Backfeed setup by your Database and System Administrators.
2. In the Submit Requests window, select the HR Data Uploader concurrent process.
3. Enter the file name of the header file you want to use and choose submit.

Tracking Errors Using Data Uploader

If any errors are detected whilst using Data Uploader, you must view the concurrent request log file for more information.

Running Data Pump

Once you have the payroll run data in the Data Pump batch tables you must run the Data Pump Engine concurrent process to upload the data into the Backfeed tables.

To run the Data Pump Engine concurrent process:

1. Select the Data Pump Engine concurrent process.
2. Enter the required batch name and indicate whether you want the process to be validated.

   The batch name will be of the form: <batch name>-<batch ID> where batch name relates to the batch name entered in the header file and batch ID is the internally allocated ID. For example:

   Week12-1234

3. Choose Submit.

For information on finding and fixing errors in Data Pump see the *Oracle HRMS Data Pump* technical essay.

Viewing Third Party Payroll Results in Oracle HRMS

After uploading your third party payroll results into the Backfeed tables, you can view them by:

- Employee (in the Third Party Payroll Run Employee Results window)
- Organization, job, grade, group, position, or location (in the Third Party Payroll Run Organization Results window)

These windows each contain two folders, Balance Details and Payment Details, that enable you to display the information you require using the standard folder utilities.

To query payroll run details using the Find Third Party Payroll Run Employee Results window:

1. Do one or any number of the following:
   - Enter a full or partial query on the person's name. Where a prefix has been defined for the person, a full name query should be in the format 'Maddox, Miss Julie'.
   - Enter a query on employee number, assignment number, payroll, or payroll identifier.
   - Specify an earliest and latest date for payroll period start and end dates, and payroll process dates. This means that you can retrieve a range of payroll run results.
2. Choose the Find button.

   The payroll run details found by the query are displayed in the Third Party Payroll Run Employee Results window. If the query found more than one record, you can use the [Down Arrow] key or choose Next Record from the Go menu to display the next record.

To query payroll run details using the Find Third Party Payroll Run Organization Results window:

1. Do one or any number of the following:
   - Enter a query on organization, people group, job, position, grade, or location.
   - Enter a query on payroll, or payroll identifier.
   - Specify an earliest and latest date for payroll period start and end dates, and payroll process dates. This means that you can retrieve a range of payroll run results.
2. Choose the Find button.

   The payroll run details found by the query are displayed in the Third Party Payroll Run Organization Results window. If the query found more than one record, you can use the [Down Arrow] key or choose Next Record from the Go menu to display the next record.

To view third party payroll run results:

1. Query the required information using the Find Third Party Payroll Run Employee Results window or the Find Third Party Payroll Run Organization Results window.
   - If you queried using the Find Third Party Payroll Run Employee Results window, details about the employee and the payroll run are displayed, including additional flexfield information.

- If you queried using the Find Third Party Payroll Run Organization Results window, details about the payroll run are displayed, including additional flexfield information. The find window remains open in the background so that you can refer to it to see the query that has retrieved the displayed results.

2. Choose the Balance Details alternative region. This displays all the balance information relating to the displayed employee and payroll run such as run amount, financial year to date amount, and element name. Any additional flexfield information will also be displayed here. You can use standard folder tools to control the data that is displayed in this folder.
3. Choose the Payment Details alternative region. This displays all the payment information relating to the displayed employee and payroll run such as check number, payment date, and amount. Any additional flexfield information will also be displayed here. You can use standard folder tools to control the data that is displayed in this folder.

Index

D

E

F

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W