Implementing Oracle HRMS

RELEASE 11*i*

December 1999

ORACLE®

Implementing Oracle HRMS Release 11*i*

The part number for this book is A73313–01.

Contributing Authors: Louise Raffo, Juliette Fleming and John Cafolla.

Contributors: Bill Kerr, John Thuringer and Janet Harrington–Kuller.

Contents

Preface

Audience for This Guide

Welcome to Release 11*i* of Implementing Oracle® HRMS.

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, we suggest you attend one or more of the Oracle HRMS training classes available through Oracle University.
- The Oracle Applications graphical user interface.

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

See Other Information Sources for more information about Oracle Applications product information.

How To Use This Guide

This guide contains the information you need to implement Oracle HRMS.

This preface explains how this user guide is organized and introduces other sources of information that can help you. This guide contains the following information:

- Part I provides you with implementation information:
  - Chapter 1 is designed to help you plan your implementation. It includes flowcharts showing the major stages of an implementation and a summary checklist.
  - Chapter 2 contains a step–by–step implementation sequence summarizing the decisions and tasks required for each stage.
- Part II provides technical essays which may be required by the implementation team for initial data loading, customizing Oracle HRMS, or integrating it with other applications or processes. Chapters 3 to 13 include essays on these subjects:
  - Chapter 3: APIs in Oracle HRMS
  - Chapter 4: Oracle HRMS Data Pump
  - Chapter 5: DateTrack

- Chapter 6: FastFormula
- Chapter 7: Extending Security in Oracle HRMS
- Chapter 8: Batch Element Entry
- Chapter 9: Validation of Flexfield Values
- Chapter 10: Payroll Processes
- Chapter 11: Payroll Archive Reporter Process
- Chapter 12: Balances in Oracle Payroll
- Chapter 13: Payroll Advice Report (UK Only)

- Appendix A describes any post install steps you must perform before you implement Oracle HRMS.

Finding Out What's New

From the HTML help window for Oracle HRMS, choose the section that describes new features or what's new from the expandable menu. This section describes:

- New features in 11*i*. This information is updated for each new release of Oracle HRMS.
- Information about any features that were not yet available when this guide was printed. For example, if your system administrator has installed software from a mini pack as an upgrade, this document describes the new features.

Other Information Sources

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

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

Online Documentation

All Oracle Applications documentation is available online (HTML and PDF). The technical reference guides are available in paper format only. Note that the HTML documentation is translated into over twenty languages.

The HTML version of this guide is optimized for onscreen reading, and you can use it to follow hypertext links for easy access to other HTML guides in the library. When you have an HTML window open, you can use the features on the left side of the window to navigate freely throughout all Oracle Applications documentation.

- You can use the Search feature to search by words or phrases.
- You can use the expandable menu to search for topics in the menu structure we provide. The Library option on the menu expands to show all Oracle Applications HTML documentation.

You can view HTML help in the following ways:

- From an application window, use the help icon or the help menu to open a new Web browser and display help about that window.
- Use the documentation CD.
- Use a URL provided by your system administrator.

Your HTML help may contain information that was not available when this guide was printed.

Related User Guides

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.

User Guides Related to All Products

Oracle Applications User Guide

This guide explains how to navigate the system, enter data, and query information, and introduces other basic features of the GUI available with this release of Oracle HRMS (and any other Oracle Applications product).

You can also access this user guide online by choosing "Getting Started and Using Oracle Applications" from the Oracle Applications help system.

Oracle Alert User Guide

Use this guide to define periodic and event alerts that monitor the status of your Oracle Applications data.

Oracle Applications Implementation Wizard User Guide

If you are implementing more than one Oracle product, you can use the Oracle Applications Implementation Wizard to coordinate your setup activities. This guide describes how to use the wizard.

Oracle Applications Developer's Guide

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

Oracle Applications User Interface Standards

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.

User Guides Related to This Product

Using Oracle HRMS – The Fundamentals

This user guide explains how to setup and use enterprise modeling, organization management, and cost analysis. It also includes information about defining payrolls.

Managing People Using Oracle HRMS

Use this guide to find out about using employee management, recruitment activities, career management, and budgeting.

Running Your Payroll Using Oracle HRMS

This user guide provides information about wage attachments, taxes and social insurance, the payroll run, and other processes.

Managing Compensation and Benefits Using Oracle HRMS

Use this guide to learn about compensation setup, entry and analysis, setting up basic, standard and advanced benefits, salary administration, and absence management and PTO accruals.

Customizing, Reporting and System Administration in Oracle HRMS

This guide provides information about extending and customizing Oracle HRMS, managing security, auditing, information access, and letter generation.

Implementing Oracle Self–Service Human Resources (SSHR)

This guide provides information about setting up the self–service human resources management functions for managers and employees. Managers and employees can then use an intranet and Web browser to have easy and intuitive access to personal and career management functionality.

Using Oracle FastFormula

This guide provides information about writing, editing, and using formulas to customize your system. Oracle FastFormula provides a simple way to write formulas using English words and basic mathematical functions. For example, Oracle FastFormula enables you to specify elements in payroll runs or create rules for PTO and accrual plans.

Using Oracle Training Administration (OTA)

This guide provides information about how to set up and use Oracle Training Administration to facilitate your training and certification business.

Using Oracle SSP/SMP (UK Only)

This guide provides information about setting up and using Oracle SSP/SMP to meet your statutory sick pay and statutory maternity pay obligations.

Using Application Data Exchange and Hierarchy Diagrammers

This guide provides information about using Application Data Exchange to view HRMS data with desktop tools, and upload revised data to your application. This guide also provides information about using Hierarchy Diagrammers to view hierarchy diagrams for organizations and positions.

Oracle Business Intelligence System Implementation Guide

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

BIS 11*i* User Guide Online Help

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

Using Oracle Time Management

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

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.

Installation and System Administration Guides

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, and major issues, for Applications–wide features such as Business Intelligence (BIS), languages and character sets, and self–service 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 One–Hour Install, which minimizes the time it takes to install Oracle Applications and the Oracle 8*i* Server technology stack by automating many of the required steps. This guide contains instructions for using Oracle One–Hour 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 in general and lists database upgrade and product–specific upgrade tasks. You must be at either 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.

Using the AD Utilities

Use this guide to help you run the various AD utilities, such as AutoInstall, AutoPatch, AD Administration, AD Controller, Relink, and others. It contains how–to steps, screenshots, and other information that you need to run the AD utilities.

Oracle Applications Product Update Notes

Use this guide as a reference if you are responsible 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 11i. It includes new features and enhancements and changes made to database objects, profile options, and seed data for this interval.

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 processing.

Oracle HRMS Applications Technical Reference Guide

This reference guide contains database diagrams and a detailed description of database tables, forms, reports, and programs for Oracle HRMS, including Oracle HRMS and related applications. This information helps you convert data from your existing applications, integrate Oracle HRMS with non–Oracle applications, and write custom reports for Oracle HRMS.

You can order a technical reference guide for any product you have licensed. Technical reference guides are available in paper format only.

Oracle Workflow Guide

This guide explains how to define new workflow business processes as well as customize existing Oracle Applications–embedded workflow processes. You also use this guide to complete the setup steps necessary for any Oracle Applications product that includes workflow–enabled processes.

Training and Support

Training

We offer a complete set of training courses to help you and your staff master Oracle Applications. We can help you develop a training plan that provides thorough training for both your project team and your end users. We will work with you to organize courses appropriate to your job or area of responsibility.

Training professionals can show you how to plan your training throughout the implementation process so that the right amount of information is delivered to key people when they need it the most. You can attend courses at any one of our many Educational Centers, or you can arrange for our trainers to teach at your facility. We also offer Net classes, where training is delivered over the Internet, and many multimedia–based courses on CD. In addition, we can tailor standard courses or develop custom courses to meet your needs.

Support

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

Do Not Use Database Tools to Modify Oracle Applications Data

We STRONGLY RECOMMEND that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications tables, unless we tell you to do so in our guides.

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 forms, you might 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 forms to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. But, 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.

About Oracle

Oracle Corporation develops and markets an integrated line of software products for database management, applications development, decision support and office automation, as well as Oracle Applications. Oracle Applications provides the E–Business Suite, a fully integrated suite of more than 70 software modules for financial management, internet procurement, business intelligence, supply chain management, manufacturing, project systems, human resources and sales and service management.

Oracle products are available for mainframes, minicomputers, personal computers, network computers, and personal digital assistants, enabling organizations to integrate different computers, different operating systems, different networks, and even different database management systems, into a single, unified computing and information resource.

Oracle is the world's leading supplier of software for information management, and the world's second largest software company. Oracle offers its database, tools, and application products, along with related consulting, education and support services, in over 145 countries around the world.

Your Feedback

Thank you for using Oracle HRMS and this user guide.

We value your comments and feedback. This guide contains a Reader's Comment Form you can use to explain what you like or dislike about Oracle HRMS or this user guide. Mail your comments to the following address or call us directly at (650) 506–7000.

Oracle Applications Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Or, send electronic mail to **appsdoc@us.oracle.com**.

PART

I

Implementation

This part of *Implementing Oracle HRMS* provides you with the information you need for your Oracle HRMS implementation. It includes information about:

- Planning your implementation
- Performing implementation steps

CHAPTER

1 Planning Your Implementation

Implementation Steps

The flexibility of Oracle HRMS enables you develop an implementation project plan that meets your own specific business needs for both Oracle Human Resources, Oracle Payroll, Oracle Advanced Benefits 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 initially 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 customize 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 a HRMS system that matches your specific business needs in the most efficient and cost–effective manner.

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.

Consult your *Oracle Applications Installation Manual* 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 A – 2.

Oracle Applications Implementation Wizard

If you are implementing more than one Oracle Applications product, we recommend that you use the Oracle Application Implementation Wizard to coordinate your setup activities. The Implementation Wizard guides you through the setup steps for the applications you have installed, suggesting a logical sequence that satisfies cross–product implementation dependencies and reduces redundant setup steps.

You can use the Implementation Wizard to see a graphical overview of setup steps, read online help for a setup activity and open the appropriate setup window. You can also document your implementation, for further reference and review, by using the Wizard to record comments for each step.

See: *Oracle Applications Implementation Wizard User's Guide*.

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.

> **Note:** Refer to the Post Install Steps: page A – 2 to see any steps you must perform before you implement Oracle HRMS.

- ❑ Administration: page 2 – 2 (Required)

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

- ❑ Work Structures: page 2 – 19 (Required)

  Includes organizations, jobs, positions, grades and payrolls

- ❑ Compensation and Benefits: page 2 – 27 (Optional)

  Includes compensation elements, input value validation, balances, formulas, salary administration, absence management/accruals of paid time off and element sets.

- ❑ Total Compensation: page 2 – 38 (Optional)

  Includes online benefits services, benefits eligibility, eligibility factors, life events, program setup and flex credits calculations.

- ❑ People and Assignments: page 2 – 49 (Required)

  Includes person types, assignment statuses and special personal information.

- ❑ Specific Business Functions: page 2 – 53 (Optional)

  Includes human resource budgets, evaluation systems and requirements matching.

- ❑ Career and Succession Management: page 2 – 56 (Optional)

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

- ❑ Control: page 2 – 62 (Optional)

  Includes reports, letter generation, customization, task flows, user security, audit requirements and iHelp.

Implementation Flowchart

Some of the steps outlined in this section are Required, and some are Optional. Required with Defaults means that the setup functionality comes with predefined, default values in the database; however, you should review those defaults and decide whether to change them to suit your business needs. If you want or need to change them, you should perform that setup step. You need to perform Optional steps only if you plan to use the related feature or complete certain business functions.

Figure 1 – 1 Implementation Flowchart for Administration

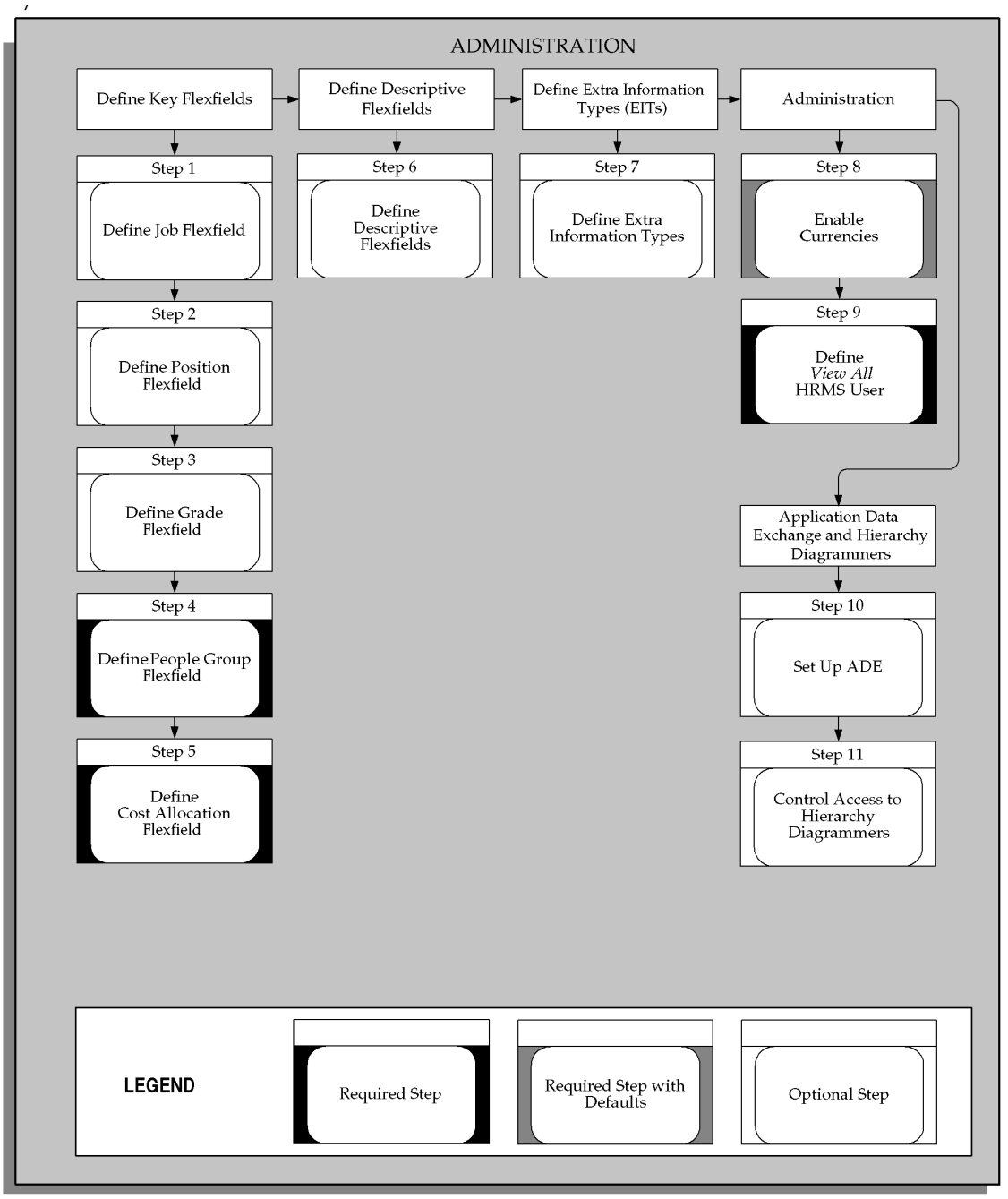


Figure 1 – 2 Implementation Flowchart for Work Structures 1

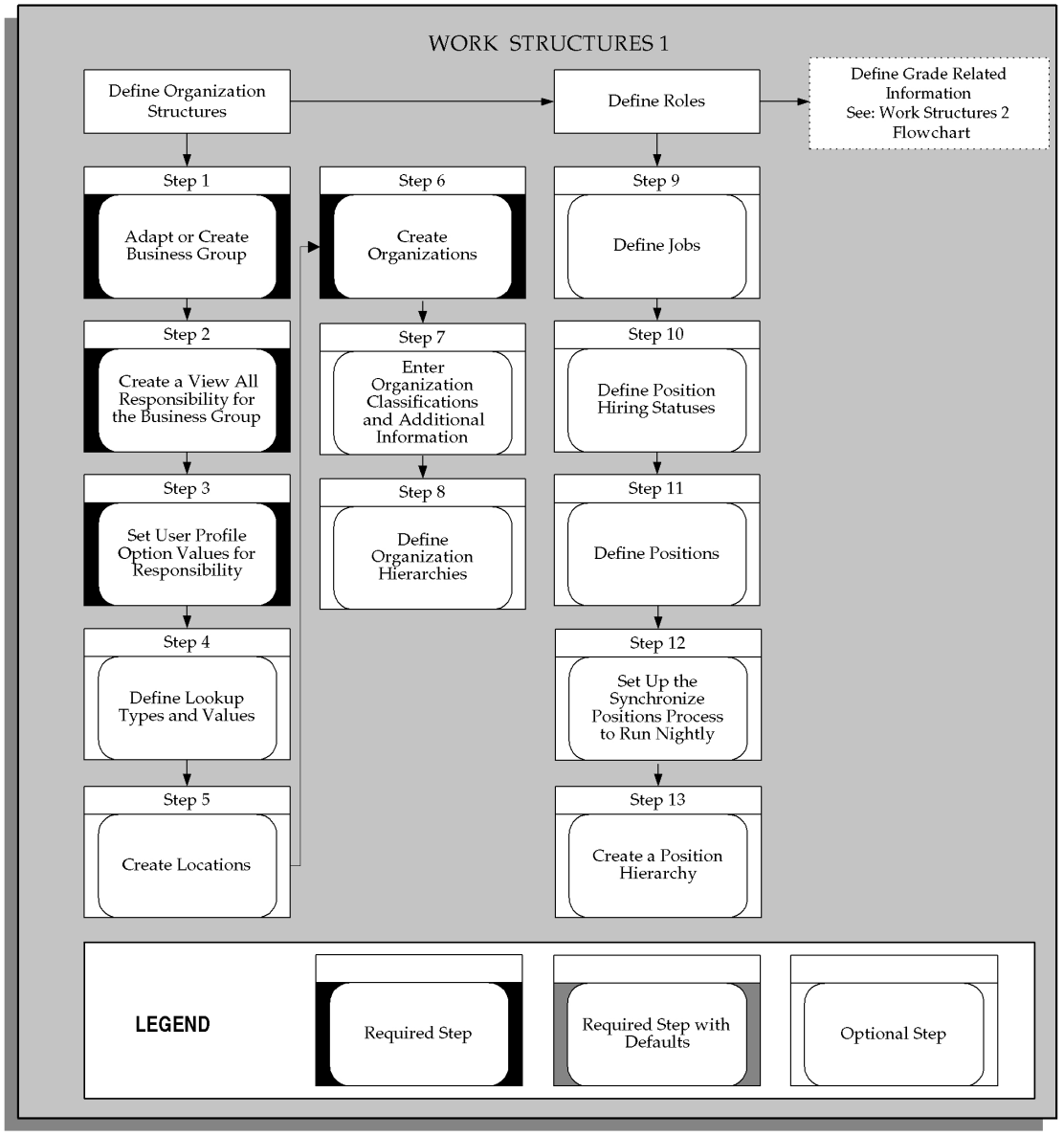


Figure 1 – 3 Implementation Flowchart for Work Structures 2

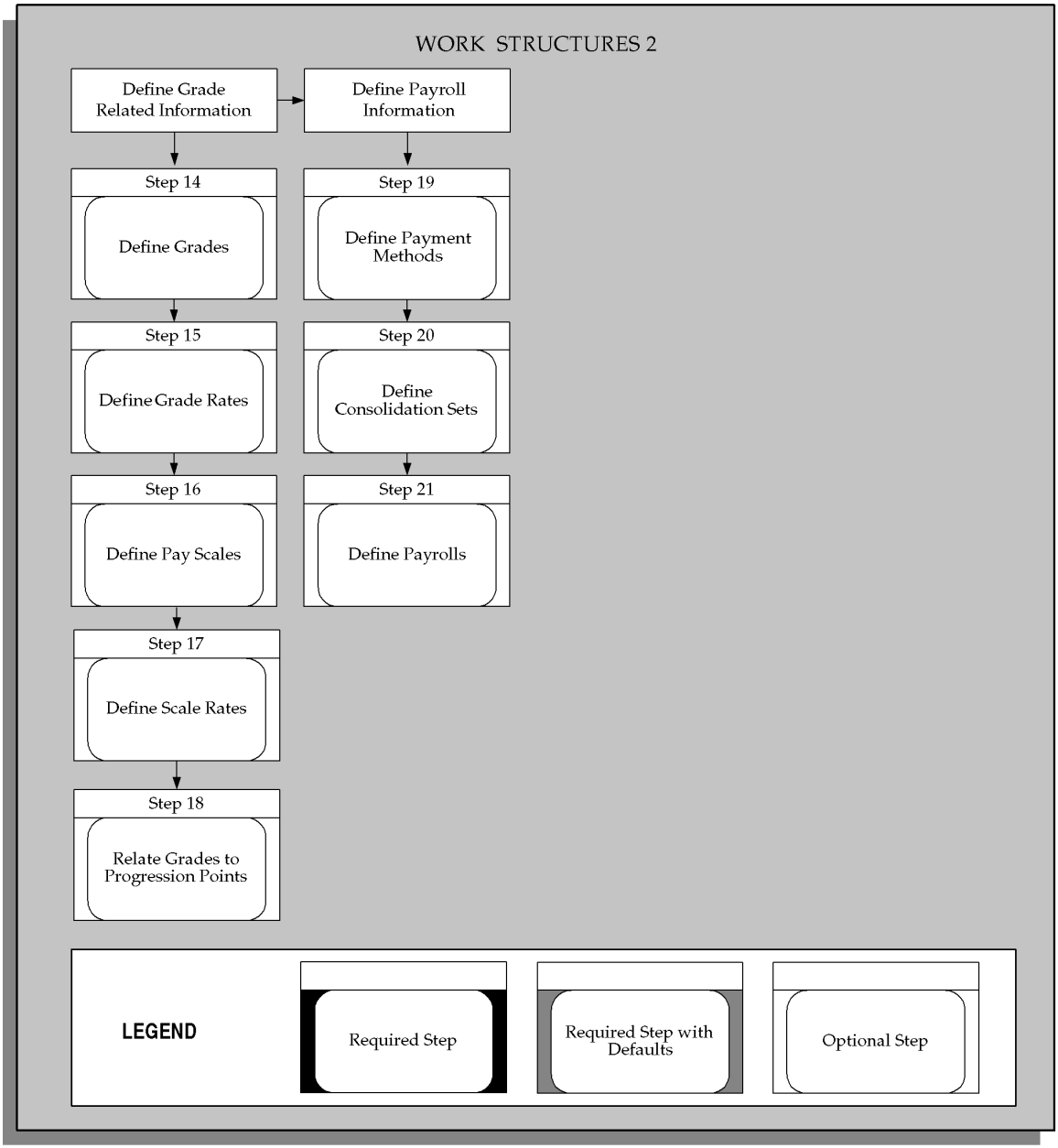


Figure 1 – 4 Implementation Flowchart for Compensation and Benefits 1

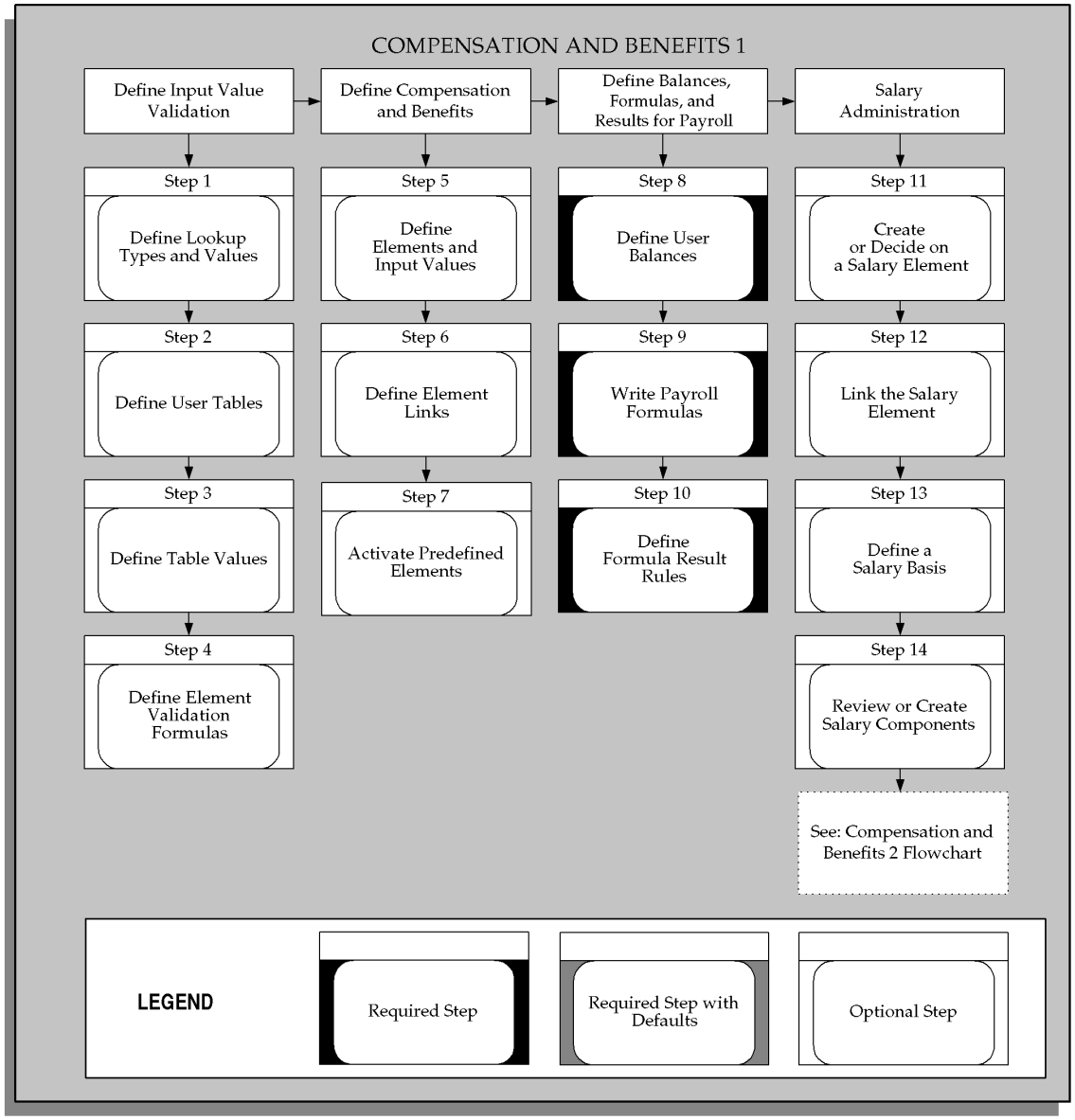


Figure 1 – 5 Implementation Flowchart for Compensation and Benefits 2

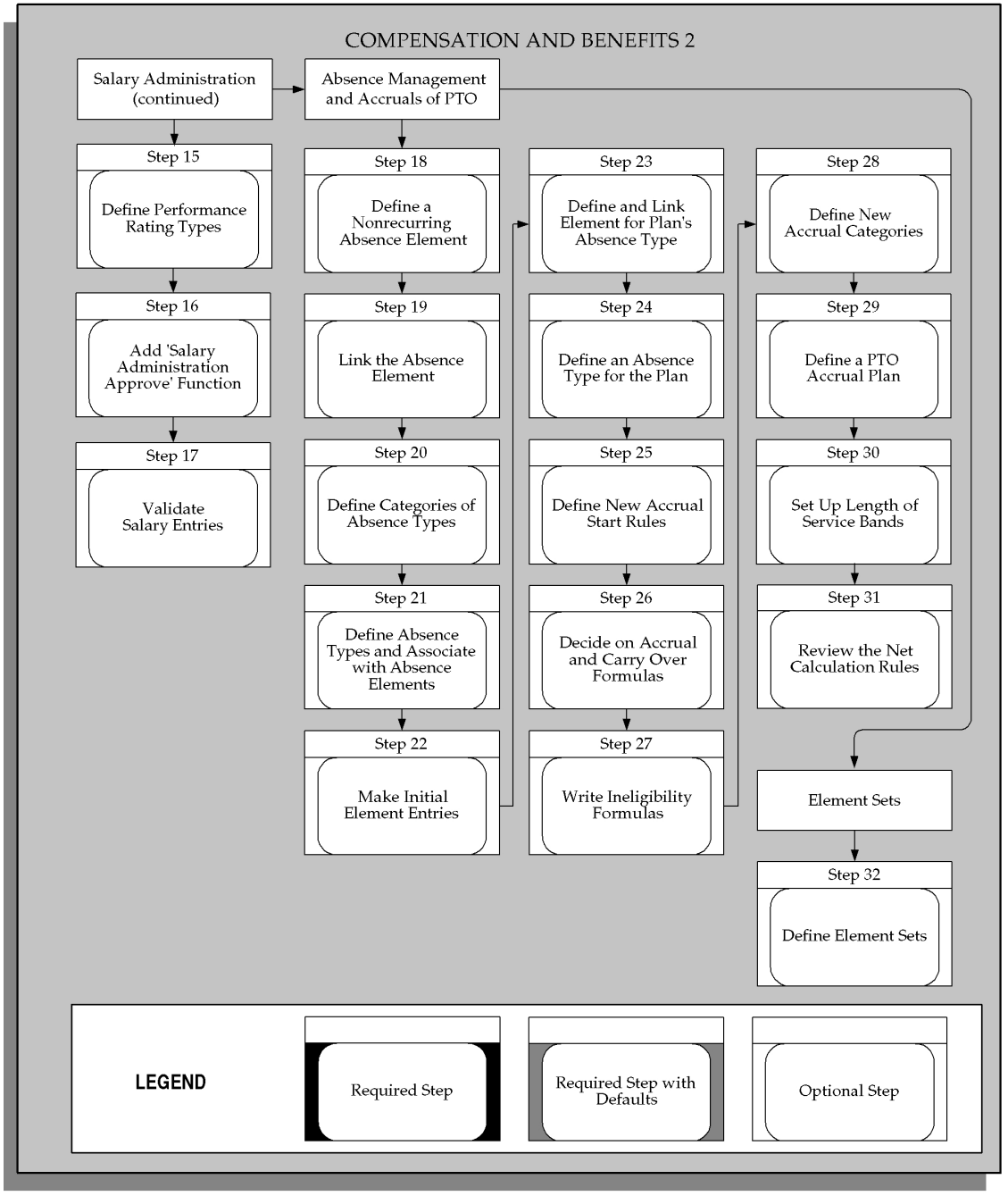


Figure 1 – 6 Implementation Flowchart for Total Compensation 1

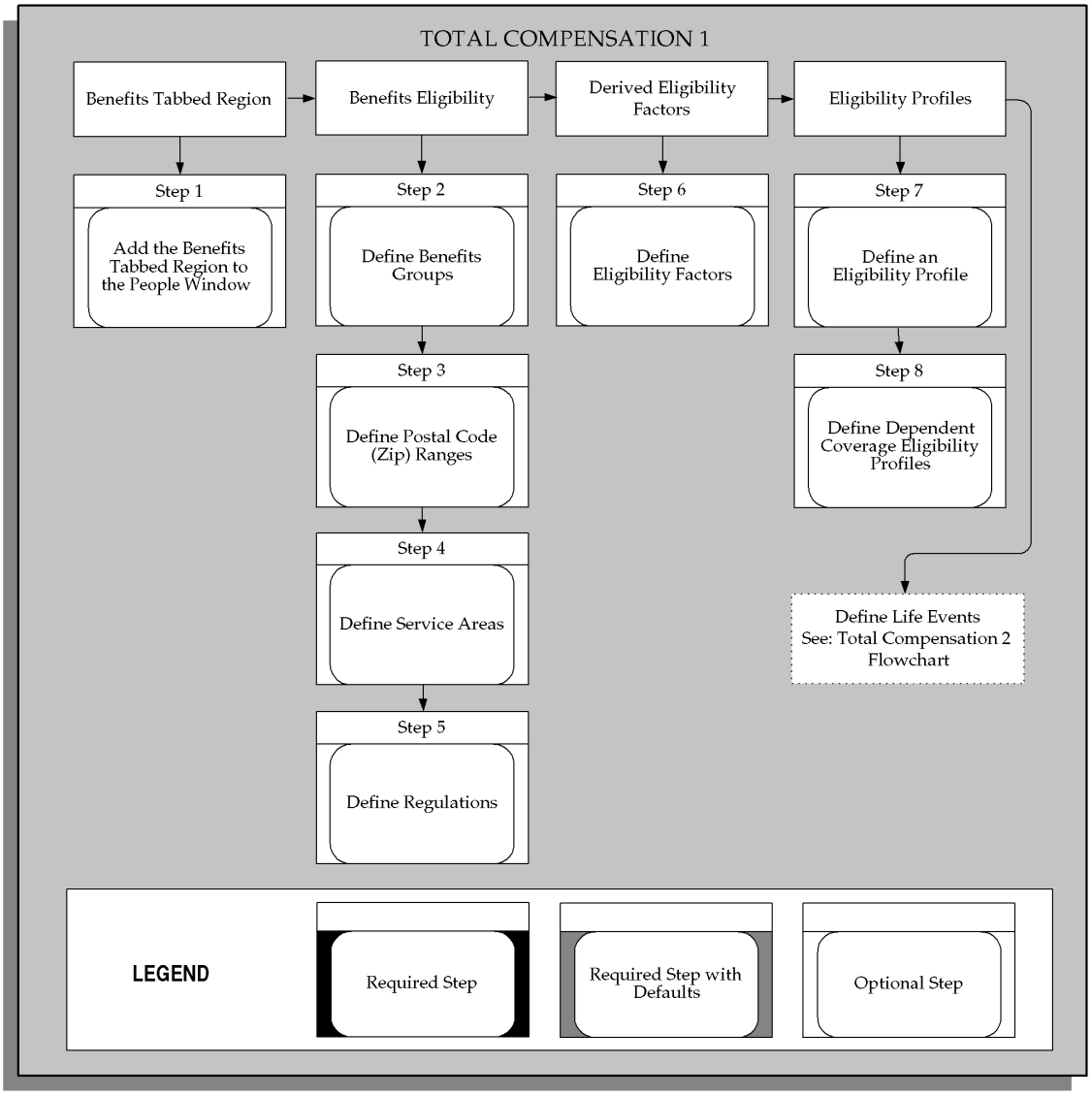


Figure 1 – 7 Implementation Flowchart for Total Compensation 2

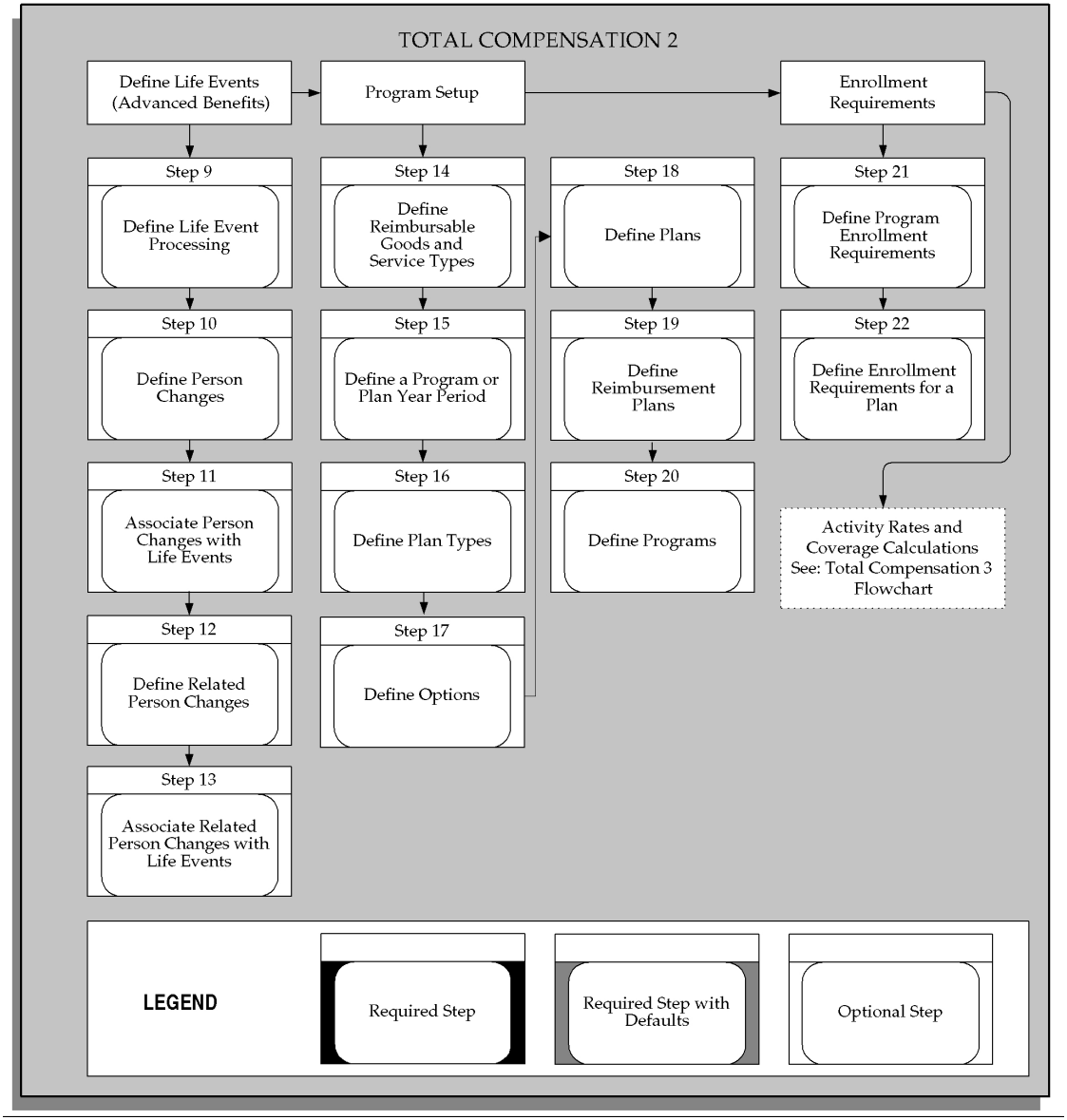


Figure 1 – 8 Implementation Flowchart for Total Compensation 3

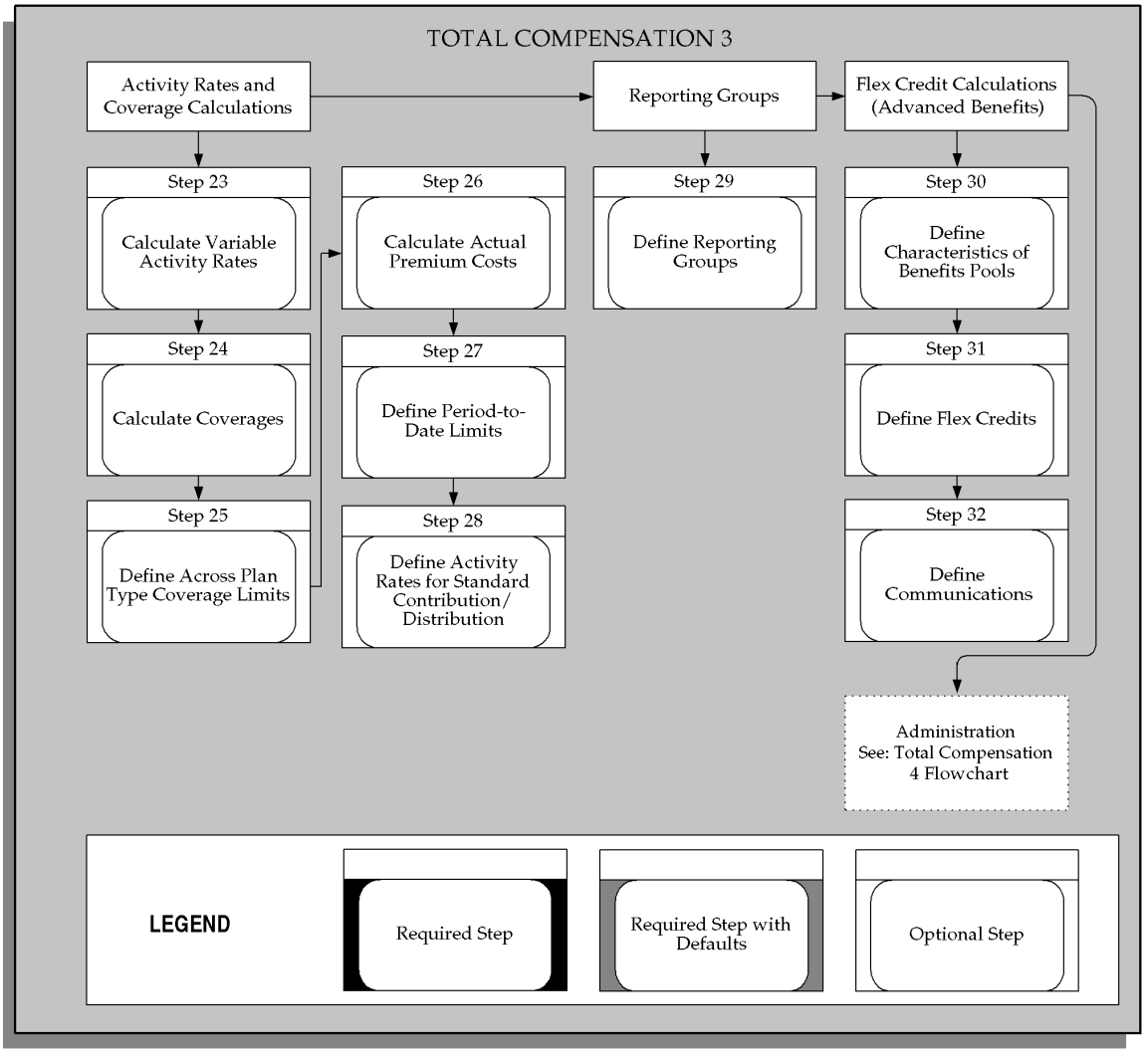


Figure 1 – 9 Implementation Flowchart for Total Compensation 4

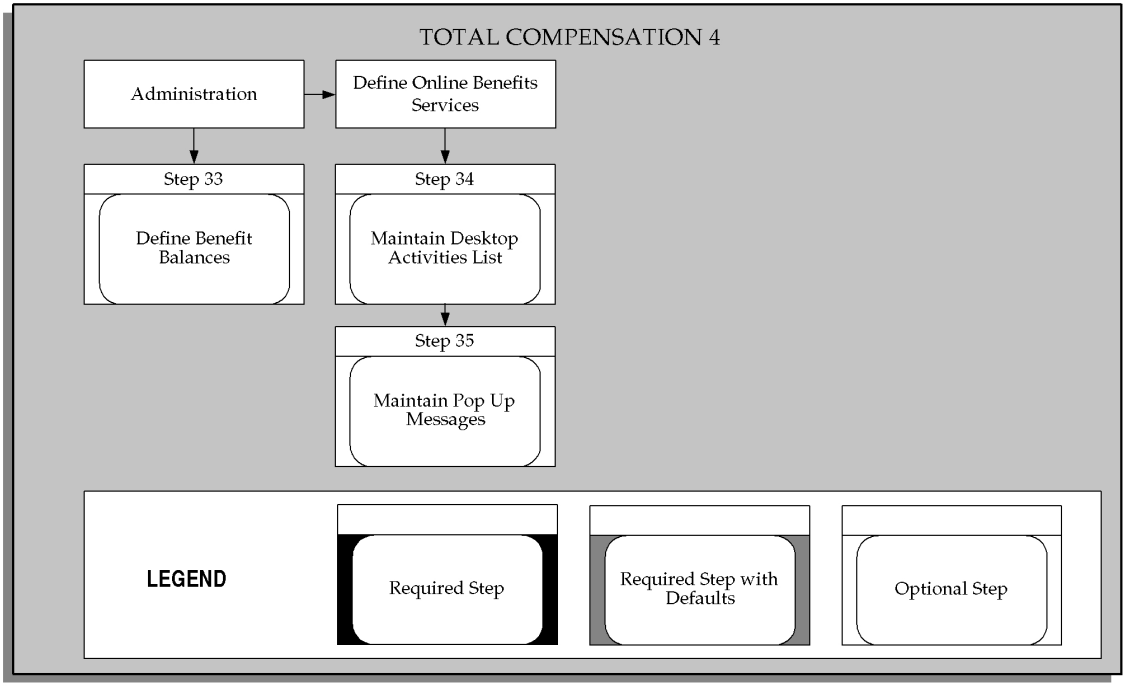


Figure 1 – 10 Implementation Flowchart for People and Assignments

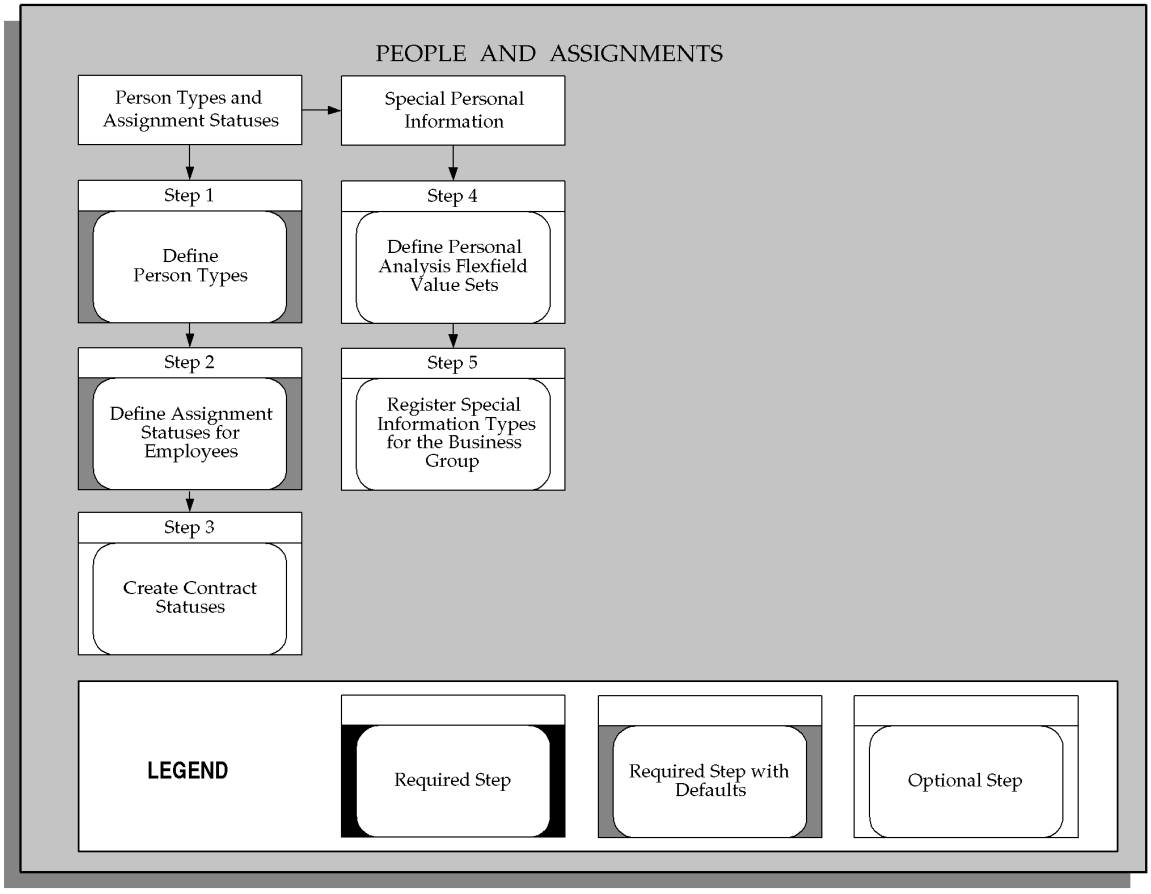


Figure 1 – 11 Implementation Flowchart for Specific Business Functions

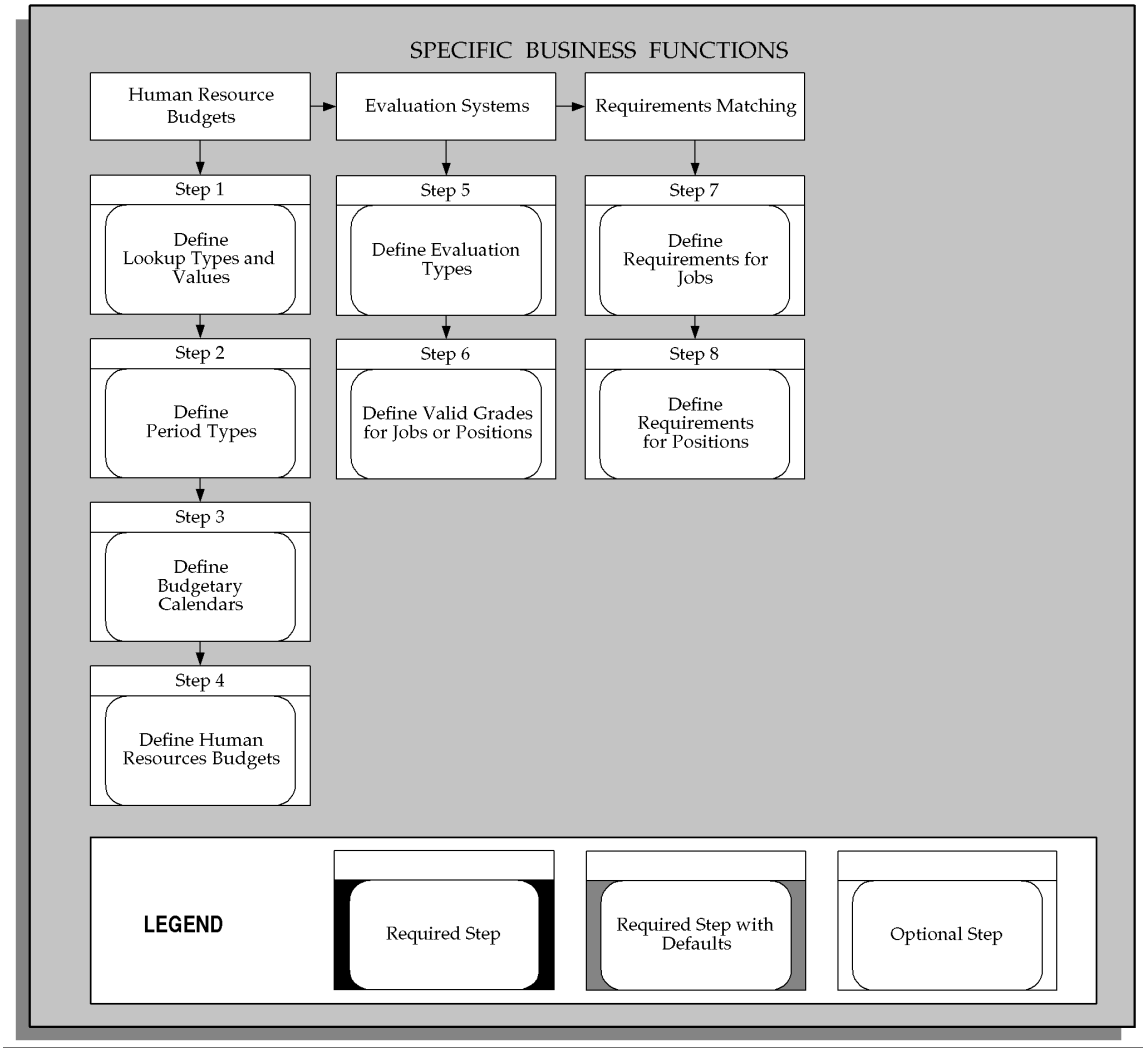


Figure 1 – 12 Implementation Flowchart for Career and Succession Management

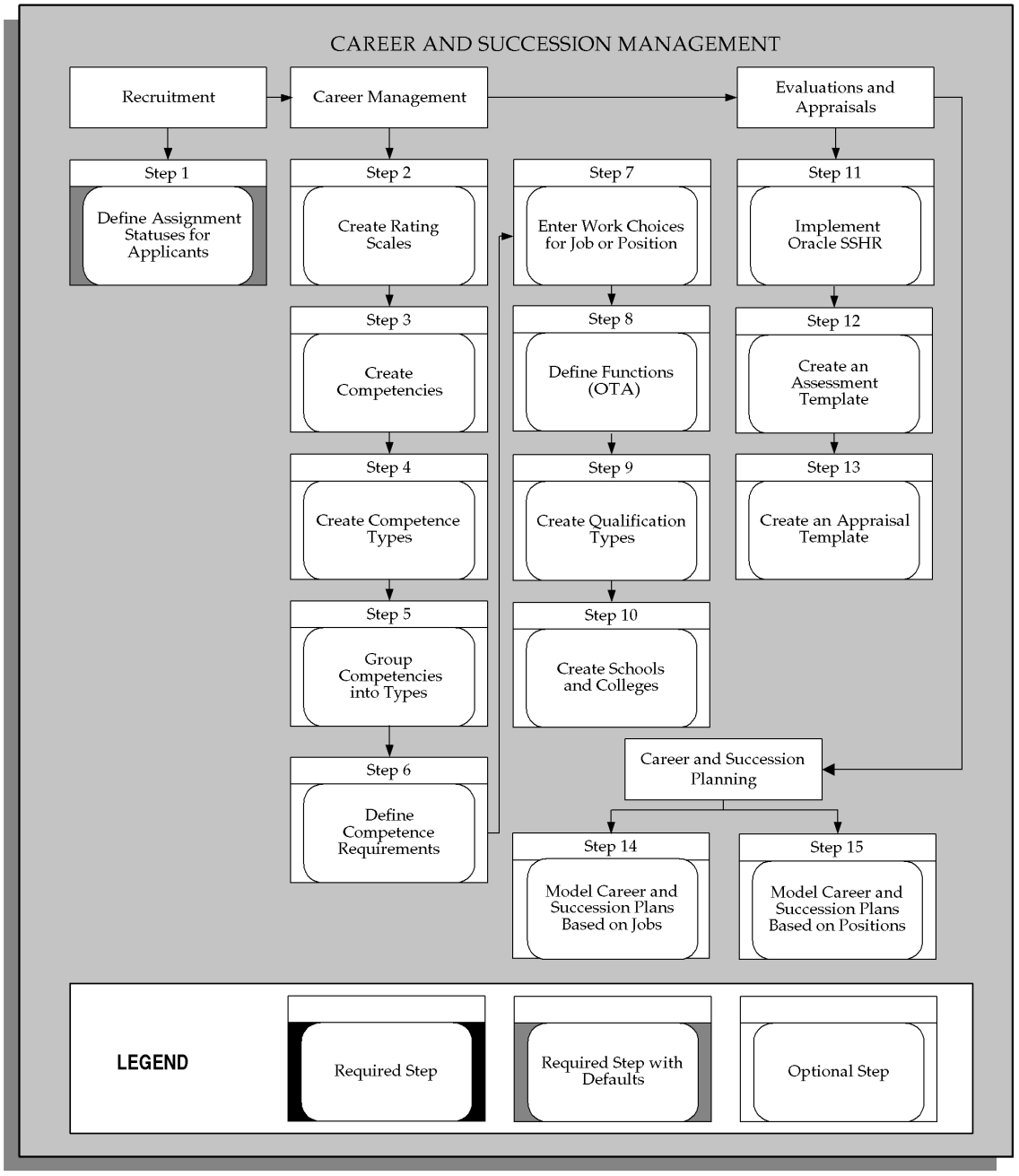


Figure 1 – 13 Implementation Flowchart for Control 1

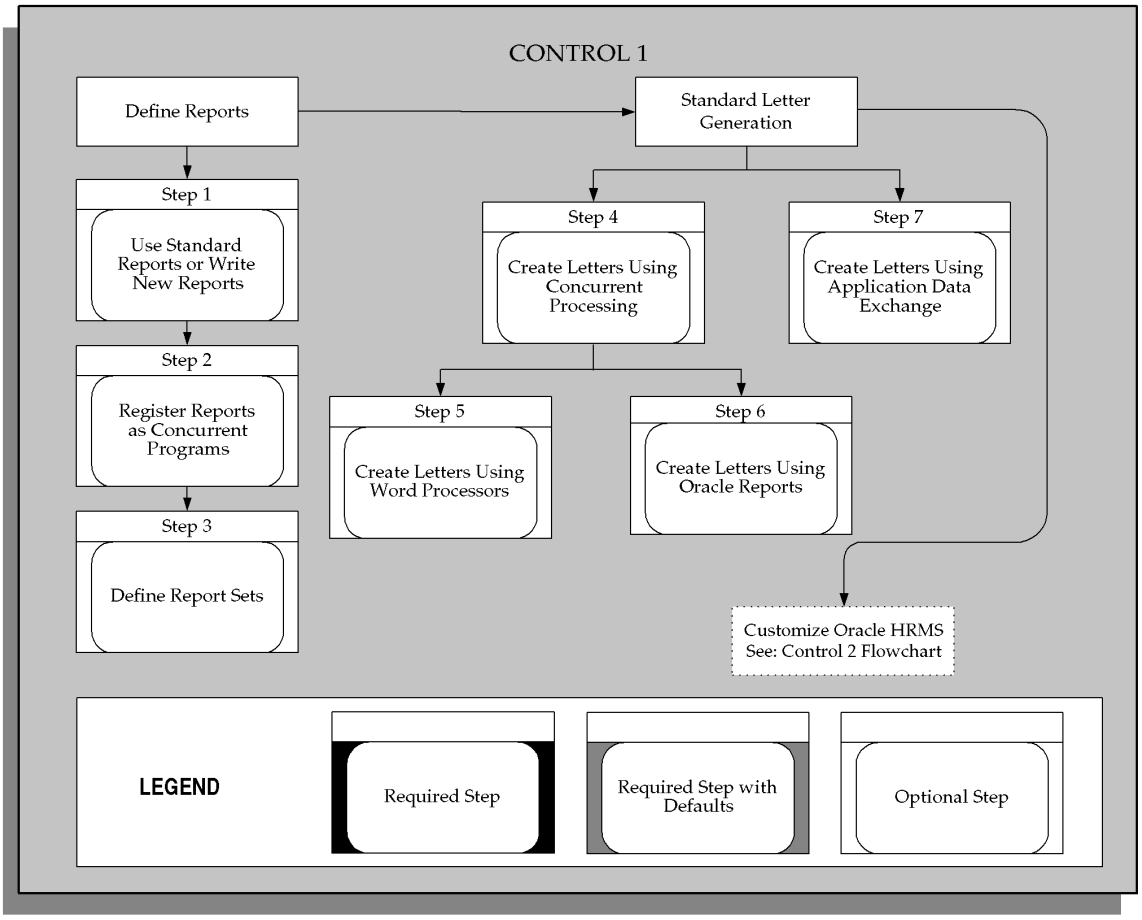


Figure 1 – 14 Implementation Flowchart for Control 2

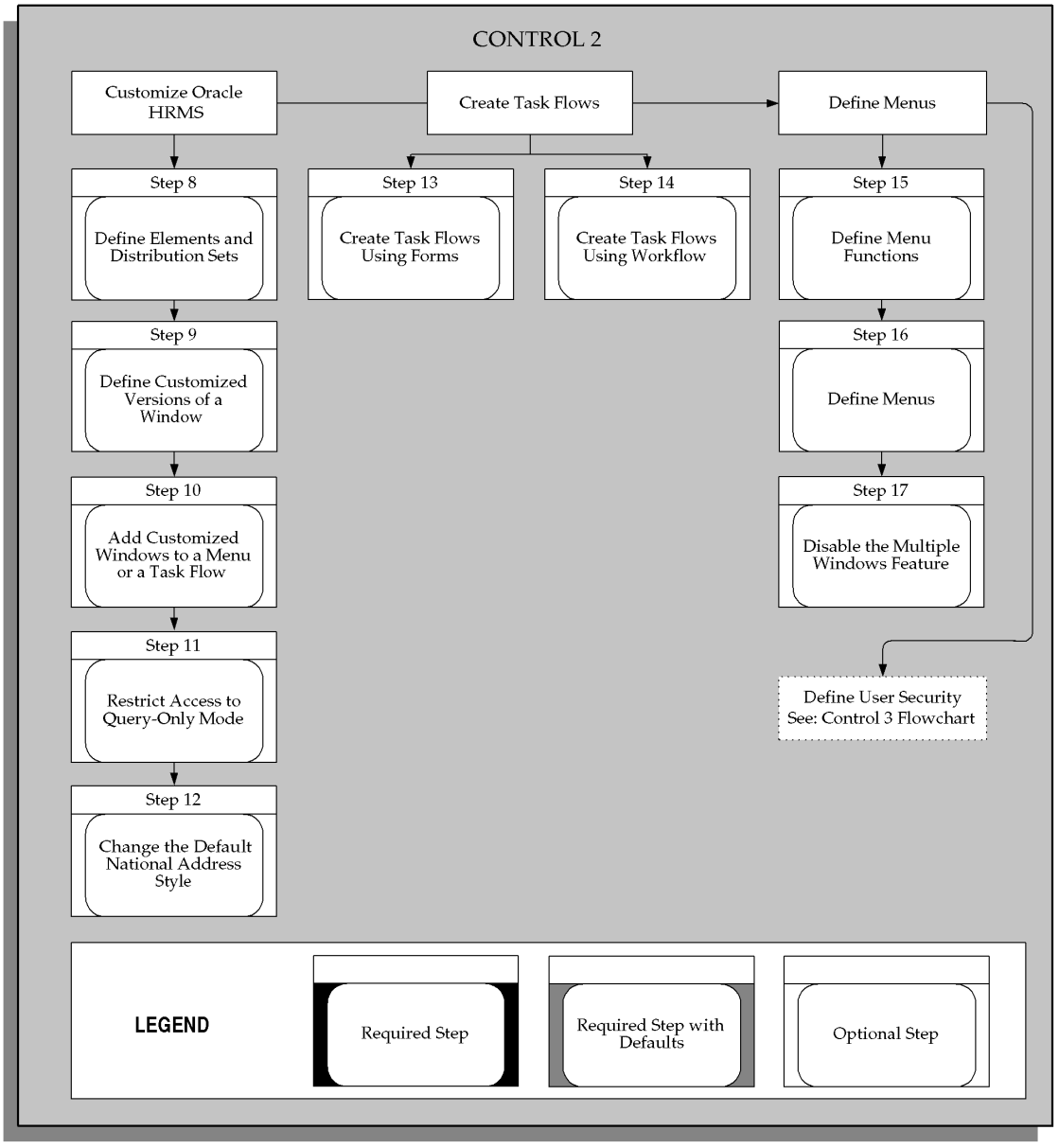
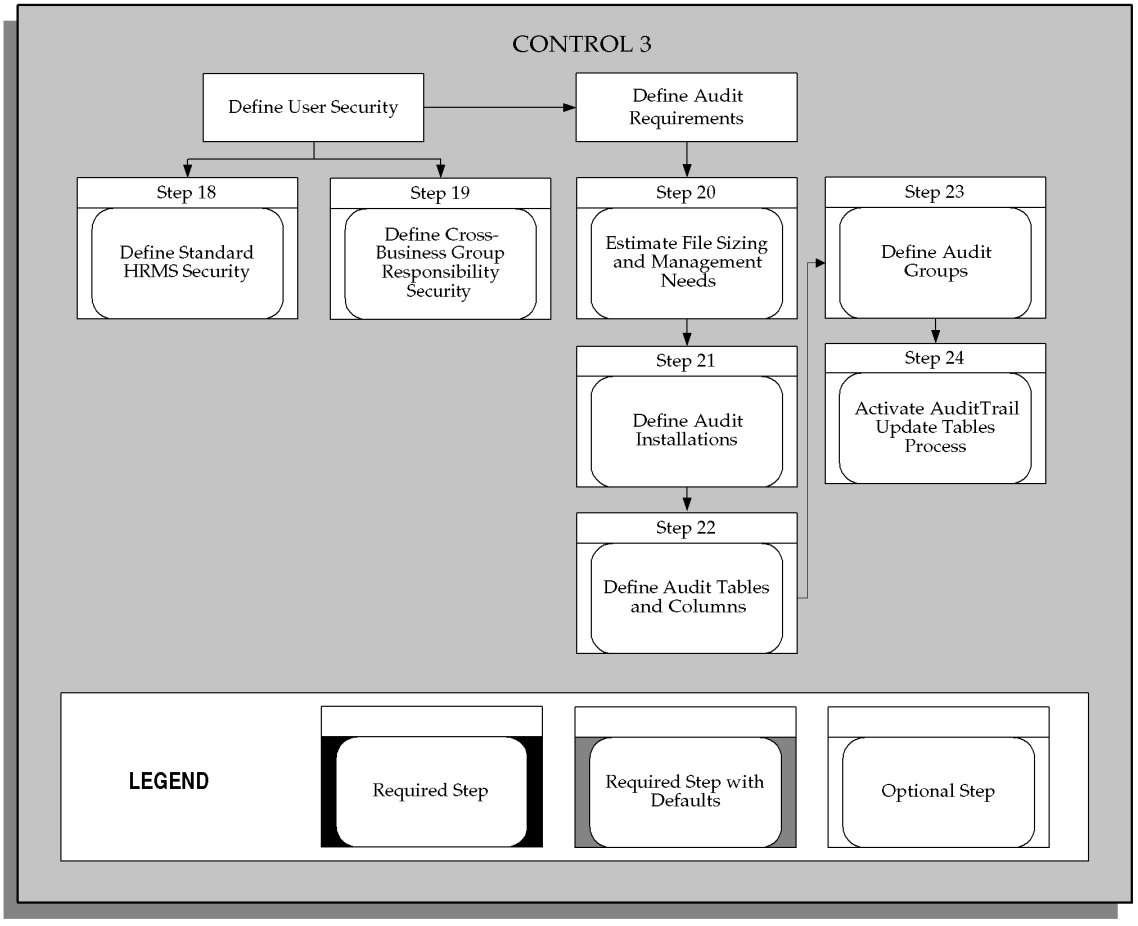


Figure 1 – 15 Implementation Flowchart for Control 3



CHAPTER

2 Implementation Steps

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 5 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

Before you begin your implementation of these 5 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 and People Group Key Flexfields only.

Define Job Flexfield

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

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.

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 Flexfields Guide.*

Step 2 Define Job Flexfield Segments

Define a structure for your Job Flexfield which contains the segments you want to use for your Business Group. You will use this structure to create your unique Job Names in the Job window.

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

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 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 Flexfields Guide.*

Step 9 Define Position Flexfield Segments

Define a structure for your Position Flexfield which contains the segments you want to use for your Business Group. You will use this structure to create your unique Position Names in the Position window.

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: 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. Oracle Human Resource Management Systems 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

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

Step 15 Define Grade 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 Flexfields Guide.*

Step 16 Define Grade Flexfield Segments

Define a structure for your Grade Flexfield which contains the segments you want to use for your Business Group. You will use this structure to create your unique Grade Names in the Grades window.

You must enter Yes in the Allow Dynamic Inserts field. If you enter No, you will not be able to create new grade name combinations in the Grades 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 17 Define Grade Flexfield Segment Values

If you have chosen Independent or Dependent validation for a Value Set used by a Grade 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 18 Define Grade 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 19 Define Grade 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 20 Freeze and Compile Your Grade Flexfield Structure

You are now ready to freeze your Grade Flexfield definition. Navigate to the 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 Grade Flexfield definition. Compiling the flexfield definition enables the Grade 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 21 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 assignments and is used to identify special groups of employees in your enterprise, such as members of a union.

Warning: In Oracle HRMS 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 or applicants.

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

Step 22 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 Flexfields Guide.*

Step 23 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.*

Step 24 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 25 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 26 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 27 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. Oracle Human Resource Management Systems 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 28 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: Create Key Flexfield Database Items, *Customizing, Reporting and System Administration in Oracle HRMS*

Define Cost Allocation Flexfield

Cost Allocation information is normally used to record 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 to take best advantage of the flexibility for recording and reporting costing information in your enterprise, the implementation sequence which you follow is:

Warning: In Oracle HRMS 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 29 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 Flexfields Guide.*

Step 30 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 only key flexfield in Oracle HRMS which makes use of Qualifiers is the Cost Allocation Flexfield. 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. There are 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 31 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 Segments Values, *Oracle Applications Flexfields Guide*

Step 32 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 33 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 34 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. Oracle HRMS 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 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.

Warning: 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.

Suggestion: 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.

Suggestion: 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.

Step 35 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 system.

It is possible that Oracle HRMS will use other segments of these flexfields in the future. Therefore, do not add segments to any protected flexfield. This can affect your ability to upgrade your system in the future.

Use the Descriptive Flexfields window

Step 36 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 Flexfields Guide.*

Step 37 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: If you define a segment as 'Required', it will be required for every record on the system. There are two common problems you can encounter:

- 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, Applicant or Contact.

Use the Descriptive Flexfield Segments window.

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

Step 38 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 Segments Values, *Oracle Applications Flexfields Guide*

Step 39 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*

Define 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 Information Types (Excluding Organizations)

Step 40 Define Extra Information Types for Locations, Jobs, Positions, People and Assignments

Once you have decided which extra information types you require, you need to select the 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.

Use the Descriptive Flexfield Segments window.

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

Step 41 Set Up Responsibility Access for Extra Information Types

EITs will 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 against a Responsibility, *Customizing, Reporting and System Administration in Oracle HRMS*

Define Extra Information 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 42 Define Organization Classification

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

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 43 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, *Customizing, Reporting and System Administration in Oracle HRMS*

Administration

These are tasks for your System Administrator.

Step 44 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.

> **Note:** Extended precision is not used in Oracle HRMS. You can control the precision in any calculation using a formula.

Use the Currencies window

See: Enabling Currencies, *Customizing, Reporting and System Administration in Oracle HRMS*.

Step 45 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*

Application Data Exchange (ADE) and Hierarchy Diagrammers

Step 46 Set Up ADE

You can set up Application Data Exchange (ADE) to export information between your Oracle HRMS database to other applications.

See: Outline of Setup Steps, *Using Application Data Exchange and Hierarchy Diagrammers.*

Step 47 Control Access to Hierarchy Diagrammers

You can also graphically create and maintain your Organization and Position hierarchies, known as Hierarchy Diagrammers. Organization and Position hierarchies reflect reporting lines in your enterprise. The Hierarchy Diagrammers are launched within Oracle HRMS from Application Data Exchange (ADE). You must have ADE installed to use them.

See: Setting Up the Hierarchy Diagrammers, *Using Application Data Exchange and Hierarchy Diagrammers*

Define Organization Structures

Step 48 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, grades, jobs, positions, payrolls, employees, 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.

Note: When you create a business group, the exchange rate type default is Corporate. However, you can define a different exchange rate type for BIS, HRMS Reporting, or Payroll processes using the Table Values window.

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, *Using Oracle HRMS – The Fundamentals.*

Step 49 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, *Using Oracle HRMS – The Fundamentals.*

Step 50 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 set also set up other User Profile Options.

Use the System Profile Values window.

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

Step 51 Define Lookup Types and Values

Lookups supply many of the lists of values in Oracle HRMS. For example, both Title and Nationality in the Person window use Lookups.

Some Lookup Types have been predefined. You only need to define values for these types.

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. 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, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 52 Create Locations

Create each work location used by your enterprise. You define each location and address once only. This saves you time if you have several organizations with the same address.

Use the Location window.

See: Setting Up Locations, *Using Oracle HRMS – The Fundamentals*.

Step 53 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 tax and government reporting purposes, for third party payments.

Suggestion: 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 to an organization before the start date of the organization.

Suggestion: Consider using a fixed date as a default for your initial setup, for example, 01–JAN–1951. This will simplify your data–entry.

Step 54 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, and Entering Additional Information, *Using Oracle HRMS – The Fundamentals*

Accounting Reference Information for Cash Management Integration

If you are using Oracle Payroll with Oracle Cash Management for the reconciliation of payments, then you will also need to set up accounting reference information.

Choose the Operating Unit classification for your organization and then choose the GRE/Legal Entity classification for the organization. Enter the Set of Books and VAT Registration Number in the extra information for Legal Entity Accounting.

Use the Organization window.

See: Creating an Organization, *Using Oracle HRMS – The Fundamentals.*

Step 55 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.

Suggestion: 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.

You can create organization hierarchies using the:

- Organization Hierarchy Window

  See: Creating Organization Hierarchies, *Using Oracle HRMS – The Fundamentals.*

- 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, *Using Application Data Exchange and Hierarchy Diagrammers.*

Define Roles

Step 56 Define Jobs

Jobs can be generic or specific roles within your enterprise. By definition they are independent of organization structures and are generally used where there is flexibility in employee roles.

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.

As you define jobs add any additional information that is appropriate.

Use the Job window.

See: Defining a Job, *Using Oracle HRMS – The Fundamentals.*

See: Entering Additional Information about Jobs and Positions, *Using Oracle HRMS – The Fundamentals*

Step 57 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, *Using Oracle HRMS – The Fundamentals.*

Step 58 Define Positions

In Oracle HRMS a position is a job within an organization. Positions are generally used where roles are fixed within a single organization. If you decide to use positions you may want to use jobs to identify the common job groups of individual positions.

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.

As you define positions add any additional information that is appropriate.

Use the Position window.

See: Defining a Position, *Using Oracle HRMS – The Fundamentals.*

See: Entering Additional Information about Jobs and Positions, *Using Oracle HRMS – The Fundamentals.*

Step 59 Set up the Synchronise Positions Process to Run Nightly

Oracle HRMS uses the Synchronise Positions process to update the non–datetracked Positions table (PER_ALL_POSITIONS_F) 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. Hence, future dated changes are not applied until they become effective.

Running the Synchronise Positions process every night ensures that the system 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's Guide.*

Step 60 Create a Position Hierarchy

You can structure positions into hierarchies to show detailed position reporting structures. You can also use position hierarchies to define security profile groups within your enterprise, or to define career progression paths for positions.

You can create position hierarchies using the:

- Position Hierarchy Window

  See: Creating a Position Hierarchy, *Using Oracle HRMS – The Fundamentals.*

- 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, *Using Application Data Exchange and Hierarchy Diagrammers.*

Define Grade Related Information

Step 61 Define Grades

Grades show the relative status of employees within an enterprise and are often used as the basis for eligibility to Compensation and Benefits.

The Grade Name is a unique combination of values in the segments of the job flexfield structure that you have linked to your Business Group.

You can define Valid Grades for jobs or positions which will be used to cross check the details a user enters as part of the *Employee Assignment*.

Use the Grades window.

See: Defining a Grade, *Using Oracle HRMS – The Fundamentals.*

Step 62 Define Grade Rates

Grade rates are normally used to show valid rates of pay which are directly related to grades. These can be expressed as a fixed value, or as a range of values.

When you define a grade rate you are setting up a table of values. You can use these values with an employee's grade to control, or compare, the salary of the employee.

- You can use grade rate values in a formula to validate the input value of any element for an employee.
- Grade rate values are used to calculate comparatio values in the View Employee Grade Comparatio window and in the Salary Administration window for salary validation.

Use the Grade Rate window.

See: Defining a Grade Rate, *Using Oracle HRMS – The Fundamentals.*

Step 63 Define Pay Scales

Pay scales are used commonly in government and regulated or unionized enterprises where actual values of pay are defined as a 'pay scale', a 'schedule', or a 'spine'.

In this environment it is common to find an automatic incrementing of employee pay based on length of service or on a fixed date. When you define the Pay Scale you define the points in the incrementing sequence you want to use.

A predefined incrementing process is supplied with Oracle HRMS. This will automatically increment step and point values for employees using a fixed date.

> **Note:** You can modify the process to meet your specific business rules for incrementing.

Use the Pay Scale window.

See: Defining a Pay Scale, *Using Oracle HRMS – The Fundamentals.*

Step 64 Define Scale Rates

You define a scale rate for each point on the pay scale. These values are DateTracked.

Use the Scale Rate window.

See: Defining Scale Rates, *Using Oracle HRMS – The Fundamentals.*

Step 65 Relate Grades to Progression Points

Define the valid points for each grade as a numeric sequence of steps.

The steps you define are used in the auto–incrementing process which will increment an employee's grade point up to a ceiling which you can define for the grade. Points above the ceiling can be entered by users in the *Grade Step Placement* window.

Use the Grade Scale window.

See: Relating Grades to Progression Points, *Using Oracle HRMS – The Fundamentals.*

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 66 Define Payment Methods

Standard categories of payment methods such as Cheque/Check and Direct Deposit are predefined with your system. You can define your own names for each of these methods, and if you have installed Oracle Payroll you can also use these methods to control payments to your employees.

Use the Organizational Payment Method window.

See: Defining a Payment Method, *Using Oracle HRMS – The Fundamentals.*

Step 67 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.

See: Defining Consolidation Sets, *Running Your Payroll Using Oracle HRMS.*

Step 68 Define Payrolls

You define your own payroll groups to meet your business needs for processing and payment. For example, you may have a semi–monthly and a weekly payroll but you might want to manage and process your weekly payroll by plant location. In this case you could define one semi–monthly payroll and two weekly payrolls, one for each plant.

> **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.

See: Defining a Payroll, *Using Oracle HRMS – The Fundamentals.*

Compensation and Benefits

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

Attention: If you intend to set up benefit plans using the standard or advanced benefits functionality, follow the implementation steps in the **Total Compensation** section to set up your benefit plans. See: Total Compensation: page 2 – 38.

Before you define any 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–1951, as a default for your initial setup definitions. This will simplify your data–entry.

Define Input Value Validation

Step 69 Define Lookup Types and Values

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, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 70 Define User Tables

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 *Customizing, Reporting and System Administration in Oracle HRMS*.

Step 71 Define Table Values

You now need to define the table values.

Use the Table Values window.

See: Entering Table Values, *Customizing, Reporting and System Administration in Oracle HRMS*.

Step 72 Define Element Validation Formulas

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

Attention: You must define the formula before you define the element input value.

The type of formula is *Element Input Validation* with the following constraints:

- The formula has one Input only:

  ENTRY_VALUE(char)

- The formula must return a predefined status code for success or error:

  ```
  FORMULA_STATUS =  'S'  or   'E'
  ```

- You can also return a message for the user, which is displayed in a message window:

  ```
  FORMULA_MESSAGE = ' ...  '
  ```

Define Compensation and Benefits

Step 73 Define Elements and Input Values

Elements are the basic components of all compensation and benefit types. You can also use elements to represent tangible items distributed to employees, such as tools or safety equipment.

For each element you can:

- Define up to 15 input values
- Set validation options for each value

- Fixed
- Range
- List of values using Lookups
- Formula

- Set *Hot* and *Cold* Defaulting Rules

If you are using the element for payroll processing, you can also:

- Make one input value the 'Pay Value' for the element

  Note: If you set the *Process In Run* flag to 'Yes' a pay value will be created automatically.

  You must set this flag to 'Yes' if you want to process this type of element in a payroll run.

- use the Balance Feed window to modify the individual balances that an element will feed.

Use the Element window.

See: Defining an Element, *Managing Compensation and Benefits Using Oracle HRMS*

Step 74 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, employment category or salary basis.

Use the Element Link window.

See: Defining Element Links, *Managing Compensation and Benefits Using Oracle HRMS*

Step 75 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.

If you have installed Oracle Payroll you will also have all of the formulas and balances you need for processing these deductions. If you have not installed Oracle Payroll, you can still use these elements to record information for transfer to your own payroll system.

To activate these predefined elements you need only define links for them.

Use the Element Link window.

See: Defining Element Links, *Managing Compensation and Benefits Using Oracle HRMS*

UK Implementations Only

Examples of elements representing legislative deductions in the UK are: PAYE, NI, PAYE, Court Orders, EAS Court Order (only applies in Scotland) and CMA Court Order (only applies in Scotland).

If you are a UK implementation and you want to use the employee Tax window to enter PAYE and NI details you should define a 'Standard' link to all Payrolls for both the PAYE and NI elements.

When you define links for the PAYE and NI elements you will have to specify some default input values. We suggest you use the following defaults:

- **Tax Code** 'BR'
- **Tax Basis** 'Cumulative'
- **NI Category** 'A' or 'D' (Use category D if the majority of your employees are enrolled in a company pension scheme.)

Define Balances, Formulas and Results for Payroll

If you have not installed Oracle Payroll these windows will not be available in your menus. You should skip these steps and go to the next section.

Step 76 Define User Balances

Attention: Oracle Payroll has many predefined balances installed with the system to support all your legislative requirements for calculation of gross to net balances. To protect the integrity of the payroll processes you cannot change any of these balances.

You can define other balances. For example, you might want to define a special balance to calculate a 'Stop Rule' on a recurring deduction. You might also need to define a special balance for calculating retroactive payments.

When you define a payroll balance you must specify the feeds and the dimensions.

Use the Balance window.

See: Defining User Balances, *Managing Compensation and Benefits Using Oracle HRMS.*

Step 77 Write Payroll Formulas

You write the formula for every element that you want to process in a payroll run. The formula type is 'Oracle Payroll'

Warning: Remember that formula definitions are datetracked. After you have used a formula in a payroll calculation you should always 'Update' any changes to the formula.

This will keep the history of formulas for any re–calculation of retrospective earnings or deductions.

Use the Formula window.

See: Writing Payroll Formulas for Elements, *Using Oracle FastFormula*.

Step 78 Define Formula Result Rules

When you process an element in a payroll run the system will calculate the results using a formula. The results of the formula are the values you include in the *Return* statement to end the formula. The result rules define what will happen to each of the results produced by the formula.

You can calculate any number of different results in a single formula. The different types of result are:

- Direct
- Indirect
- Message
- Stop Recurring
- Update Recurring

There can be only one *Direct* result of a payroll calculation. This would normally be the Pay Value of the entry.

Warning: If you allow users to enter the Pay Value of any earnings or deduction type, this value will override any formula calculation to provide the direct result for payment.

Use the Formula Result Rules window.

See: Defining Formula Processing and Result Rules, *Managing Compensation and Benefits Using Oracle HRMS*

Salary Administration

Use the Salary Administration function to manage the basic remuneration for individual employees.

Step 79 Create or Decide on Salary Elements

You need at least one salary element for each salary basis in your enterprise.

If predefined elements exist in your localization, you might decide to use these. If your localization does not include predefined elements, or if the predefined elements are insufficient or inappropriate, you must create new elements to store the salary value.

> **Note:** Consider how many different elements you will need to define for the different salary bases you want to manage. Remember that you can administer the salary on an annual basis but store the value as a monthly value.

Use the Element window.

See: Creating a Salary Element, *Managing Compensation and Benefits Using Oracle HRMS.*

Step 80 Link the Salary Element

Link the salary elements to components of employee assignments to establish employee eligibility for the elements.

Use the Element Link window.

See: Linking the Salary Element, *Managing Compensation and Benefits Using Oracle HRMS.*

Step 81 Define a Salary Basis

Define a *salary basis* for each salary element to be used for salary administration. This establishes the duration for which a salary is quoted, for example, hourly, monthly or annually.

Use the Salary Basis window.

See: Defining a Salary Basis, *Managing Compensation and Benefits Using Oracle HRMS.*

Step 82 Review or Create Salary Components

Review the salary components predefined as values for the Lookup Type PROPOSAL_REASON. If necessary, create your own salary components.

If you want your new components to be displayed in the Salary Management folder, you must also change a view.

See: Creating Salary Components, *Managing Compensation and Benefits Using Oracle HRMS*

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 83 Define Performance Rating Types

If you want to record performance ratings such as Outstanding, Superior and Average, enter them in the Application Utilities Lookups window for the Lookup Type PERFORMANCE_RATING.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 84 Add the Salary Administration Approve Function

Add the function "Salary Administration Approve" to the menu of responsibilities that should be able to approve salary proposals.

> **Note:** If this function does not exist for a Responsibility then a user can enter but not approve salary proposals.

Use the Menus window.

See: Defining Menus, *Customizing, Reporting and System Administration in Oracle HRMS*.

Step 85 Validate Salary Entries

You can validate salary entries in one of two ways:

- *Warn* users when they enter a salary proposal that is outside a valid range defined for an employee's grade. This approach uses grade rate ranges.
- *Prevent* users from approving a salary that is outside a valid range, or that fails validation performed by a formula. Notice that this validation is not performed until you try to approve a salary proposal. This approach uses element input value validation.

See: Validating Salary Entries, *Managing Compensation and Benefits Using Oracle HRMS.*

Absence Management and Accruals of Paid Time Off (PTO)

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.

Set Up Absence Management

Step 86 Define a Nonrecurring Absence Element

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

Use the Element window.

See: Defining and Linking an Absence Element, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 87 Link the Absence Element

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

Use the Element Link window.

See: Defining Element Links, *Managing Compensation and Benefits Using Oracle HRMS*

Step 88 Define Categories of Absence Types

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, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 89 Define Absence Types and Associate with Absence Elements

Define each absence type, and associate it with an absence element

Use the Absence Attendance Type window.

See: Defining an Absence Type, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 90 Make Initial Element Entries

For an absence type with a decreasing balance, use the Element Entries window or the MIX batch facility to make initial element entries for employees eligible for the type.

If you want to make entries for individual employees, see Making Manual Element Entries, *Managing Compensation and Benefits Using Oracle HRMS*. If you want to make batch entries, see Making Batch Element Entries Using BEE, *Managing Compensation and Benefits Using Oracle HRMS*

Set Up Accrual Plans

Step 91 Define and Link Element for Plan's Absence Type

Define and link an absence element, if you haven't already done this.

Use the Element window.

See: Defining and Linking an Absence Element, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 92 Define an Absence Type for the Plan

If you expect to record accrued time taken under the plan using the Absence Detail window, define an absence type for the plan, associating its absence element with this type.

Use the Absence Attendance Type window.

See: Defining an Absence Type, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 93 Define New Accrual Start Rules

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.

Step 94 Decide on Accrual and Carry Over Formulas

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

Use the Formula window.

See: Writing Formulas for Accrual Plans, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 95 Write Ineligibility Formula

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, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 96 Define New Accrual Categories

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, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 97 Define a PTO Accrual Plan

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, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 98 Set Up Length of Service Bands

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

Use the Accrual Bands window.

See: Setting Up Length of Service Bands, *Managing Compensation and Benefits Using Oracle HRMS*.

Step 99 Review the Net Calculation Rules

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, *Managing Compensation and Benefits Using Oracle HRMS*.

Element Sets

Step 100 Define Element Sets

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, *Customizing, Reporting and System Administration in Oracle HRMS*.

Total Compensation

Many implementation steps are shared by Standard and Advanced Benefits. Those implementation steps that only apply to Advanced Benefits are indicated.

Benefits Tabbed Region

You need to add the benefits tabbed region to the People window for those secure responsibilities that should have access to the benefits information contained in this region. This region includes such information as the benefits group to which a person belongs and their medical plan.

Step 101 Add the Benefits Tabbed Region to the People Window

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, *Oracle Applications System Administrator's Guide*

Benefits Eligibility

You define participation eligibility profiles to determine eligibility for benefits. Eligibility factors can also be used when determining variable contribution and distribution rates for a benefit.

Step 102 Define Benefits Groups

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 103 Define Postal Code (ZIP) Ranges

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 104 Define Service Areas

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 105 Define Regulations

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, *Managing Compensation and Benefits Using Oracle HRMS*

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 106 Define Derived Compensation Level Factors

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 107 Define Derived Percent of Full Time Employment Factors

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 108 Define Derived Hours Worked in Period Factors

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 109 Define Age Factors

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

Use the Derived Factors window.

See: Defining Derived Factors: Age, *Managing Compensation and Benefits Using Oracle HRMS*

Step 110 Define Length of Service Factors

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 111 Define Combination Age and Length of Service Factors

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, *Managing Compensation and Benefits Using Oracle HRMS*

Eligibility Profiles

Step 112 Define an Eligibility Profile

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 are 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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 113 Define Dependent Coverage Eligibility Profiles

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

Use the Dependent Coverage Eligibility Profiles window.

See: Defining the Criteria in a Dependent Coverage Eligibility Profile, *Managing Compensation and Benefits Using Oracle HRMS*

Define Life Events (Advanced Benefits)

You define a life event as a change in a person's record that can potentially trigger an enrollment action. Life events can be external to work, such as a marriage or the birth of a dependent, or they can be internal, such as a job change. Scheduled enrollments are also considered life events.

Step 114 Define Life Event Processing

Define the life events that you use to control electability, activity rates and coverage levels, coverage dates, communications, and automatic and default enrollment processing.

Use the Life Event Reasons window.

See: General Characteristics of Life Event Reasons, *Managing Compensation and Benefits Using Oracle HRMS*

Step 115 Define Person Changes

You define the changes to a person's record that trigger a life event by specifying the value of the database field that indicates this person change has occurred.

Use the Person Changes window.

See: Defining Person Changes, *Managing Compensation and Benefits Using Oracle HRMS*

Step 116 Associate Person Changes with Life Events

You associate the *person change* that triggers the life event for each life event that you define.

Use the Person Change Causes Life Event window.

See: Associating a Person Change with a Life Event, *Managing Compensation and Benefits Using Oracle HRMS*

Step 117 Define Related Person Changes

You define the changes to a person's record that trigger a life event for a related person by specifying the value of the database field that indicates this related person change has occurred.

For example, you could define a termination life event to end benefits coverage for terminated employees. You would define a corresponding related person life event that ends coverage for the dependents of the primary participant when the primary participant is terminated.

Use the Related Person Changes window.

See: Defining Person Changes, *Managing Compensation and Benefits Using Oracle HRMS*

Step 118 Associate Related Person Changes with Life Events

You associate a *related person change* with each related person life event that you define. A related person change is a change to the primary participant's HR record that may generate a life event for a person related to the primary participant.

Use the Related Person Change Causes Life Event window.

See: Associating a Person Change with a Life Event, *Managing Compensation and Benefits Using Oracle HRMS*

Program Setup

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 119 Define Reimbursable Goods and Service Types

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 120 Define a Program or Plan Year Period

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 121 Define Plan Types

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 122 Define Options

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 123 Define Plans

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 General Plan Information, *Managing Compensation and Benefits Using Oracle HRMS*

Step 124 Define Reimbursement Plans

Reimbursement plans allow you 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 General Plan Reimbursement Information, *Managing Compensation and Benefits Using Oracle HRMS*

Step 125 Define Programs

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 General Characteristics of a Program, *Managing Compensation and Benefits Using Oracle HRMS*

Enrollment Requirements

You define enrollment requirements to control when an eligible person can enroll in a benefit.

Step 126 Define Program Enrollment Requirements

Enrollment requirements determine how an eligible participant enrolls in a program.

Standard benefits customers define enrollment requirements based on the unrestricted enrollment type. Advanced Benefits benefits customers can specify whether default or automatic enrollment rules apply for a program.

Use the Program Enrollment Requirements window.

See: Defining Enrollment Methods for a Program, *Managing Compensation and Benefits Using Oracle HRMS*

Step 127 Define Enrollment Requirements for a Plan

You use the Plan Enrollment Requirements window to define enrollment requirements for a not in program plan or an option in plan.

You also use this window to set up requirements for beneficiary designations.

Use the Plan Enrollment Requirements window.

See: Defining an Enrollment Method for a Plan, *Managing Compensation and Benefits Using Oracle HRMS*

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 128 Calculate Variable Activity Rates

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 129 Calculate Coverages

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 130 Define Across Plan Type Coverage Limits

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

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

Step 131 Calculate Actual Premium Costs

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 Imputed Income Calculation, *Managing Compensation and Benefits Using Oracle HRMS*

Step 132 Define Period–to–Date Limits

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, *Managing Compensation and Benefits Using Oracle HRMS*

Step 133 Define Activity Rates for Standard Contribution/Distribution

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 Standard Contribution/Distribution, *Managing Compensation and Benefits Using Oracle HRMS*

Reporting Groups

Step 134 Define Reporting Groups

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, *Managing Compensation and Benefits Using Oracle HRMS*

Flex Credit Calculations (Advanced Benefits)

Step 135 Define Characteristics of Benefit Pools

You define benefit pools to limit how a participant can spend flex credits and how excess flex credits can be rolled over, distributed as cash, or forfeited.

Use the Benefit Pools window.

See: Defining the General Characteristics of a Benefit Pool, *Managing Compensation and Benefits Using Oracle HRMS*

Step 136 Define Flex Credits

You define a flex credit calculation and link the calculation with a compensation object. The compensation object to which you link a flex credit calculation must be part of a program regardless of the level at which you define flex credits.

Use the Flex Credit Definitions window.

See: Defining Flex Credits, *Managing Compensation and Benefits Using Oracle HRMS*

Step 137 Define Communications

You define the communications you send to employees and other potential participants. You specify the conditions that trigger a communication and the delivery method and medium.

Use the Communication Types window.

See: Defining Communications, *Managing Compensation and Benefits Using Oracle HRMS*

Administration

Step 138 Define Benefit Balances

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 Type, *Managing Compensation and Benefits Using Oracle HRMS*

Define Online Benefits Services

You use the Online Benefit Services window to access a variety of benefits windows from a central location. You can configure the

windows that are accessible from this window and you can define the pop up messages that display based on user events that you define.

Step 139 Maintain Desktop Activities List

The Maintain Online Activities window lets you define the functions and windows that are available from the Desktop Activities list of the Online Benefits Services windows.

Use the Maintain Online Activities window.

See: Maintaining Online Activities, *Managing Compensation and Benefits Using Oracle HRMS*

Step 140 Maintain Pop Up Messages

You can configure messages to display in the Online Benefit Services window based on user events that you define. You create the message text in the Messages window

Use the Maintain Pop Up Messages window.

See: Maintaining Pop Up Messages, *Managing Compensation and Benefits Using Oracle HRMS*

People and Assignments

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 in each assignment using your own names.

Person Types and Assignment Statuses

Step 141 Define Person Types

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, *Managing People Using Oracle HRMS*.

Step 142 Define Assignment Statuses for Employees

With Oracle HRMS you can identify the status of employees in each assignment using your own names. For example, you might want to define a special status to identify assignments which have been *Suspended* while the employee is temporarily assigned to another role.

Use the Assignment Statuses window.

See: Defining Assignment Statuses, *Managing People Using Oracle HRMS*.

Step 143 Create Contract Statuses

You can create up to 250 contract statuses. You can select a contract status in the Contracts window. Create the contract statuses you require using the Lookups CONTRACT_STATUS.

The contract status can contain a prefix that defines whether a contract is active, inactive or obsolete.

- **A–**: You should use this prefix for statuses that indicate a contract is Active.
- **O–**: You should use this prefix for statuses that indicate a contract is Obsolete.

> **Note:** If a contract status has no prefix it is assumed to mean that the contract is Inactive.

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Customizing, Reporting and System Administration in Oracle HRMS*

Special Personal Information (Personal Analysis Key Flexfield Structures)

The Personal Analysis Key Flexfield is used to record special personal information which is not included as standard information. Each type of information is defined 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 144 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 145 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.

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

Step 146 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.

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

Step 147 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 148 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 149 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 150 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
  - *ADA* for use only in the US, for special information types set up to record information about employees with disabilities.
  - *OSHA* for use only in the US, for a special information type set up to record information about employees' work–related injuries or illness.

Suggestion: 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, *Managing People Using Oracle HRMS*.

Specific Business Functions

Human Resource Budgets

Step 151 Define Lookup Types and Values

Headcount and Full–Time Equivalent budget measurement types are already predefined in Oracle HRMS. You can change the names of these predefined types or add any new types you might need.

Define values for BUDGET_MEASUREMENT_TYPES

Use the Application Utilities Lookups window.

See: Adding Lookup Types and Values, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 152 Define Period Types

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

Use the Period Types window.

See: Renaming Period Types, *Managing People Using Oracle HRMS.*

Step 153 Define Budgetary Calendars

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

Use the Budgetary Calendar window.

See: Defining Budgetary Calendars, *Managing People Using Oracle HRMS.*

Step 154 Define Human Resources Budgets

When you define staffing budgets you can use the system to measure actual budget values of assignments against planned budget values.

An assignment which does not have an actual value is not counted in the budget. Actual values for each budget type for an assignment are entered in the Assignment Budget Values window.

Use the Budget window.

See: Defining Human Resource Budgets, *Managing People Using Oracle HRMS.*

Evaluation Systems

Step 155 Define Evaluation Types

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 156 Define Valid Grades for Jobs or Positions

Oracle HRMS lets you define Valid Grades for Jobs. These definitions provide warning messages to users in the Assignment window when you enter Job and Grade information.

Use the Valid Grades window.

See: Entering Valid Grades for Jobs or Positions, *Using Oracle HRMS – The Fundamentals.*

Requirements Matching

If you have 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. You do this by selecting

categories in the Special Information Type window. Enabling entry of information for people enables you to 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.

Step 157 Define Requirements for Jobs

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, *Using Oracle HRMS – The Fundamentals.*

Step 158 Define Requirements for Positions

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, *Using Oracle HRMS – The Fundamentals.*

Career and Succession Management

Recruitment

Step 159 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.

Use the Assignment Statuses window.

See: Defining Assignment Statuses, *Managing People Using Oracle HRMS*.

Career Management

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 (SSHR), *Implementing Oracle Self–Service Human Resources (SSHR).*

Step 160 Create Rating Scales

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, *Managing People Using Oracle HRMS.*

Step 161 Create Competencies

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.

Use the Competencies window.

See: Creating a Competence, *Managing People Using Oracle HRMS.*

Step 162 Create Competence Types

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 163 Group Competencies into Types

You now need to group related competencies together.

Use the Competence Types window.

See: Grouping Competencies into Types, *Managing People Using Oracle HRMS.*

Step 164 Define Competence Requirements

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

Use the Competence Requirements window.

See: Defining Competence Requirements – Core or Generic Competencies, *Managing People Using Oracle HRMS.*

See: Defining Competence Requirements – No Core Competencies, *Managing People Using Oracle HRMS.*

Step 165 Enter Work Choices for a Job or Position

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, *Managing People Using Oracle HRMS.*

Step 166 Define Functions (to Implement the Competence Approach in OTA)

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

You use parameters to enable you to phase in the delivery of competencies through training activities. This enables you to indicate whether users can enter skills, competencies, or both from the Activities window. You also use parameters to enable selected users to add competencies gained through an activity directly to a student's Competence Profile.

Use the Form Functions window.

See: Defining Functions, *Using Oracle Training Administration.*

Step 167 Create Qualification Types

You can enter all the qualification types your enterprise recognizes.

Use the Qualification Types window.

See: Creating Qualification Types, *Managing People Using Oracle HRMS.*

Step 168 Create Schools and Colleges

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: Schools and Colleges, *Managing People Using Oracle HRMS.*

Evaluations and Appraisals

Step 169 Implement Oracle Self–Service Human Resources (SSHR)

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 (SSHR), *Implementing Oracle Self–Service Human Resources (SSHR).*

Step 170 Create an Assessment Template

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

Use the Assessment Template window.

See: Creating an Assessment Template, *Managing People Using Oracle HRMS.*

Step 171 Create an Appraisal Template

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 an Appraisal Template, *Managing People Using Oracle HRMS.*

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

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.

Attention: 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, *Managing People Using Oracle HRMS.*

Step 172 Create and Map Career Paths

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, *Managing People Using Oracle HRMS.*

Step 173 Enter Work Choices

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, *Managing People Using Oracle HRMS.*

Model Career and Succession Plans Based on Positions

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 174 Create Position Hierarchies

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, *Using Oracle HRMS – The Fundamentals.*

Step 175 Use Succession Planning (SSHR with a Line Manager Responsibility)

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 176 Use Suitability Matching (SSHR with a Line Manager Responsibility)

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

Use the Suitability Matching function in SSHR.

Step 177 Use Attachments or Special Information Types

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

Use the Personal Analysis Key Flexfield.

See: Defining Special Information Types, *Managing People Using Oracle HRMS.*

Control

Define Reports

Step 178 Use Standard Reports or Write New Reports

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 179 Register Reports as Concurrent Programs

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 User's Guide.*

Step 180 Define Report Sets

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 User'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 two different methods to create standard letters:

- Method 1 Concurrent Processing : page 2 – 63
- Method 2 – Online, using Application Data Exchange (ADE). See: *Using Application Data Exchange and Hierarchy Diagrammers*.

Method 1 – Concurrent Processing

There are two methods of using concurrent processing to set up your standard letters:

- Using word processors
  - 1a – MultiMate or WordPerfect: page 2 – 63
  - 1b – Microsoft Word: page 2 – 65
- Using Oracle Reports: page 2 – 66

Using Word Processors Option 1a – MultiMate or WordPerfect

Step 181 Plan Standard Letter Requirements

You need to identify the database information to include in the letters.

See: Planning Standard Letter Requirements, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 182 Write a SQL*Plus Script for MultiMate or WordPerfect

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 183 Register the SQL*Plus Script

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 recognise it as a type of letter.

Use the Concurrent Programs window.

See: Registering the SQL*Plus Script, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 184 Link the SQL*Plus Script to the Letter

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 the Letter, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 185 Writing a Skeleton Letter

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 186 Requesting Letters

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 187 Merging the Data Files

You now need to merge the data in the Data File with your skeleton letters.

See: Merging the Data File with the Standard Letter, *Customizing, Reporting and System Administration in Oracle HRMS.*

Using Word Processors Option 1b – Microsoft Word

Step 188 Plan Standard Letter Requirements

You need to identify the database information to include in the letters.

See: Planning Standard Letter Requirements, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 189 Write a SQL*Plus Script for Microsoft Word

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 Microsoft Word, *Customizing, Reporting and System Administration in Oracle HRMS*

Step 190 Register the SQL*Plus Script

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 191 Link the SQL*Plus Script to the Letter

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 to the Letter, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 192 Writing a Skeleton Letter

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 193 Requesting Letters

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 194 Merging the Data Files

You now need to merge the data in the Data File with your skeleton letters.

See: Merging the Data Files, *Customizing, Reporting and System Administration in Oracle HRMS.*

Option 2 – 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 195 Plan Standard Letter Requirements

You need to identify the database information to include in the letters.

See: Planning Standard Letter Requirements, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 196 Write and Register the Report

You now need to write and register the report.

See: Writing and Registering the Report, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 197 Link the Report with a Letter

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 198 Run the Report

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: Running the Report, *Customizing, Reporting and System Administration in Oracle HRMS.*

Customize Oracle HRMS

Step 199 Define Elements and Distribution Sets

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 200 Define Customized Version of a Window

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 customization option. Remember that the user guides and the online help use the default window names to identify windows.

You can call the customized 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: Defining Customized Version of a Window, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 201 Add Customized Window to a Menu or a Task flow

You must add your customized windows to a menu or task flow.

See: Adding Customized Windows to a Menu or a Task Flow, see *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 202 Restrict Access to Query–Only Mode

You can restrict access to query–only mode for an individual form.

See: Adding Customized Windows to a Menu or a Task Flow, see *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 203 Change the Default National Address Style

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, see *Customizing, Reporting and System Administration in Oracle HRMS.*

Create 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 'customized' 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 – 69
- Workflow: page 2 – 69

Create Task Flows Using Forms

Step 204 Define Task Flow Nodes

All of the taskflow 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 taskflow.

Use the Define Task Flow Nodes window.

See: Defining Task Flow Nodes, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 205 Define Task Flows

Arrange the nodes of your task flows in sequential or branched groups

See: Defining Task Flows, *Customizing, Reporting and System Administration in Oracle HRMS.*

Create Task Flows Using Workflow

Step 206 Create a Top Level Process

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 207 Create Sub Processes

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 208 Create Button Labels

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 209 Position Button Display

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 210 Identify Windows or Blocks to Display

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 211 Identify Customized Forms to Include in the Task Flow

If you have created a customized version of a window, you can use the customized version of the window in the task flow. If not, you can skip this step.

See: Identifying Customized Forms to Include in the Task Flow, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 212 Verify and Save the Workflow

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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 213 Generate a Task Flow From Oracle Workflow

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: Generate a Task Flow From Oracle Workflow, *Customizing, Reporting and System Administration in Oracle HRMS.*

Define Menus

Step 214 Define Menu Functions

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 customization options or task flows you should define your own menus.

Use the Form Functions window.

See: Defining Menu Functions, *Customizing, Reporting and System Administration in Oracle HRMS*.

Step 215 Define Menus

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, *Customizing, Reporting and System Administration in Oracle HRMS*.

Step 216 Disable the Multiple Windows Feature

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.

Attention: You must disable this feature on menu structures that access Oracle HRMS windows.

See: Disabling Multiple Windows, *Customizing, Reporting and System Administration in Oracle HRMS.*

Define 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 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, one for reporting.

- Standard HRMS: page 2 – 72
- Cross Business Group Responsibility: page 2 – 75

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: Reporting Users: page 2 – 77.

Define Standard HRMS Security

Set up standard HRMS security if your enterprise sets up a different responsibility for each Business Group.

Step 217 Ensure that the Enable Security Group option is Set

Ensure the Enable Security Groups profile option is set to No at site and application level.

Use the System Profiles Value window

See: System Profile Values, *Oracle Applications User's Guide.*

Step 218 Define a Security Profile

Use a view–all responsibility to define security profiles.

Use the Security Profile window.

See: Defining a Security Profile, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 219 Ensure Required Functions or Menus are Set Up

This is required for the responsibility. For menu functions calling customized forms or task flows, you must enter a parameter in the Parameter field of the Form Functions window.

See: Set up Menus, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 220 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: Set up Menus, *Customizing, Reporting and System Administration in Oracle HRMS.*

See: Request Groups Window, *Oracle Applications User's Guide.*

Step 221 Define a Responsibility

You need to define a responsibility.

Use the Responsibilities window.

See: Responsibilities Window, *Oracle Applications User's Guide*

Step 222 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: Security Profile

  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.

You can set also set up other User Profile Options.

Use the System Profile Values window.

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

Step 223 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 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, *Customizing, Reporting and System Administration in Oracle HRMS.*

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 customized subset of the localized or verticalized help files.

See: Customizing Oracle Applications Help, *Oracle Applications User's Guide.*

Step 224 Create Usernames and Passwords

You need to create usernames and passwords. Do not link responsibilities and security groups (Business Groups) to users in this window for HRMS, use the HRMS Assign Security Profile window. If you do enter a responsibility and security group in this window, 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 User window.

See: Users Window, *Oracle Applications User's Guide*

Step 225 Run Security List Maintenance Process (LISTGEN)

Oracle HRMS uses the Security List Maintenance process to generate the lists of organizations, positions, payrolls, employees and applicants that each security profile can access.

Attention: When you initiate the Listgen process you must enter the resubmission interval to run Listgen 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.*

Define Cross Business Group Responsibility Security

Create cross Business Group security if your enterprise wants to enable many Business Groups for one responsibility. This type of security is most commonly used by Service Centers.

Step 226 Set the Enable Security Groups Profile Option

Ensure the Enable Security Groups profile option is set to Yes at the application level.

Use the System Profiles Value window

See: System Profile Values, *Oracle Applications User's Guide.*

Step 227 Run the Enable Multiple Security Group Process.

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 228 Define a Security Profile

Use a view–all responsibility to define security profiles.

Use the Security Profile window.

See: Defining a Security Profile, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 229 Ensure Required Functions or Menus are Set Up

This is required for the responsibility. For menu functions calling customized forms or task flows, you must enter a parameter in the Parameter field of the Form Functions window.

See: Set up Menus, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 230 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: Set up Menus, *Customizing, Reporting and System Administration in Oracle HRMS.*

See: Request Group window, *Oracle Applications User's Guide.*

Step 231 Define a Responsibility

You need to define a responsibility.

Use the Responsibilities window.

See: Responsibilities Window, *Oracle Applications User's Guide.*

Step 232 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.

> **Note:** For Cross Business Group Responsibility security 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 User's Guide*

Step 233 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 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, *Customizing, Reporting and System Administration in Oracle HRMS.*

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 customized subset of the localized or verticalized help files.

See: Customizing Oracle Applications Help *Oracle Applications User's Guide.*

Step 234 Create Usernames and Passwords

You need to create usernames and passwords. Do not link responsibilities and security groups (Business Groups) to users in this

window for HRMS, use the HRMS Assign Security Profile window. If you do enter a responsibility and security group in this window, 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 User window.

See: Users Window, *Oracle Applications User's Guide*

Step 235 Assign Security Profiles

Associate a security profile with a user, responsibility and Business Group.

Use the Assign Security Profile window.

See: Assigning Security Profiles, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 236 Run Security List Maintenance Process (LISTGEN)

Oracle HRMS uses the Security List Maintenance process to generate the lists of organizations, positions, payrolls, employees and applicants that each security profile can access.

Attention: When you initiate the Listgen process you must enter the resubmission interval to run Listgen 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.*

Reporting Users

Step 237 Create a New Reporting User Oracle IDs

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 238 Register the New Oracle ID

Register the new ORACLE ID with Application Object Library.

Use the Register window.

Step 239 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, *Customizing, Reporting and System Administration in Oracle HRMS.*

Step 240 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.*

Define Audit Requirements

Step 241 Estimate File Sizing and Management Needs

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.

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.

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 242 Define Audit Installations

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.

See: Audit Installations Window, *Oracle Applications User's Guide*

Step 243 Define Audit Tables and Columns

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 User's Guide*

Step 244 Define Audit Groups

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 Tables Window, *Oracle Applications User's Guide*

Step 245 Activate AuditTrail Update Tables Process

To start the AuditTrail activity you must submit the *Activate AuditTrail Update Tables Process*.

Use the Submit a New Request window.

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

PART

II

Technical Essays

These essays provide technical information which may be required by the implementation team for initial data loading, customizing Oracle HRMS, or integrating it with other applications or processes. This part contains essays on the following topics:

- APIs in Oracle HRMS
- Oracle HRMS Data Pump
- DateTrack
- FastFormula
- Extending Security in Oracle HRMS
- Batch Element Entry
- Validation of Flexfield Values
- Payroll Processes
- Payroll Archive Reporter (PAR) Process
- Balances in Oracle Payroll
- Payroll Advice Report (UK Only)

CHAPTER

3 APIs in Oracle HRMS

APIs in Oracle HRMS

In common usage an Application Programmatic Interface, or API, is usually a logical grouping of all external process routines. For the Oracle HRMS products we have an API strategy that 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 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 3 – 3

  Describes how you can use the Oracle HRMS APIs and the advantages of this approach.

- Understanding the Object Version Number (OVN): page 3 – 6

  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 3 – 8

  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 3 – 23

  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 3 – 24

  Describes how the APIs validate key flexfield and descriptive flexfield values.

- Multilingual Support: page 3 – 25

  Explains how to use the Multilingual Support APIs.

- Alternative APIs: page 3 – 26

  Explains that we provide legislation–specific APIs for some business processes, such as Create Address.

- API Errors and Warnings: page 3 – 28

  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 3 – 30

  Shows how to load a batch of person address data and how to handle validation errors.

- WHO Columns and Oracle Alert: page 3 – 33

  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 3 – 34

  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 3 – 55

  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 3 – 56

  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 will be 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 3 – 19

Understanding the p_datetrack_delete_mode control parameter: page 3 – 20

Handling Object Version Numbers in Oracle Forms: page 3 – 56

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 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:

- The Publicly Callable Business Process APIs topic in the guide *Customizing, Reporting and System Administration in Oracle HRMS* and in the help system.
- 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.

Attention: 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:

```
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.

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

```
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 affect 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 following values:

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 the following grade rate values already exist in the pay_grade_rules_f table:

```
                    Effective       Effective_      Version_
Grade_rule_id       Start_Date      End_Date        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 following values:

```
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 following grade rate values 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 following grade rate values already exist in the pay_grade_rules_f table:

```
                Effective_          Effective_
Grade_rule_id   Start_Date          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 to the following:

```
                Effective_           Effective_
Grade_rule_id   Start_Date           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 3 – 25

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 follows:

```
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 3 – 34

For further information about, and solutions to, some problems that you may encounter with flexfield validation, see: Validation of Flexfield Values: page 9 – 2.

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, 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.

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 3 – 53

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 follows:

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. 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_commit_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_id%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 error 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.

Warning: 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 3 – 35
2. Identify the data values available at the user hooks you intend to use. See: Data Values Available at User Hooks: page 3 – 39
3. Implement your extra logic in a PL/SQL server–side package procedure. See: Implementing Extra Logic in a Separate Procedure Package: page 3 – 41
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 3 – 43
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 3 – 48

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.

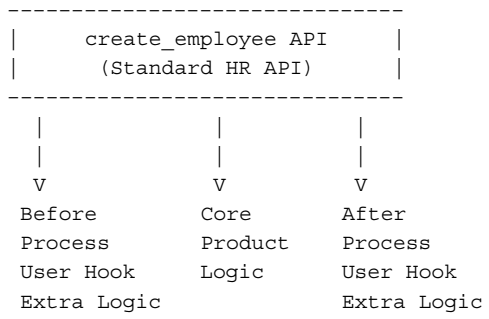
See: API User Hook Support Scripts: page 3 – 55

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 topic in the guide *Customizing, 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 will be executed after the validation specific to the current table columns has been successfully completed and immediately after the corresponding table DML statement.

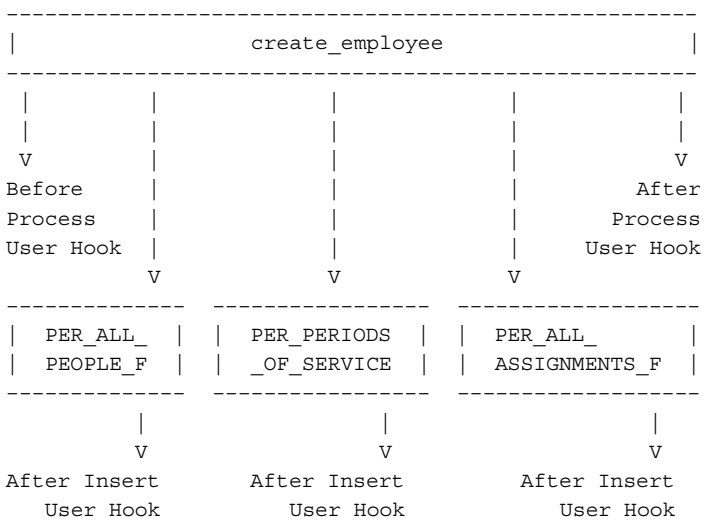
These row handler user hooks are provided after the DML has been completed for two reasons:

- 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 3 – 55

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.

Attention: 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 3 – 55

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, you need to 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 order of hook calls. Lowest numbers are processed first.

  The user hook mechanism is also used by Oracle to supply legislation specific 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

HR_API_MODULES 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

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:

```
User Hook Type    API_HOOK_TYPE
---------------   -------------
After Insert      AI
After Update      AU
After Delete      AD
Before Process    BP
After Process     AP
```

Warning: 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 *Technical Reference Manual*.

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 legislations and vertical versions of the products. Calls to this PL/SQL are also generated into the hook package body.

> **Warning:** 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.

Attention: 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.

Attention: 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.

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:

```
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.

Attention: 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.

Suggestion. 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.

Suggestion. 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.

Warning: Any changes you make may affect product functionality and may invalidate your support agreement with Oracle, and prevent product upgrades.

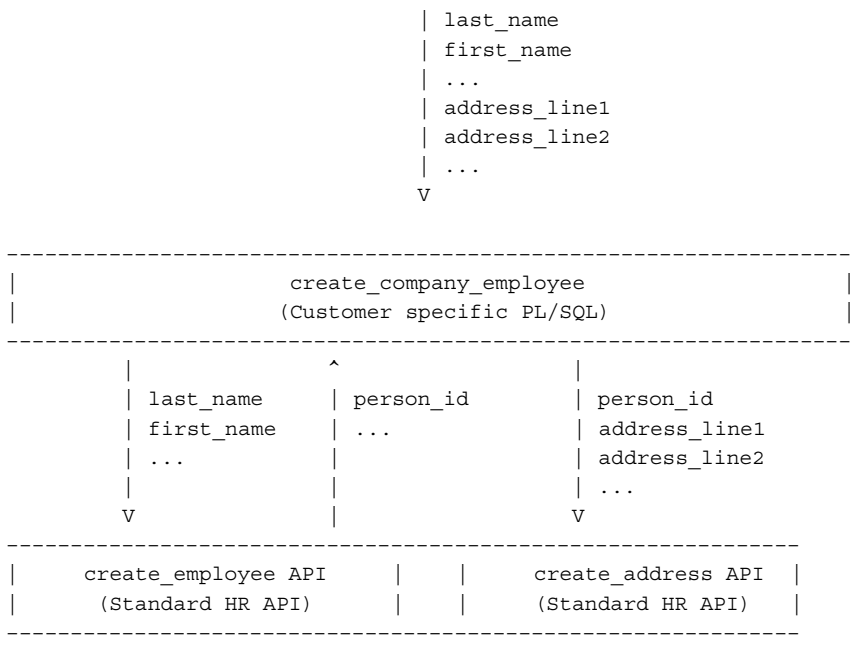
Oracle Applications supports direct calls to the published APIs. Direct calls to any other server–side package procedures or functions written as part of the Oracle HRMS product set are not supported, unless explicitly specified.

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.



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.

```
      OVN in
Row   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.

```
      OVN in
Row   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.

```
      OVN in
Row   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.

```
      OVN in                    New_OVN
Row   Database   OVN in Form    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.

```
      OVN in                    New_OVN
Row   Database    OVN in Form   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.

```
      OVN in                    New_OVN
Row   Database    OVN in Form   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.

```
      OVN in                    New_OVN
Row   Database    OVN in Form   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.

```
      OVN in                    New_OVN
Row   Database    OVN in Form   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.)

```
      OVN in                    New_OVN
Row   Database    OVN in Form   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.

```
      OVN in                    New_OVN
Row   Database   OVN in Form    in Form
---   --------   -----------    -------
A     7          7              7
```

8. The user continues with the update, the update API is called, and the commit is successful.

```
      OVN in                    New_OVN
Row   Database   OVN in Form    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 ensure 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_number
      ,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
API
    :<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_number
      ,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') then
      --
      -- 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)
and
         (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;
```

CHAPTER

4 Oracle HRMS Data Pump

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 document 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 *Technical Reference Manual*.

The Oracle HRMS Data Pump does not support all of the APIs that are delivered with Oracle HRMS. For the list of supported APIs, see: APIs Supported by Data Pump: page 4 – 36. 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 4 – 28.

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 and you must run it before you try to do any data validation or loading using the predefined APIs.

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 two parameters:

- Batch name
- Processing mode

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'.

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_PARAMETERS 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 4 – 17.

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 4 – 8.

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 4 – 14.

   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 4 – 17.

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 4 – 18.

6. If necessary, rerun Data Pump to load corrected batch lines.

   See: Rerunning the Data Pump Process: page 4 – 22.

   Repeat 5 and 6 until all lines are successfully loaded.

7. Optional. Purge data from the batch tables.

   See: Purging Data: page 4 – 22.

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 all APIs:

```
sql> execute hr_pump_meta_mapper.generateall;
```

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',
'create_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.

> **Note:** You must call the Meta–Mapper before using the Data Pump to load any data. After calling the Meta–Mapper for all the required APIs, restart the concurrent manager before running the Data Pump.

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 4 – 31 .

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.
```

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.

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_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.

> **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 4 – 28 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 correct format for all dates is now YYYY/MM/DD.
- 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.

Suggestion: 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 4 – 17 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.

> **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, the personal payment method and element entry APIs are known to have problems in this area.

Default Values for API Parameters

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.

Running the Data Pump Process

Use the Submit Reports and Processes form to start the Data Pump Engine process. It takes two 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.

  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.

Running In Parallel

To enable parallel processing you set a value for the THREADS parameter in PAY_ACTION_PARAMETERS.

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. When you have completed Data Pump processing you should reset the THREADS parameter so that the parameters for Data Pump are not transferred to normal payroll processing.

Other Parameters

There are three 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 4 – 18.

Checking Run Status

The Data Pump runs as a concurrent process so you can check its status at any time using the View Concurrent Requests window. Failure is reported by the concurrent manager only 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 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.

Suggestion: These 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

You can combine any number of these options by concatenating the values, separated by a colon. For example:

```
update pay_action_parameters
set    parameter_value = 'MSG:RRI:RRP'
where  parameter_name  = 'PUMP_DEBUG_LEVEL';
```

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, or the processing halts 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 a row in the HR_PUMP_BATCH_EXCEPTIONS table. 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 – connected, 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.

Attention: 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_name,
```

```
      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 || '
employees.');
         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.

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 identify 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 (initial 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	

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
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_PARAMETERS

```
Name                           Null?    Type
-------------------------------------- ----
PARAMETER_NAME                 NOT NULL VARCHAR2(30)
PARAMETER_VALUE                NOT NULL VARCHAR2(80)
```

APIs Supported by Data Pump

This list shows the API modules supported by the first release of Data Pump.

Package Name	Procedure Name
HR_EMPLOYEE_API	CREATE_EMPLOYEE
	CREATE_GB_EMPLOYEE
	CREATE_US_EMPLOYEE
HR_ASSIGNMENT_API	ACTIVATE_EMP_ASG
	ACTUAL_TERMINATION_EMP_ASG
	CREATE_SECONDARY_EMP_ASG
	CREATE_GB_SECONDARY_EMP_ASG
	CREATE_US_SECONDARY_EMP_ASG
	UPDATE_EMP_ASG
	UPDATE_GB_EMP_ASG
	UPDATE_US_EMP_ASG
	UPDATE_EMP_ASG_CRITERIA
	SUSPEND_EMP_ASG
HR_JOB_API	CREATE_JOB
HR_POSITION_API	CREATE_POSITION
	UPDATE_POSITION
HR_VALID_GRADE_API	CREATE_VALID_GRADE
HR_PERSON_ADDRESS _API	CREATE_PERSON_ADDRESS
	CREATE_GB_PERSON_ADDRESS
	CREATE_US_PERSON_ADDRESS
	UPDATE_PERSON_ADDRESS
	UPDATE_GB_PERSON_ADDRESS
	UPDATE_US_PERSON_ADDRESS

Package Name	Procedure Name
HR_CONTACT_API	CREATE_PERSON
HR_CONTACT_REL_API	CREATE_CONTACT
PY_ELEMENT_ENTRY_API	CREATE_ELEMENT_ENTRY
	UPDATE_ELEMENT_ENTRY
	DELETE_ELEMENT_ENTRY
HR_GRADE_API	CREATE_GRADE_RATE_VALUE
	UPDATE_GRADE_RATE_VALUE
	DELETE_GRADE_RATE_VALUE
HR_PERSONAL_PAY_METHOD_API	CREATE_PERSONAL_PAY_METHOD
	CREATE_GB_PERSONAL_PAY_METHOD
	CREATE_US_PERSONAL_PAY_METHOD
	UPDATE_PERSONAL_PAY_METHOD
	UPDATE_GB_PERSONAL_PAY_METHOD
	UPDATE_US_PERSONAL_PAY_METHOD
HR_SIT_API	CREATE_SIT
HR_APPLICANT_API	CREATE_APPLICANT
	CREATE_GB_APPLICANT
	CREATE_US_APPLICANT
HR_JOB_REQUIREMENT_API	CREATE_JOB_REQUIREMENT
HR_POSITION_REQUIREMENT_API	CREATE_POSITION_REQUIREMENT
HR_PERSON_API	UPDATE_PERSON
	UPDATE_GB_PERSON
	UPDATE_US_PERSON

Package Name	Procedure Name
HR_PAY_SCALE_API	CREATE_PAY_SCALE_VALUE
	UPDATE_PAY_SCALE_VALUE
	DELETE_PAY_SCALE_VALUE
HR_EX_EMPLOYEE_API	ACTUAL_TERMINATION_EMP
	FINAL_PROCESS_EMP

CHAPTER

5 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

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

Example of DateTracked Table Contents

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.

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		

Example of Query Results for a DateTracked Table

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.

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.

Global Variables at DT_SELECT_MODE Event

Global Variable Name	Read/Write	Description
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.

Global Variables at DT_SELECT_MODE Event

Attention: 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;
```

How to Create and Modify DateTrack History Views

DateTrack History is available in most windows where DateTrack is available to enter information. It lets you track when changes have been made to a record, which fields were changed, and by whom. You can select the fields you want to focus on and view the changing values of these fields over time.

DateTrack History is available from a button on the toolbar.

Additional Information: For more information on DateTrack, refer to *Using Oracle HRMS – The Fundamentals*.

When you request DateTrack History, the information is extracted either from the base table or the DateTrack History view for the table, if one exists. You can create new views or modify the existing views to customize the information displayed. For example:

- 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.
- You can decide which fields are displayed, by modifying the views.
- You can modify views so that column names display an alias for the meaningful names you have defined for descriptive flexfield segments.
- You can determine which view to use depending on criteria of your choice, such as the Business Group ID.

This essay provides the background information you require before modifying or creating DateTrack History views.

What Happens When You Request DateTrack History

When you request DateTrack History from a DateTracked window, the window passes the name of the base table, the name of the unique id field, and the value of the unique id to DateTrack History.

Before the DateTrack History Change Field Summary window displays, the code checks whether the expected DateTrack History view exists. Usually the name of the 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 code 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 may be examples where the History view name does not follow this naming convention.

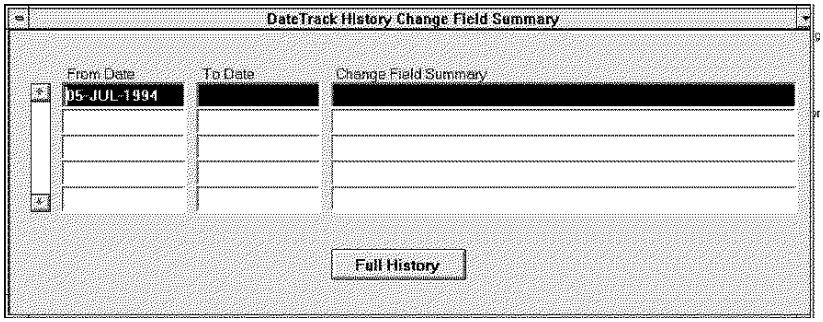
If the expected view does exist, the code uses it; otherwise, in some cases, it attempts to extract information from the base table.

When a view exists, information about the entity name and column prompts is read from the DateTrack tables. These are:

- 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.

Change Field Summary



Rules for Creating or Modifying DateTrack History Views

Where possible all DateTrack History views should adopt the naming convention described above. That is, they should have the same name as the corresponding base table, except that the suffix _F is replaced by _D. If you are using custom library to specify an alternative view, the view name can be 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.

You must 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. 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.

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.

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_PEOPLE_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:

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

DateTrack History Views

View Name	Based on (table)	View Creation Script
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
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_STEPS_D	PER_SPINAL_POINT_STEPS_F	pedtspst.sql
PER_PERSON_TYPE_USAGES_D	PER_PERSON_TYPE_USAGES_F	pedtptu.sql
PER_CONTRACTS_D	PER_CONTRACTS_F	pedtctc.sql

DateTrack History Views

CHAPTER

6 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.

Entities in the Dictionary

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_ENTITY 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.

Step 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.

Step 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.

Step 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).

Step 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.

Step 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_ENTITY, 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.

Step 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.

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 6 – 16

- 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 6 – 21

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 6 – 24

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.

Attention: 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	The input name, such as RATE, or ASSIGN-MENT_ID
DATATYPE	Can be DATE, NUMBER, or TEXT

Inputs Table

Inputs Table	
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.

Inputs Table

Outputs Table	
NAME	The output name, such as RESULT1, or MESSAGE
DATATYPE	Can be DATE, NUMBER, or TEXT
VALUE	The actual value returned from the formula

Outputs table

> **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.

Attention: 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 6 – 19.

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.

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

Parameters to init_formula

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).

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

Parameters to run_formula

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
includes
    -- 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_canonical(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_cnt).name);
```

```
      hr_utility.trace('output datatype : ' ||
l_outputs(l_out_cnt).datatype);
      hr_utility.trace('output value    : ' ||
l_outputs(l_out_cnt).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.

Parameter Name	Data Type	Comments
p_formula_id	number	Formula_id to execute
p_effective_date	date	Effective execution date

Parameters to init_formula

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.

Parameter Name	Data Type	Comments
p_input_name	varchar2	Name of input to set
p_value	varchar2	Input value to set

Parameters to set_input

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.

Parameter Name	Data Type	Comments
p_input_name	varchar2	Name of input to set
p_return_value	varchar2	Value of varchar2 output

Parameters to get_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_formula_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_canonical(l_input1));
    ff_client_engine.set_input('INPUT2',
fnd_date.date_to_canonical(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.

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

Parameters to run_id_formula

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.

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

Parameters to run_name_formula

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).

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

Values for FF_DEBUG Profile Option

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.

CHAPTER

7

Extending Security in Oracle HRMS

Extending Security in Oracle Human Resources

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 chapter on Security in *Customizing, Reporting and System Administration in Oracle HRMS*, and to the setup steps in *Implementing Oracle HRMS.*

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

Restricted security profiles are available only to users of Oracle Human Resources, Oracle Payroll, and Oracle Advanced Benefits. Notice that Oracle Training Administration does not make use of restricted security profiles.

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 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
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

Secure Table and Views

Table or View	Description
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

Secure Table and Views

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 is dependent 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 will be 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.

Here is the text of the HR_ORGANIZATION_UNITS secure view:

```
CREATE OR REPLACE VIEW HR_ORGANIZATION_UNITS AS
SELECT *
FROM HR_ALL_ORGANIZATIONS HAO
WHERE DECODE(HR_SECURITY.VIEW_ALL, 'Y', 'Y',
           HR_SECURITY.SHOW_RECORD
           ('HR_ALL_ORGANIZATION_UNITS',HAO.ORGANIZATION_ID))='Y'
```

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.

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. An additional security profile list table (PER_PERSON_LIST_CHANGES) is populated on employee and applicant termination to enable terminated employees and applicants to continue to be visible; the PERSON_LIST table references only current employees and applicants.

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 LISTGEN process. They are also written to when some relevant business processes are performed through Oracle HR, for employee, employee hire or transfer.

Security Processes

Three processes are used to implement Oracle HRMS security:

- Grant Secure Role Permission (ROLEGEN)
- Generate Secure User (SECGEN)
- Create Security Lists (LISTGEN)

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 chapter on Security in *Customizing, Reporting and System Administration in Oracle HRMS* for details of how to submit SECGEN and LISTGEN from the Submit Requests window. This section describes how the processes work.

Note: There is another security process called GLISTGEN. Use this to generate lists for *global* security profiles. These are security profiles that are not associated with a Business Group. They secure organizations and people through a global (cross–Business Group) organization hierarchy.

Global security profiles cannot be used in Oracle HRMS at present. They can be used for other Oracle applications.

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. And 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 nor 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.

LISTGEN – Create Security Lists Process

You should run LISTGEN periodically (for example, nightly) to refresh the security lists upon which the secure views are built.

LISTGEN is a PL/SQL procedure that you submit from the Submit Requests window.

LISTGEN builds the security lists from the organization and position hierarchies by performing tree walks on the PER_ORG_STRUCTURE_ELEMENTS and PER_POS_STRUCTURE_ELEMENTS tables. It uses the parent–child relationship between the nodes and starts with the specified top node. It uses the current version of the hierarchy, as of the date passed to the process as the effective run date.

For each security profile, LISTGEN checks that the organization named as the top organization exists in the current version of the hierarchy. If it does not, LISTGEN writes an error message to a log file and fails with an error status. This might happen if a new version of a hierarchy did not contain an organization referenced as a top organization in a security profile.

A similar check is made for the top position, if specified.

For each security profile, LISTGEN performs the following steps:

1. If the View All flag is Y, LISTGEN ends leaving all security lists empty for the specified security profile.
2. Builds a payroll list.

   If the View All Payrolls flag is Y, LISTGEN leaves the payroll list empty. If the View All Payrolls flag is N, LISTGEN checks the Include Payroll flag. If this flag is Y, LISTGEN makes a list of all payrolls in the pay_security_payrolls list. If the flag is N, LISTGEN 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, LISTGEN leaves the organization list empty. If this flag is N, LISTGEN builds a list of all organizations below the top one you specified for the organization hierarchy you chose on the Define Security Profile screen. If the Include Top Organization flag is Y, the top organization you specified is included in the list. The Business Group is always 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, LISTGEN builds a list of all positions within the organizations on the organization list. If this flag is N, LISTGEN builds a list of all positions below the top one you specified for the position hierarchy you chose on the Define Security Profile screen. 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.

   If the View All Positions flag is N, LISTGEN builds a list of all employees or applicants with current assignments to positions in the position list, unless they are also assigned to a payroll excluded from the payroll list. LISTGEN also includes people who are not assigned to a position but are assigned to a payroll in the payroll list, or to any payroll if the View All Payrolls flag is Y.

   The people in the list have current assignments as of the date passed into LISTGEN, or are the first assignments for a person starting in the future who does not have a previously terminated assignment. New starters in the future are therefore visible through the secure view.

   If the View All Organizations flag is N and the View All Positions flag is Y, LISTGEN builds a list of all people with current assignments to organizations in the organization list. If the View All Payrolls flag is N, the list is restricted to people with assignments to payrolls in the payroll list, or with no payroll assignments.

   People not yet assigned are included in the person list for every security profile.

6. Adds person list changes.

   Employees or applicants visible to security profiles at the point of their termination should continue to be visible after termination. To enable this, the termination forms insert a row into the person list changes table for each security profile that can see the person at termination.

   LISTGEN 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 no current application for the person. It only adds people if they are not already in the list.

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 or HXT.

   Consult *Oracle Applications Coding Standards* for information on valid prefixes for custom code.

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;
   ```

CHAPTER

8 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

This is done in three parts:

1. Create a control total type in Lookup Values under the type CONTROL_TYPE. See: Adding Lookup Types and Values, *Customizing, Reporting, and System Administration in Oracle HRMS*.
2. Create the SQL code necessary to perform the validation.
3. Add the control total name and expected value into the Control Totals screen for the batch. See: Entering a Batch Header, *Managing Compensation and Benefits Using Oracle HRMS*.

Task 2 is the most complex and is elaborated below.

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 check the total value of the first column of the lines, 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 = 'TOT1' THEN
-- Calculate the total
    SELECT SUM(value_1) 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(rror)
      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 total "Pay Value":

```
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_name
        FROM pay_batch_lines
        WHERE batch_id = p_batch_id;
--
    r1 c1%ROWTYPE;
    gtotal NUMBER;
    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 = 'TOT2' THEN
      gtotal := 0;
--
-- Loop through each element in the batch lines
      FOR r1 IN c1 LOOP
--
-- Find out the value number that 'Pay Value' is in
        SELECT display_sequence
          INTO value_num
          FROM pay_input_values iv,
               pay_batch_headers bh,
               pay_element_types_f et
          WHERE bh.batch_id = p_batch_id AND
                iv.business_group_id = bh.business_group_id AND
                et.element_name = r1.element_name AND
                iv.element_type_id = et.element_type_id AND
                iv.name = 'Pay Value';
--
-- Create an SQL string to add the values
        sqlstr := 'SELECT SUM(value_' || value_num || ') ' ||
                    'FROM pay_batch_lines ' ||
                    'WHERE batch_id = ' || p_batch_id || ' AND '
||
                          'element_name = ''' || r1.element_name
|| '''';
--
```

```
-- 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);
--
-- Add the total to the grand total of all Pay Values
        gtotal := gtotal+total;
      END LOOP;
--
-- Check the grand total matches the user entered value and create
an error message
-- if it doesn't
      IF gtotal <> p_control_total THEN
        p_message := 'Control Total ' || p_control_type || '
expected ' ||
                     p_control_total || ' but got ' || gtotal;
        p_status := 'E';
      END IF;
    END IF;
  END check_control;
```

CHAPTER

9

Validation of Flexfield Values

Validation of Flexfield Values

Oracle Self Service HR, Application Data Exchange, 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
- Using the segment separator in data

The rest of this essay explains these five 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.

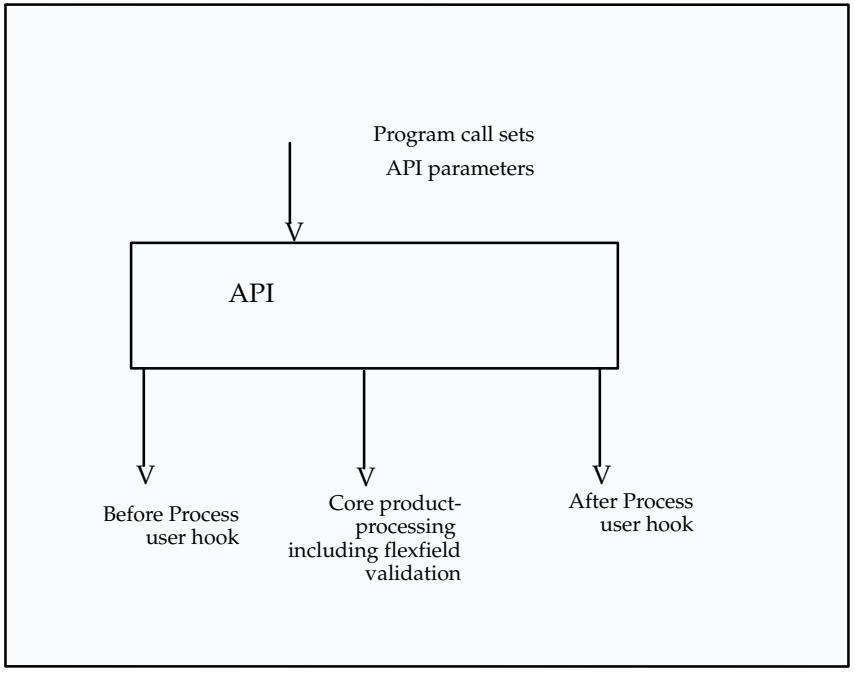
However, when the APIs are executed directly in a SQL*Plus database session, there is no application log–on. If the profile is not defined at site level, its value will be null. Even if the profile is defined at site level, this may not give the appropriate values. For example, the

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



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" 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.

Attention: There may be some instances in the Self Service screens where it is not possible to display these flexfield values. This is because their 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.

Using Segment Separator in Data

The API flexfield validation logic does not support using the character selected as the flexfield segment separator within the data itself. For example, if you use a period as the segment separator, you cannot use periods within your data. A data value such as “Sales N. Region” will be interpreted as two segments: Sales N and Region. You may see errors of the type “ORA–20001: Too many segment values entered.” or validation errors because the truncated value (Sales N) is not valid.

Recommended Solution

This is a data or configuration problem for you to resolve in one of the following ways:

- Changing the segment separator to another character. Care needs to be taken to ensure existing data is modified to contain the same value.
- Removing the segment separator characters from the data. This may involve making other changes to the flexfield definition to avoid the need to use the segment separator in the data. For example, the flexfield definition could use organization ID instead of organization name (while still displaying the name to end users).

CHAPTER

10 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.

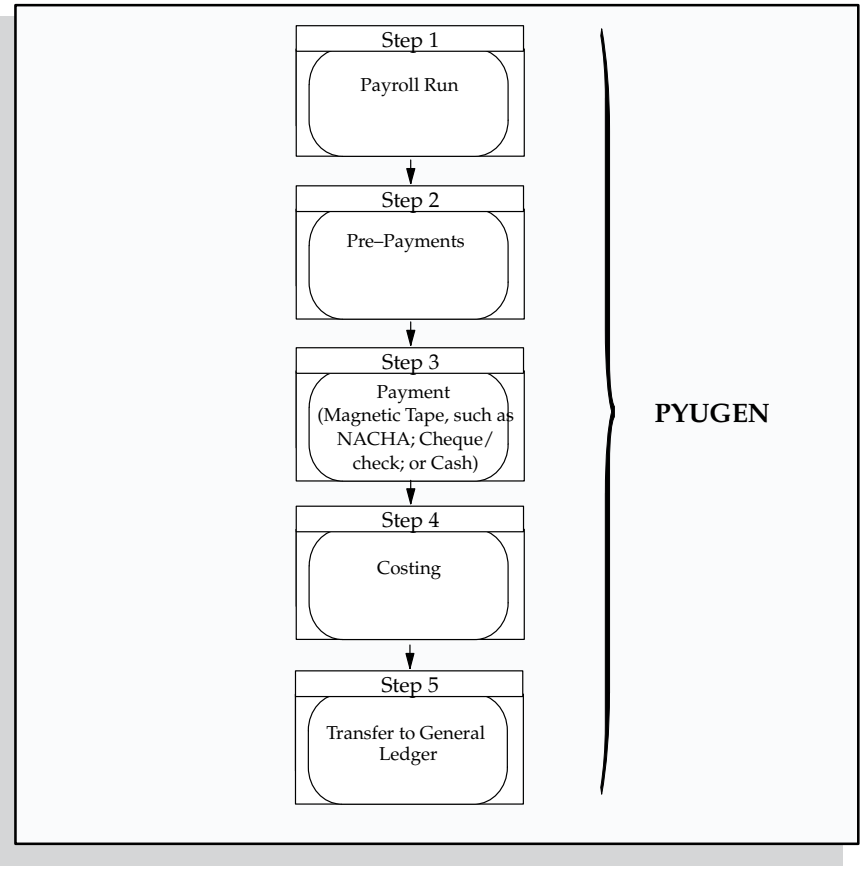
PYUGEN

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



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 Administator'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 10 – 64

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 10 – 6

Pre–Payments Process: page 10 – 58

Payment Processes: page 10 – 18

- Magnetic Tape Process: page 10 – 19
- Cheque Writer/Check Writer Process: page 10 – 38
- Cash Process: page 10 – 47

Costing Process: page 10 – 48

Transfer to General Ledger Process: page 10 – 52

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 11 – 2 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 10 – 53.

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 12 – 2.

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 12 – 8.

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 12 – 7.

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 12 – 7.
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.

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, there are three types of payment process that you can run:

- The Magnetic Tape process – MAGTAPE

  See: Magnetic Tape Process: page 10 – 19

- The Cheque process – CHEQUE

  See: Cheque Writer/Check Writer Process: page 10 – 38

- The Cash Payments process – CASH (UK only)

  See: Cash Process: page 10 – 47

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 11 – 2

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 10 – 25

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.

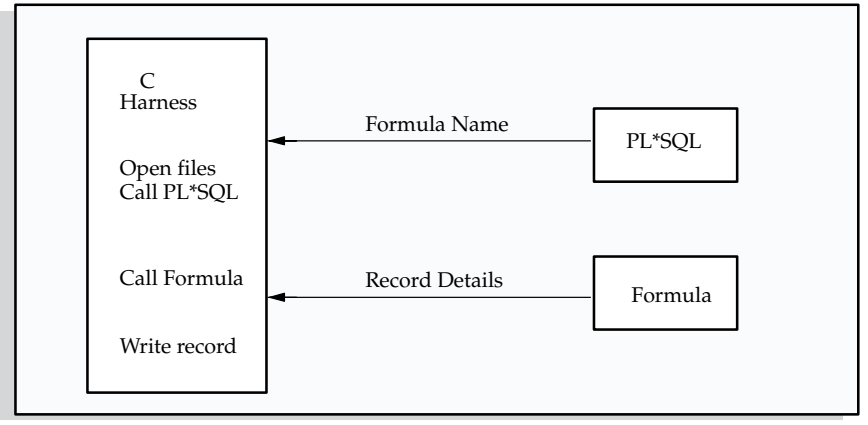
Similarly, the actual records have strict field place and length requirements. For example:

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.

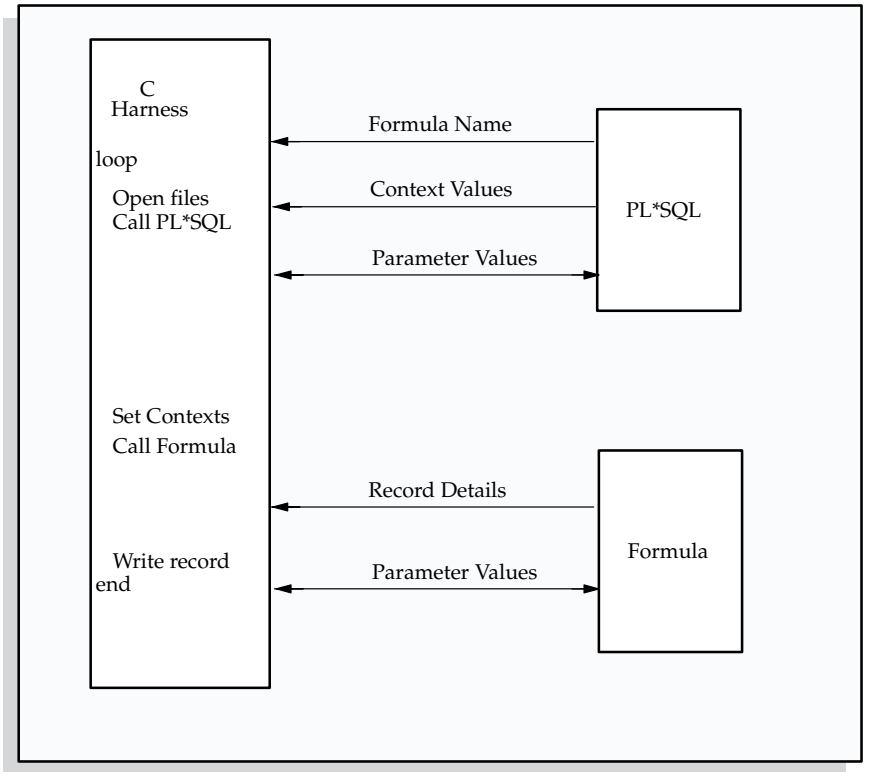
The process of getting the formula and record name, then writing the record to tape is repeated until all the records are processed.

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.

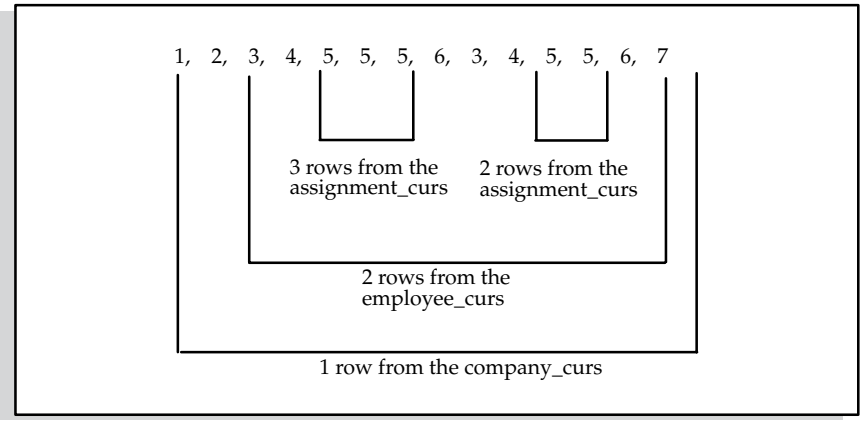
  Similarly, if the formula is a Repeated overflow (set to R), and the TRANSFER_RUN_OVERFLOW flag is set to Y then that formula is continually repeated until the formula does not return TRANSFER_RUN_OVERFLOW set to Y.
- Exec.last can apply to all the types of formula but most commonly the intermediate 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:

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_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_pay-
            ment_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 legisla-
           tion_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-YYYY');
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.

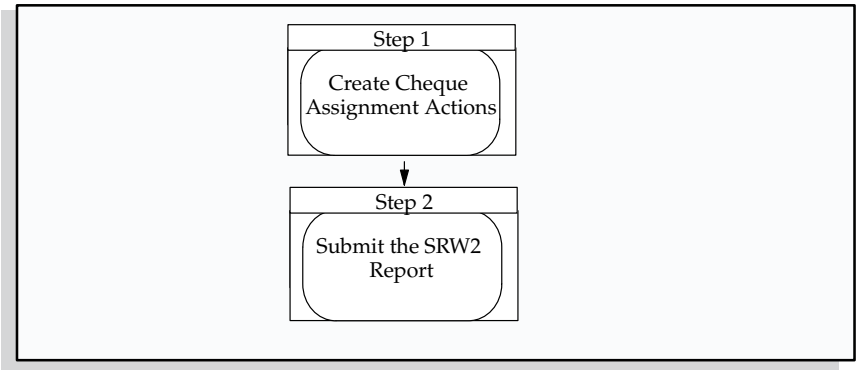
You run the Cheque Writer process to produce cheque payments for unpaid pre–payment actions. Before you run the process, you need to set up certain things, for example, the SRW2 report and the 'order by' option to sequence cheques (if required).

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 10 – 62 (Pre–Payments)

   See: Running the Magnetic Tape Payments Process: page 10 – 19

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 10 – 45

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 behaviour 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.

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	*Work Structure*		*Earnings and Deductions*			
	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	*Cost Center*		*Product Line*				
	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						
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: $1,000.00
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 = 1000/4000 = .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 = 0.6667
Product Line H201 overhead = Total Wages overhead $75
x .6667 = $50.00
Ratio for Wages distribution, Product Line H202 =
1000/3000 = 0.3334
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
```

Distribution of Overhead Over Cost Center and Production Line Totals

Account Code	*Cost Center*	*Product Line*	
	Production	***H201***	***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

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.

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

Processes and Action Types	Classification	Insert Interlock Rows?
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 10 – 54.

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.

	Payroll Action		Assignment Action	
Action Type Name	**Retry**	**Rollback**	**Retry**	**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

Action Type Name	Retry	Rollback	Retry	Rollback
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.

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 10 – 60

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.
   - Enter the Lookup Values window and query the Lookup type of CASH ANALYSIS.
   - Add the new Cash rule with the meaning and description fields set to TENS AND FIVES.
   - 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 10 – 53

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.

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_ASSIGN_END	See later	
LOG_ASSIGN_START	See later	
LOGGING	See later	

MAX_ERRORS_ALLOWED	1 or more	CHUNK_SIZE or 20 (if no chunk size)
MAX_SINGLE_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.

Attention: 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.

Attention: 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.

Logging Parameters

LOGGING

```
Parameter Name: LOGGING
Parameter Value: G, M, P, E, L, B, I, R, F, C, Q
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: Only use the trace facility to help with the investigation of problems. Setting the value to 'Y' will cause a significant deterioration in database performance. If you experience a significant problem with the performance of your payroll processes, you should always 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.

CHAPTER

11

Payroll Archive Reporter Process

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 11 – 4.

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 11 – 8

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 11 – 8.

5. Provide a SRS (Standard Report Submission) definition from which users can launch the PAR process.

   See: Provide an SRS Definition for the PAR Process: page 11 – 10.

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 11 – 11.

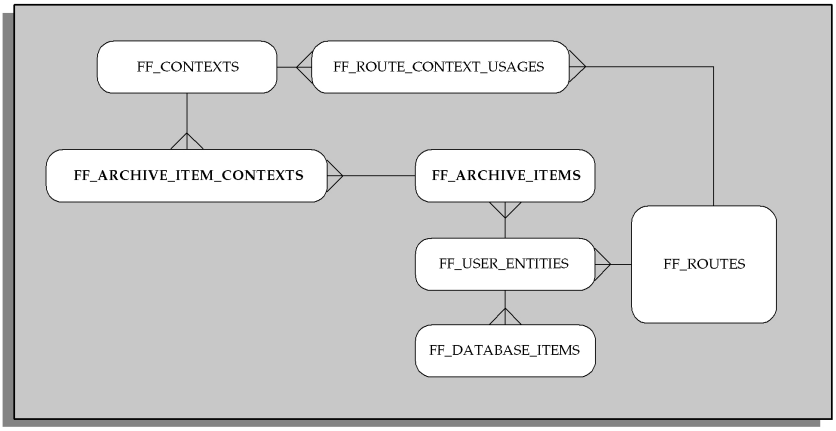
Create Database Items for Archiving

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 11 – 13.

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 11 – 11.

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,
pactid,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:

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 *

Parameters for the PAR Process

* 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 follows:

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	
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 11 – 8.
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 11 – 8.

Columns of the PAY_REPORT_FORMAT_MAPPINGS_F Table

Column Name	Type	Comments
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 11 – 7 and Examples: INITIALIZATION_CODE and ARCHIVE_CODE: page 11 – 13). 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 11 – 7 and Examples: INITIALIZATION_CODE and ARCHIVE_CODE: page 11 – 13.
MAGNETIC_CODE	VARCHAR2(60)	The standard generic magnetic tape driving PL/SQL procedure (see: Magnetic Tape Process: page 10 – 19). 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.
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

Columns of the PAY_REPORT_FORMAT_MAPPINGS_F Table

Column Name	Type	Comments
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.

Columns of the PAY_REPORT_FORMAT_MAPPINGS_F Table

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
date) 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_id
                                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;
```

CHAPTER

12 Balances in Oracle Payroll

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 10 – 10.

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 12 – 24.

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.

Attention: 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-M
M-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

   – feed checking
3. 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 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.

Attention: 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.

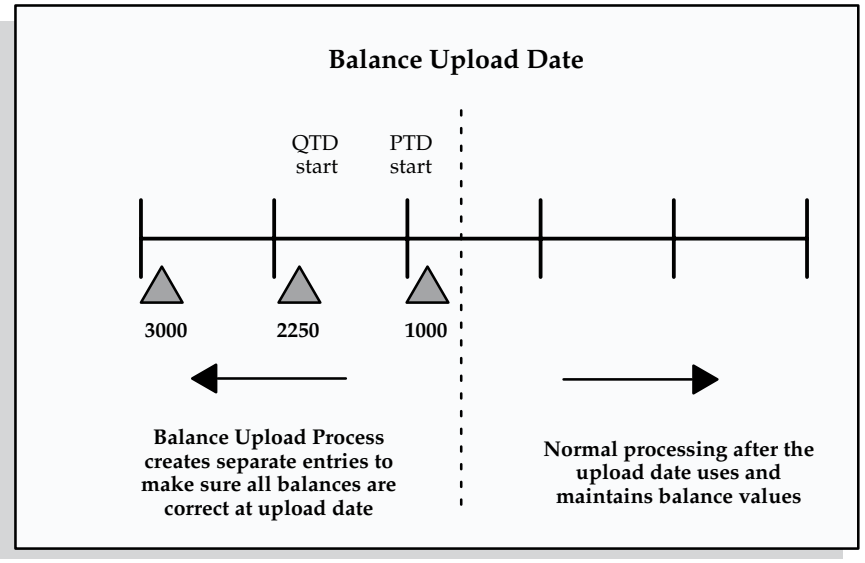
Consider the following example of three dimensions for gross pay balance values for one employee.

- 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



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.

Attention: 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.

Attention: US 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

Batch Name	Batch Ref	Batch Source	Payroll	Upload Date
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 only)	
JURISDICTION_CODE (US 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.

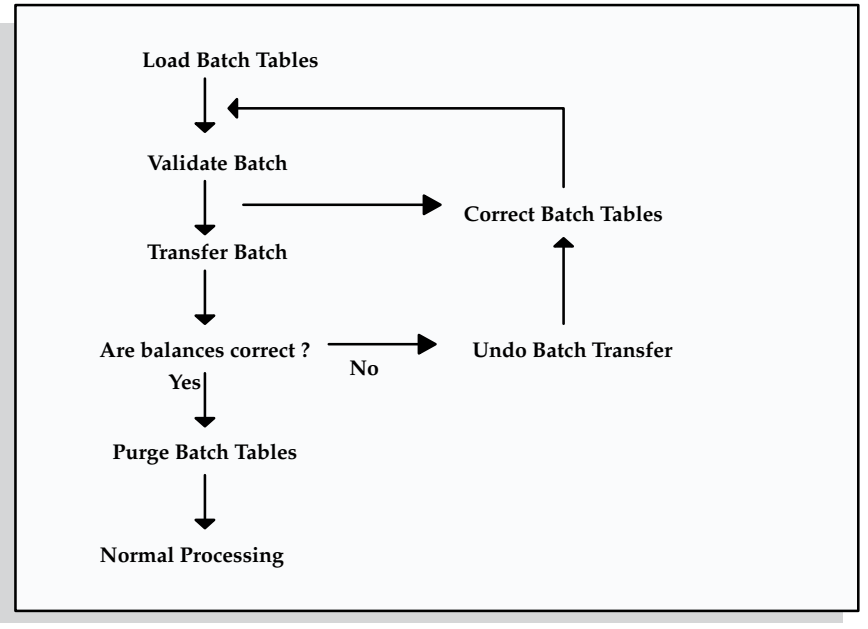
Use Purge mode only when you are sure that the balances for all assignments in a batch have been successfully entered into the HRMS database.

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.

   Attention: The next step applies to US 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 12 – 12.
   - 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 users the context may include a GRE and a jurisdiction (federal, state or local).

     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

Attention: 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): page 12 – 27

Suggestion: 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_value 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 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_action_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_action_id);
```

Including Balance Values in Reports (UK Only)

This section describes the PL/SQL interface for the GB balance function that enables you to access balance values for inquiry and reporting tools.

Advantages

Using this PL/SQL function to retrieve balance values has several advantages:

- You can easily call the function from a form or report.
- The function accesses latest balance values, where they exist.

  This optimizes performance automatically.
- You can call the function from a user–defined view.

  This is because the function has pragma levels WNPS and WNDS set (Write No Package State, and Write No DML).

The Balance Function

The interface to the 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:

- The GB Balance User Exit works in two modes: date and assignment action mode.

  For the balance function the interface is simplified using the PL/SQL overloading feature. The same function name is used, but different parameters are passed in according to the mode. If in Date Mode, the function calculates the values using the assignment action previous to the date passed in. The value obtained is checked to make sure it hasn't expired between the assignment action that it represents and the date passed in.
- The function uses the rubric of 'quickest value first'.

  If a value can be retrieved from the latest balances table, it will be. This is so that performance of the code is optimized.

Although one interface is used to directly call the GB Balance value function, the PL/SQL code resides in three packages:

- hr_dirbal
- hr_gbbal

- hr_routes

Function: Get balance value (Assignment Action mode):

```
function get_balance (p_assignment_action_id     in      number,
                      p_defined_balance_id   in      number)
return number;
```

Function: Get balance value (Date mode):

```
function get_balance (p_assignment_id            in      number,
                      p_defined_balance_id   in      number,
                      p_effective_date       in      date)
return number;
```

The balance value is returned by these functions. The parameters required for the function have been kept to a minimum.

Example

Supposing we take an assignment action id of 12345 and a defined balance id of 111:

```
l_balance := hr_dirbal.get_balance (12345, 111);
```

This would return a balance value, using the Assignment Action mode call to the package.

Supposing we take an assignment id of 2, the same defined balance id, and an arbitrary date:

```
l_balance := hr_dirbal.get_balance
(2,111,to_date('01/01/1998','DD/MM/YYYY'));
```

This would return a balance value, using the Date mode call to the package.

Legislative Balance Initialization (UK Only)

Balance Initialization Elements

The following elements need to be linked to all payrolls from the date 01–Jan–0001 in order to permit balance initialization of predefined balances.

- Setup Court Order Balance
- Setup NI Balance 1
- Setup NI Balance 2
- Setup NI Car Balance
- Setup Tax Balance

The inputs from these elements provide the initial balance feeds for predefined balances.

Supported Dimensions

The following dimensions are currently supported:

- ASG_PROC_YTD
- ASG_YTD
- ASG_TD_YTD
- ASG_STAT_YTD
- PER_TD_DIR_YTD
- ASG_PROC_PTD
- ASG_ITD

Predefined Balances That May Need Initializing

The following predefined balances may need initializing if you migrate to Oracle Payroll in the middle of the financial year.

Note that the statutory balances need to be initialized as they are reported on the P35 End of Year return. For this purpose the Dimensions ASG_TD_YTD and ASG_STAT_YTD are used. NI Category balances such as NI A Able, NI A Employee, NI A Employer, NI A Total should be kept in step. For example:

- If you load the NI A Able balance you must also load the NI A Employee balance.

- If you load the NI A Employer balance you must also load the NI Employer balance.

Similarly Taxable Pay and PAYE should be kept in step. NI Y is reported a year in arrears – either set up the NI Y in the year that it was accrued or set up NI Y Last Year in the year it is reported.

For directors the dimension PER_TD_DIR_YTD only needs to be initialized if it differs from the ASG_TD_YTD figure (that is, the director has been appointed part way through the financial year). Some of the Court Order elements use the ELEMENT_ITD dimension, which is tied to a particular element entry via an ORIGINAL_ENTRY_ID. You will need first to set up the element entry and enter the Element_entry_id from that as the ORIGINAL_ENTRY_ID on pay_balance_batch_lines.

Balance Name	Dimension	On EOY Return
CAO Scotland Payments EAS	_ASG_ITD	
Court Order	_ELEMENT_ITD	
Court Order Arrears Deduction	_ELEMENT_ITD	
Court Order Arrears Protected Pay	_ELEMENT_ITD	
Court Order Non Priority	_ELEMENT_ITD	
EAS Scotland	_ASG_ITD	
EAS Scotland Payments	_ASG_ITD	
GAYE	_ASG_YTD	
GAYE Taxed	_ASG_YTD	
Gross Pay	_ASG_TD_YTD	Yes
	_ASG_YTD	
NI A Able	_ASG_TD_YTD	Yes

Balance Name	Dimension	On EOY Return
	_PER_TD_DIR_YTD	
NI A Employee	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI A Employer	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI A Total	_ASG_TD_YTD	Yes
NI Arrears	_ASG_TD_YTD	
NI B Able	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI B Employee	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI B Employer	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI B Total	_ASG_TD_YTD	Yes
NI C Able	_ASG_TD_YTD	
	_PER_TD_DIR_YTD	
NI C CO	_ASG_TD_YTD	
	_PER_TD_DIR_YTD	
NI C CO Able	_ASG_TD_YTD	

Balance Name	Dimension	On EOY Return
	_PER_TD_DIR_YTD	
NI C Employee	_ASG_TD_YTD	
	_PER_TD_DIR_YTD	
NI C Employer	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI C Total	_ASG_TD_YTD	Yes
NI Car Payment	_ASG_STAT_YTD	
NI Car Payment Secondary	_ASG_STAT_YTD	
NI Car Primary	_ASG_STAT_YTD	
NI Car Secondary	_ASG_STAT_YTD	
NI D Able	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI D CO	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI D CO Able	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI D Employee	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI D Employer	_ASG_TD_YTD	Yes

Balance Name	Dimension	On EOY Return
	_PER_TD_DIR_YTD	
NI D Total	_ASG_TD_YTD	Yes
NI E Able	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI E CO	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI E CO Able	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI E Employee	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI E Employer	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
NI E Total	_ASG_TD_YTD	Yes
NI Employee Arrears	_ASG_TD_YTD	
NI Employer	_ASG_TD_YTD	
NI Employer Arrears	_ASG_TD_YTD	
NI Y	_ASG_STAT_YTD	Yes
NI Y Last Year	_ASG_STAT_YTD	Yes
NIable Pay	_ASG_TD_YTD	

Balance Name	Dimension	On EOY Return
	_PER_TD_DIR_YTD	
Net Pay	_ASG_TD_YTD	
PAYE	_ASG_TD_YTD	Yes
	_PER_TD_DIR_YTD	
PAYE Dispute Refund	_ASG_TD_ITD	
PAYE Starter Refund	_ASG_TD_YTD	
SMP Total	_ASG_TD_YTD	Yes
SSP Total	_ASG_TD_YTD	Yes
Superannuation Total	_ASG_TD_YTD	Yes
Taxable Pay	_ASG_TD_YTD	Yes
Total Deductions	_ASG_TD_YTD	
Widows and Orphans	_ASG_TD_YTD	Yes

Balance View Usage

The balance view provided, PAY_GB_BALANCES_BY_ACTION_V, enables you to query balances, their dimensions and values either by Assignment ID or by Assignment Action ID. Each row returned represents a single balance dimension and value.

Name	Null?	Type
ASSIGNMENT_ID	NOT NULL	NUMBER(9)
ASSIGNMENT_ACTION_ID	NOT NULL	NUMBER(15)
BALANCE_TYPE_ID	NOT NULL	NUMBER(9)
BALANCE_NAME	NOT NULL	VARCHAR2(80)
DEFINED_BALANCE_ID	NOT NULL	NUMBER(9)
DATABASE_ITEM_SUFFIX	NOT NULL	VARCHAR2(30)
VALUE		NUMBER

If, for example, you wanted to retrieve values for a particular balance, for an assignment action of '5268', for the balance type of 'Gross Pay', you would issue the following SQL query:

```
select balance_name, database_item_suffix, value
from pay_gb_balances_by_action_v
where assignment_action_id = 5268
and balance_name = 'Gross Pay'
```

As long as there were relevant balance values, the query might for example yield the following results:

Balance Name	Database Item Suffix	Value
Gross Pay	_ASG_PROC_PTD	4100
Gross Pay	_ASG_RUN	4100
Gross Pay	_ASG_TD_YTD	16700

Balance Name	Database Item Suffix	Value
Gross Pay	_ASG_YTD	16700
Gross Pay	_ASG_ITD	16700

You can see that the database item suffix column relates to the relevant dimensions of the particular balance.

You can use the view to find out which balances are relevant for a particular assignment. You can do this by using the following SQL query:

```
Using an example assignment ID of 662;
select distinct balance_name
from pay_gb_balances_by_action_v
where assignment_id = 662
```

This would, for example, yield the result in the following format:

```
BALANCE_NAME
____________________

Attachable
Gross Pay
NI B Able
NI B Employee
NI B Employer
NI B Total
NI D Able
NI D CO
NI D CO Able
NI D Employee
NI D Employer

BALANCE_NAME
____________________

NI D Total
NI E Able
NI E CO
NI E CO Able
NI E Employee
NI E Employer
NI E Total
NI Employer
NIable Pay
NW Earns
Net Pay
```

BALANCE_NAME

PAYE
Taxable Pay
Total Deductions
Total Pay

Note: Only balances relevant to the assignment are shown.

CHAPTER

13 Payroll Advice Report (UK Only)

Pay Advice Report

The Pay Advice Report produces the Pay Advice document on preprinted stationery. Part of the report is based on views for efficiency and to reduce complexity.

A complex dynamic sort is a feature of this report.

Parameter Values

The following are the parameters users can enter to generate the report:

p_payroll_id	Number (9)
p_time_period_id	Number (9)
p_pay_advice_date	Date
p_assignment_id	Number (20)
p_bus_grp_id	Number (15)
p_sort_order1/6	Char (60)
p_sort_order7	Char (20)

Queries

There are five queries in the report.

Q_Personel	This is the driving query based on the view PAY_ASSIGNMENT_ACTIONS_V2.
Q_Payment	This query is based on the view PAY_ELEMENT_TYPES_V1. It is linked to the main query by assignment_action_id.
Q_Deduction	This query is based on the view PAY_ELEMENTS_TYPES_V1. It is linked to the main query by assignment_action_id.
Q_Accounts	This query is based on the view PAY_EXTERNAL_ACCOUNTS_V. It is linked to the main query by assignment_action_id.
Q_Messaes	This query is based on the PAY_PAYROLL_ACTIONS table. It is linked to the main query by run_payroll_action_id.

Groups

The following are formula type fields in the Personnel group:

c_name	Concatenates the person's title, initials and last name.
c_get_address	Fetches the home or work address depending upon the value in the expense_check_send_to_address field. The address id is found from the package pay_gb_payroll_actions.get_home_address or .get_work_add.
c_ff	Gets the tax details from the package pay_gb_payroll_actions_pkg.get_report_db_items and .get_report_balances.

Triggers

The following triggers are used:

Before Report	This trigger sets the session id and date into the fnd_session table.
After Form	This trigger checks if the parameter has been left defaulted, and if so, sets it to null. This is part of dynamic sort order.

Layout

The layout is a complex grouping of the various groups and summaries. It is possible to overflow in some of the groups, thus producing extra pages. Not all the fields are printed on the extra pages (only fields such as Person Name). When you run the report in the designer, not all fields are correctly aligned. However, when you run it in the correct environment with the correct.prt file, it aligns correctly.

Dynamic Sort Order

You can dynamically change the sort order of the report. The last parameter (sort_order7) is mandatory, and consists of either the employee's last name or assignment number.

The working of the sort is as follows:

1. The sort is defaulted to Not Sort by the first six parameters, and only by the last name.
2. You can enter the required parameters, for example, Segment1, Segment2, when running the report.
3. The Post Form trigger checks whether you have entered a parameter. If not, the parameter default is lost and the parameter is set to null.
4. The main query begins and checks the send expense_check_send_to_address field. If it is set, the person's home address is used to address the pay advice, and the dynamic sort order is not used. If it is not set, the dynamic sort order is used to order the pay advices.

Warning: The design of the sort order means that you cannot change the main query without compile errors resulting. This is because the lexical parameter in the query uses the default sort order value segmentx. If you want to change the query, you first need to change segmentx to segment1, and so on. The query now complies. You then must change it back to segmentx in order for the trigger to work.

APPENDIX

A Post Install Steps

Post Install Steps

There are three post install utilities for Oracle HRMS in Release 11i:

- DataInstall allows 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.
- Installing Quantum. Quantum is a third party taxation product, produced by Vertex, that is used by Oracle Payroll (US). This step should only be performed when installing Oracle Payroll for a United States legislation.

  Note: If you are performing an upgrade you will have completed the first two steps as part of your upgrade process.

DataInstall

Step 1 Run the DataInstall Utility (Required)

To specify legislations using DataInstall:

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

   ```
   java 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 will be 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.

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 onyl available if you have no existing college data. If you have exisiting 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. Here you can select choose option 1 to make further changes, or option 2 to exit.

5. When you have chosen 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.

AutoPatch (adpatch)

Step 2 Run the Global Legislation Driver (Required)

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/admin/driver

$ adpatch
```

Then apply the driver hrglobal.drv

After applying the Global Legislation Driver

Examine the out file hrlegend.lst. This logs any localizations selected in the Java utility but 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

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.

Installing Quantum

Step 3 Install Quantum for Oracle Payroll (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.

   Suggestion: 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 directry called pay. On the ZIP file will be one directory per operating system that is supported by Oracle Payroll (US). Uncompresss 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/quantum
```

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 $PAY_TOP/vendor/quantum
VNDPAYSL='$(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');
```

Glossary

360 Degree Appraisal Part of the SSHR Appraisal function and also known as a Group Appraisal. This is an employee appraisal undertaken by managers with participation by reviewers.

360 Degree Self Appraisal Part of the SSHR Appraisal function and also known as a Group Appraisal. This is a 360 Degree appraisal initiated by an employee. The employee (initiator) can add managers and reviewers to the appraisal.

A

Absence Types Categories of absence, such as medical leave or vacation leave, that you define for use in absence windows.

Accrual Band A range of values that determines how much paid time off an employee accrues. The values may be years of service, grades, hours worked, or any other factor.

Accrual Plan See: *PTO Accrual Plan*

Accrual Period The unit of time, within an accrual term, in which PTO is accrued. In many plans, the same amount of time is accrued in each accrual period, such as two days per month. In other plans, the amount accrued varies from period to period, or the entitlement for the full accrual term is given as an up front amount at the beginning of the accrual term.

Accrual Term The period, such as one year, for which accruals are calculated. In most accrual plans, unused PTO accruals must be carried over or lost at the end of the accrual term. Other plans have a rolling accrual term which is of a certain duration but has no fixed start and end dates.

Activity Rate The monetary amount or percentage associated with an activity, such as $12.35 per pay period as an employee payroll contribution for medical coverage. Activity rates can apply to participation, eligibility, coverages, contributions, and distributions.

Actual Premium The per–participant premium an insurance carrier charges the plan sponsor for a given benefit.

Administrative Enrollment A type of scheduled enrollment caused by a change in plan terms or conditions and resulting in a re–enrollment.

Applicant A candidate for employment in a Business Group.

Appraisee A person being appraised by an appraiser..

Appraiser A person, usually a manager, who appraises an employee.

Appraisal An appraisal is a process where an employee's work performance is rated and future objectives set. See also: *Assessment*.

Appraising Manager The person who initiates and performs an Employee–Manager or 360 Degree Appraisal. An appraising manager can create appraisal objectives.

Apply for a Job An SSHR function that enables an employee to, apply, search and prepare applications for an internally advertised vacancy.

Arrestment Scottish court order made out for unpaid debts or maintenance payments. See also: *Court Order*

Assessment An information gathering exercise, from one or many sources, to evaluate a person's ability to do a job. See also: *Appraisal*.

Assignment An employee's assignment identifies his or her role and payroll within a Business Group. The assignment is made up of a number of assignment components. Of these, organization is mandatory, and payroll is a required component for payment purposes.

Assignment Number A number that uniquely identifies an employee's assignment. An employee with multiple assignments has multiple assignment numbers.

Assignment Set A grouping of employees and/or applicants that you define for running QuickPaint reports and processing payrolls. See also: *QuickPaint Report*

Assignment Status For employees, used to track their permanent or temporary departures from your enterprise, and to control the remuneration they receive. For applicants, used to track the progress of their applications.

B

BACS Banks Automated Clearing System. This is the UK system for making direct deposit payments to employees.

Balances Positive or negative accumulations of values over periods of time normally generated by payroll runs. A balance can sum pay values, time periods or numbers. See also: *Predefined Components*

Balance Adjustment A correction you make to a balance. You can adjust user balances and assignment level predefined balances only.

Balance Dimension The period for which a balance sums its balance feeds, or the set of assignments/transactions for which it sums them. There are five time dimensions: Run, Period, Quarter, Year and User. You can choose any reset point for user balances.

Balance Feeds These are the input values of matching units of measure of any elements defined to feed the balance.

Bargaining Unit A bargaining unit is a legally organized group of people which have the right to negotiate on all aspects of terms and conditions with employers or employer federations. A bargaining unit is generally a trade union or a branch of a trade union.

Base Currency The currency in which Oracle Payroll performs all payroll calculations for your Business Group. If you pay employees in different currencies to this, Oracle Payroll calculates the amounts based on exchange rates defined in the system.

Behavioral Indicators Characteristics that identify how a competence is exhibited in the work context. See also: *Proficiency Level*

Benefit Any part of an employee's remuneration package that is not pay. Vacation time, employer–paid medical insurance and stock options are all examples of benefits. See also**:** *Elements*

Block The largest subordinate unit of a window, containing information for a specific business function or entity. Every window consists of at least one block. Blocks contain fields and, optionally, regions. They are delineated by a bevelled edge. You must save your entries in one block before navigating to the next. See also: *Region*, *Field*

Budget Value In Oracle Human Resources you can enter staffing budget values and actual values for each assignment to measure variances between actual and planned staffing levels in an organization or hierarchy.

Business Group The highest level organization in the Oracle HRMS system. A Business Group may correspond to the whole of your enterprise or to a major grouping such as a subsidiary or operating division. Each Business Group must correspond to a separate implementation of Oracle HRMS.

Business Number (BN) In Canada, this is the employer's account number with Revenue Canada. Consisting of 15 digits, the first 9 identify the employer, the next 2 identify the type of tax account involved (payroll vs. corporate tax), and the last 4 identify the particular account for that tax.

C

Cafeteria Benefits Plan See: Flexible Benefits Program

Calendars In Oracle Human Resources you define calendars that determine the start and end dates for budgetary years, quarters and periods. For each calendar you select a basic period type. In Oracle SSP/SMP you define calendars to determine the start date and time for SSP qualifying patterns.

Calendar Exceptions In Oracle SSP/SMP you define calendar exceptions for an SSP qualifying pattern, to override the pattern on given days. Each calendar exception is another pattern which overrides the usual pattern.

Canada/Quebec Pension Plan (CPP/QPP) Contributions Contributions paid by employers and employees to each of these plans provide income benefits upon retirement.

Candidate Offers An SSHR function used by a line manager to offer a job to a candidate. This function is supplied with its own responsibility.

Career Path This shows a possible progression from one job or position from any number of other jobs or positions within the Business Group. A career path must be based on either job progression or position progression; you cannot mix the two.

Carry Over The amount of unused paid time off entitlement an employee brings forward from one accrual term to the next. It may be subject to an expiry date i.e. a date by which it must be used or lost. See also: *Residual*

Cash Analysis A specification of the different currency denominations required for paying your employees in cash. Union contracts may require you to follow certain cash analysis rules.

Certification Documentation required to enroll or change elections in a benefits plan as the result of a life event, to waive participation in a plan, to designate dependents for coverage, or to receive reimbursement for goods or services under an FSA.

Ceiling The maximum amount of unused paid time off an employee can have in an accrual plan. When an employee reaches this maximum, he or she must use some accrued time before any more time will accrue.

Child/Family Support payments In Canada, these are payments withheld from an employee's compensation to satisfy a child or family support order from a Provincial Court. The employer is responsible for withholding and remitting the payments to the court named in the order.

Collective Agreement A collective agreement is a form of contract between an employer or employer representative, for example, an employer federation, and a bargaining unit for example, a union or a union branch.

Communications Benefits plan information that is presented in some form to participants. Examples include a pre–enrollment package, an enrollment confirmation statement, or a notice of default enrollment.

Compensation The pay you give to employees, including wages or salary, and bonuses. See also: *Elements*

Competence Any measurable behavior required by an organization, job or position that a person may demonstrate in the work context. A competence can be a piece of knowledge, a skill, an attitude or an attribute.

Competence Evaluation A method used to measure an employees ability to do a defined job.

Competence Profile Where you record applicant and employee accomplishments, for example, proficiency in a competence.

Competence Requirements Competencies required by an organization, job or position. See also: *Competence*, *Core Competencies*

Competence Type A group of related competencies.

Consolidation Set A grouping of payroll runs within the same time period for which you can schedule reporting, costing, and post–run processing.

Contact A person who has a relationship to an employee that you want to record. Contacts can be dependents, relatives, partners or persons to contact in an emergency.

Contract A contract of employment is an agreement between an employer and employee or potential employee that defines the fundamental legal relationship between an employing organization and a person who offers his or her services for hire. The employment contract defines the terms and conditions to which both parties agree and those that are covered by local laws.

Contribution An employer's or employee's monetary or other contribution to a benefits plan.

Core Competencies Also known as *Leadership Competencies* or *Management Competencies*. The competencies required by every person to enable the enterprise to meet its goals. See also: *Competence*

Costable Type A feature that determines the processing an element receives for accounting and costing purposes. There are four costable types in Oracle HRMS: costed, distributed costing, fixed costing, and not costed.

Costing Recording the costs of an assignment for accounting or reporting purposes. Using Oracle Payroll, you can calculate and transfer costing information to your general ledger and into systems for project management or labor distribution.

Court Order A ruling from a court that requires an employer to make deductions from an employee's salary for maintenance payments or debts, and to pay the sums deducted to a court or local authority. See also: *Arrestment*

Cross Business Group Responsibility Security This security model uses security groups and enables you to link one responsibility to many Business Groups.

Customizable Forms Forms that your system administrator can modify for ease of use or security purposes by means of Custom Form restrictions. The Form Customization window lists the forms and their methods of customization.

D

Database Item An item of information in Oracle HRMS that has special programming attached, enabling Oracle FastFormula to locate and retrieve it for use in formulas.

Date To and Date From These fields are used in windows not subject to DateTrack. The period you enter in these fields remains fixed until you change the values in either field. See also: *DateTrack*, *Effective Date*

DateTrack When you change your effective date (either to past or future), DateTrack enables you to enter information that takes effect on your new effective date, and to review information as of the new date. See also: *Effective Date*

Deployment Factors See: *Work Choices*

Derived Factor A factor (such as age, percent of fulltime employment, length of service, compensation level, or the number of hours worked per period) that is used in calculations to determine Participation Eligibility or Activity Rates for one or more benefits.

Descriptive Flexfield A field that your organization can customize to capture additional information required by your business but not otherwise tracked by Oracle Applications. See also: *Key Flexfield*

Developer Descriptive Flexfield A flexfield defined by your localization team to meet the specific legislative and reporting needs of your country. See also: *Extra Information Types*

Direct Deposit The electronic transfer of an employee's net pay directly into the account(s) designated by the employee.

Distribution Monetary payments made from, or hours off from work as allowed by, a compensation or benefits plan.

E

Effective Date The date for which you are entering and viewing information. You set your effective date in the Alter Effective Date window. See also: *DateTrack*

EIT See: *Extra Information Type*

Elements Components in the calculation of employee pay. Each element represents a compensation or benefit type, such as salary, wages, stock purchase plans, and pension contributions.

Element Classifications These control the order in which elements are processed and the balances they feed. Primary element classifications and some secondary classifications are predefined by Oracle Payroll. Other secondary classifications can be created by users.

Element Entry The record controlling an employee's receipt of an element, including the period of time for which the employee receives the element and its value. See also: *Recurring Elements*, *Nonrecurring Elements*

Element Link The association of an element to one or more components of an employee assignment. The link establishes employee eligibility for that element. Employees whose assignment components match the components of the link are eligible for the element. See also: *Standard Link*

Element Set A group of elements that you define to process in a payroll run, or to control access to compensation information from a customized form, or for distributing costs.

Employee Histories An SSHR function for an employee to view their, Training History, Job Application History, Employment History, Absence History, or Salary History. A manager can also use this function to view information on their direct reports.

Employment Category A component of the employee assignment. Four categories are defined: Full Time – Regular, Full Time – Temporary, Part Time – Regular, and Part Time – Temporary.

Employment Insurance (EI) Benefit plan run by the federal government to which the majority of Canadian employers and employees must contribute.

Employment Insurance Rate In Canada, this is the rate at which the employer contributes to the EI fund. The rate is expressed as a percentage of the employee's contribution. If the employer maintains an approved wage loss replacement program, they can reduce their share of EI premiums by obtaining a reduced contribution rate. Employers would remit payroll deductions under a different employer account number for employees covered by the plan.

Employment Equity Occupational Groups (EEOG) In Canada, the Employment Equity Occupational Groups (EEOG) consist of 14 classifications of work used in the Employment Equity Report. The EEOGs were derived from the National Occupational Classification system.

Enroll in a Class An SSHR function which enables an employee to search and enroll in an internally published class. An employee can also use this function to maintain their competencies.

Enrollment Action Type Any action required to complete enrollment or de–enrollment in a benefit.

ESS Employee Self Service. A predefined SSHR responsibility.

Event An activity such as a training day, review, or meeting, for employees or applicants.

Expected Week of Confinement (EWC) In the UK, this is the week in which an employee's baby is due. The Sunday of the expected week of confinement is used in the calculations for Statutory Maternity Pay (SMP).

Extra Information Type (EIT) A type of developer descriptive flexfield that enables you to create an unlimited number of information types for six key areas in Oracle HRMS. Localization teams may also predefine some EITs to meet the specific legislative requirements of your country. See also: *Developer Descriptive Flexfield*

F

Field A view or entry area in a window where you enter, view, update, or delete information. See also: *Block*, *Region*

Flex Credit A unit of "purchasing power" in a flexible benefits program. An employee uses flex credits, typically expressed in monetary terms, to "purchase" benefits plans and/or levels of coverage within these plans.

Flexible Benefits Program A benefits program that offers employees choices among benefits plans and/or levels of coverage. Typically, employees are given a certain amount of flex credits or moneys with which to "purchase" these benefits plans and/or coverage levels.

Flexible Spending Account (FSA) Under US Internal Revenue Code Section 125, employees can set aside money on a pretax basis to pay for eligible unreimbursed health and dependent care expenses. Annual monetary limits and use–it–or–lose it provisions exist. Accounts are subject to annual maximums and forfeiture rules.

Form A predefined grouping of functions, called from a menu and displayed, if necessary, on several windows. Forms have blocks, regions and fields as their components. See also: *Block*, *Region*, *Field*

G

Global Value A value you define for any formula to use. Global values can be dates, numbers or text.

Goods or Service Type A list of goods or services a benefit plan sponsor has approved for reimbursement.

Grade A component of an employee's assignment that defines their level and can be used to control the value of their salary and other compensation elements.

Grade Comparatio A comparison of the amount of compensation an employee receives with the mid–point of the valid values defined for his or her grade.

Grade Rate A value or range of values defined as valid for a given grade. Used for validating employee compensation entries.

Grade Scale A sequence of steps valid for a grade, where each step corresponds to one point on a pay scale. You can place each employee on a point of their grade scale and automatically increment all placements each year, or as required. See also: *Pay Scale*

Grade Step An increment on a grade scale. Each grade step corresponds to one point on a pay scale. See also: *Grade Scale*

Grandfathered A term used in Benefits Administration. A person's benefits are said to be grandfathered when a plan changes but they retain the benefits accrued.

Group A component that you define, using the People Group key flexfield, to assign employees to special groups such as pension plans or unions. You can use groups to determine employees' eligibility for certain elements, and to regulate access to payrolls.

H

Hierarchy An organization or position structure showing reporting lines or other relationships. You can use hierarchies for reporting and for controlling access to Oracle HRMS information.

I

Imputed Income Certain forms of indirect compensation that US Internal Revenue Service Section 79 defines as fringe benefits and taxes the recipient accordingly. Examples include employer payment of group term life insurance premiums over a certain monetary amount, personal use of a company car, and other non–cash awards.

Initiator In SSHR a person who starts a 360 Degree appraisal (Employee or Self) on an individual. An initiator and the appraisee are the only people who can see all appraisal information.

Input Values Values you define to hold information about elements. In Oracle Payroll, input values are processed by formulas to calculate the element's run result. You can define up to fifteen input values for an element.

Instructions An SSHR user assistance component displayed on a web page to describe page functionality.

K

Key Flexfield A flexible data field made up of segments. Each segment has a name you define and a set of valid values you specify. Used as the key to uniquely identify an entity, such as jobs, positions, grades, cost codes, and employee groups. See also: *Descriptive Flexfield*

L

Leaver's Statement In the UK, this Records details of Statutory Sick Pay (SSP) paid during a previous employment (issued as form SSP1L) which is used to calculate a new employee's entitlement to SSP. If a new employee falls sick, and the last date that SSP was paid for under the previous employment is less than eight calendar weeks before the first day of the PIW for the current sickness, the maximum liability for SSP is reduced by the number of weeks of SSP shown on the statement.

Life Event A significant change in a person's life that results in a change in eligibility or ineligibility for a benefit.

Life Event Collision A situation in which the impacts from multiple life events on participation eligibility, enrollability, level of coverage or activity rates conflict with each other.

Life Event Enrollment A benefits plan enrollment that is prompted by a life event occurring at any time during the plan year.

Linking Interval In the UK, this is the number of days that separate two periods of incapacity for work. If a period of incapacity for work (PIW) is separated from a previous PIW by less than the linking interval, they are treated as one PIW according to the legislation for entitlement to Statutory Sick Pay (SSP). An employee can only receive SSP for the maximum number of weeks defined in the legislation for one PIW.

Linked PIWs In the UK, these are linked periods of incapacity for work that are treated as one to calculate an employee's entitlement to Statutory Sick Pay (SSP). A period of incapacity for work (PIW) links to an earlier PIW if it is separated by less than the linking interval. A linked PIW can be up to three years long.

LMSS Line Manager Self Service. A predefined SSHR responsibility.

Lookup Types Categories of information, such as nationality, address type and tax type, that have a limited list of valid values. You can define your own Lookup Types, and you can add values to some predefined Lookup Types.

Lower Earnings Limit (LEL) In the UK, this is the minimum average weekly amount an employee must earn to pay National Insurance contributions. Employees who do not earn enough to pay National Insurance cannot receive Statutory Sick Pay (SSP) or Statutory Maternity Pay (SMP).

M

Manager–Employee Appraisal Part of the SSHR Appraisal function. A manager appraisal of an employee. However, an appraising manager does not have to be a manager.

Maternity Pay Period In the UK, this is the period for which Statutory Maternity Pay (SMP) is paid. It may start at any time from the start of the 11th week before the expected week of confinement and can continue for up to 18 weeks. The start date is usually agreed with the employee, but can start at any time up to the birth. An employee is not eligible to SMP for any week in which she works or for any other reason for ineligibility, defined by the legislation for SMP.

Menus You set up your own navigation menus, to suit the needs of different users.

N

NACHA National Automated Clearing House Association. This is the US system for making direct deposit payments to employees.

Net Accrual Calculation The rule that defines which element entries add to or subtract from a plan's accrual amount to give net entitlement.

Net Entitlement The amount of unused paid time off an employee has available in an accrual plan at any given point in time.

Nonrecurring Elements Elements that process for one payroll period only unless you make a new entry for an employee. See also: *Recurring Elements*

North American Industrial Classification (NAIC) code The North American Industrial Classification system (NAICs) was developed jointly by the US, Canada and Mexico to provide comparability in statistics regarding business activity across North America. The NAIC replaces the US Standard Industrial Classification (SIC) system, and is used in the Employment Equity Report.

National Occupational Classification (NOC) code In Canada, the National Occupational Classification (NOC) System was developed to best reflect the type of work performed by employees. Occupations are grouped in terms of particular tasks, duties and responsibilities. The use of this standardized system ensures consistency of data from year to year within the same company as well as between companies. These codes are used in the Employment Equity Report.

Not in Program Plan A benefit plan that you define outside of a program.

O

Open Enrollment A type of scheduled enrollment in which participants can enroll in or alter elections in one or more benefits plans.

Oracle FastFormula An Oracle tool that allows you to write Oracle HRMS formulas without using a programming language.

Organization A required component of employee assignments. You can define as many organizations as you want within your Business Group. Organizations can be internal, such as departments, or external, such as recruitment agencies. You can structure your organizations into organizational hierarchies for reporting purposes and for system access control.

OSSWA Oracle Self Service Web Applications.

OTM Oracle Training Management.

P

Pattern A pattern comprises a sequence of time units that are repeated at a specified frequency. Oracle SSP/SMP uses SSP qualifying patterns to determine employees entitlement to Statutory Sick Pay (SSP).

Pattern Time Units A sequence of time units specifies a repeating pattern. Each time unit specifies a time period of hours, days or weeks.

Pay Scale A set of progression points that can be related to one or more rates of pay. Employee's are placed on a particular point on the scale according to their grade and, usually, work experience. See also: *Grade Scale*

Payment Type There are three standard payment types for paying employees: check, cash and direct deposit. You can define your own payment methods corresponding to these types.

Payroll A group of employees that Oracle Payroll processes together with the same processing frequency, for example, weekly, monthly or bimonthly. Within a Business Group, you can set up as many payrolls as you need.

People List An SSHR line manager utility used to locate an employee.

Performance (within Assessment) An expectation of "normal" performance of a competence over a given period. For example, a person may exceed performance expectation in the communication competence. See also: *Proficiency (within Assessment)*, *Competence*, *Assessment*

Period of Incapacity for Work (PIW) In the UK, this is a period of sickness that lasts four or more days in a row, and is the minimum amount of sickness for which Statutory Sick Pay can be paid. If a PIW is separated by less then the linking interval, a linked PIW is formed and the two PIWs are treated as one.

Period Type A time division in a budgetary calendar, such as week, month, or quarter.

Person Search An SSHR function which enables a manager to search for a person. There are two types of search, Simple and Advanced.

Person Type There are eight system person types in Oracle HRMS. Seven of these are combinations of employees, ex–employees, applicants, and ex–applicants. The eighth category is 'External'. You can create your own user person types based on the eight system types.

Personal Tax Credits Return (TD1) A Revenue Canada form which each employee must complete. Used by the employee to reduce his or her taxable income at source by claiming eligible credits and also provides payroll with such important information as current address, birth date, and SIN. These credits determine the amount to withhold from the employee's wages for federal/provincial taxes.

Plan Design The functional area that allows you to set up your benefits programs and plans. This process involves defining the rules which govern eligibility, available options, pricing, plan years, third party administrators, tax impacts, plan assets, distribution options, required reporting, and communications.

Plan Sponsor The legal entity or business responsible for funding and administering a benefits plan. Generally synonymous with employer.

Position A specific role within the Business Group derived from an organization and a job. For example, you may have a position of Shipping Clerk associated with the organization Shipping and the job Clerk.

Predefined Components Some elements and balances, all primary element classifications and some secondary classifications are defined by Oracle Payroll to meet legislative requirements, and are supplied to users with the product. You cannot delete these predefined components.

Professional Information An SSHR function which allows an employee to maintain their own professional details or a line manager to maintain their direct reports professional details.

Proficiency (within Assessment) The perceived level of expertise of a person in a competence, in the opinion of the assessor, over a given period. For example, a person may demonstrate the communication competence at Expert level. See also: *Performance (within Assessment)*, *Competence*, *Assessment*

Proficiency Level A system for expressing and measuring how a competence is exhibited in the work context. See also: *Behavioral Indicators*.

Progression Point A pay scale is calibrated in progression points, which form a sequence for the progression of employees up the pay scale. See also: *Pay Scale*

Provincial/Territorial Employment Standards Acts In Canada, these are laws covering minimum wages, hours of work, overtime, child labour, maternity, vacation, public/general holidays, parental and adoption leave, etc., for employees regulated by provincial/territorial legislation.

Provincial Health Number In Canada, this is the account number of the provincially administered health care plan that the employer would use to make remittances. There would be a unique number for each of the provincially controlled plans i.e. EHT, Quebec HSF, etc.

PTO Accrual Plan A benefit in which employees enroll to entitle them to accrue and take paid time off. The purpose of absences allowed under the plan, who can enroll, how much time accrues, when the time must be used, and other rules are defined for the plan.

Q

QPP (See Canada/Quebec Pension Plan)

Qualification Type An identified qualification method of achieving proficiency in a competence, such as an award, educational qualification, a license or a test. See also: *Competence*

Qualifying Days In the UK, these are days on which Statutory Sick Pay (SSP) can be paid, and the only days that count as waiting days. Qualifying days are normally work days, but other days may be agreed.

Qualifying Pattern See: *SSP Qualifying Pattern*

Qualifying Week In the UK, this is the week during pregnancy that is used as the basis for the qualifying rules for Statutory Maternity Pay (SMP). The date of the qualifying week is fifteen weeks before the expected week of confinement and an employee must have been continuously employed for at least 26 weeks continuing into the qualifying week to be entitled to SMP.

Quebec Business Number In Canada, this is the employer's account number with the Ministere du Revenu du Quebec, also known as the Quebec Identification number. It consists of 15 digits, the first 9 identify the employer, the next 2 identify the type of tax account involved (payroll vs. corporate tax), and the last 4 identify the particular account for that tax.

Questionnaire An SSHR function which records the results of an appraisal.

QuickPaint Report A method of reporting on employee and applicant assignment information. You can select items of information, paint them on a report layout, add explanatory text, and save the report definition to run whenever you want. See also: *Assignment Set*

R

Rates A set of values for employee grades or progression points. For example, you can define salary rates and overtime rates.

Rating Scale Used to describe an enterprise's competencies in a general way. You do not hold the proficiency level at the competence level. See also: *Proficiency Level*

Record of Employment (ROE) A Human Resources Development Canada form that must be completed by an employer whenever an interruption of earnings occurs for any employee. This form is necessary to claim Employment Insurance benefits.

Recruitment Activity An event or program to attract applications for employment. Newspaper advertisements, career fairs and recruitment evenings are all examples of recruitment activities. You can group several recruitment activities together within an overall activity.

Recurring Elements Elements that process regularly at a predefined frequency. Recurring element entries exist from the time you create them until you delete them, or the employee ceases to be eligible for the element. Recurring elements can have standard links. See also: *Nonrecurring Elements*, *Standard Link*

Region A collection of logically related fields in a window, set apart from other fields by a rectangular box or a horizontal line across the window. See also: *Block*, *Field*

Registered Pension Plan (RPP) This is a pension plan that has been registered with Revenue Canada. It is a plan where funds are set aside by an employer, an employee, or both to provide a pension to employees when they retire. Employee contributions are generally exempt from tax.

Registered Retirement Savings Plan (RRSP) This is an individual retirement savings plan that has been registered with Revenue Canada. Usually, contributions to the RRSP, and any income earned within the RRSP, is exempt from tax.

Report Parameters Inputs you make when submitting a report to control the sorting, formatting, selection, and summarizing of information in the report.

Report Set A group of reports and concurrent processes that you specify to run together.

Requisition The statement of a requirement for a vacancy or group of vacancies.

Request Groups A list of reports and processes that can be submitted by holders of a particular responsibility. See also: *Responsibility*

Residual The amount of unused paid time off entitlement an employee loses at the end of an accrual term. Typically employees can carry over unused time, up to a maximum, but they lose any residual time that exceeds this limit. See also: *Carry Over*

Responsibility A level of authority in an application. Each responsibility lets you access a specific set of Oracle Applications forms, menus, reports, and data to fulfill your business role. Several users can share a responsibility, and a single user can have multiple responsibilities. See also: *Security Profile*, *User Profile Options*, *Request Groups*, *Security Groups*

Retry Method of correcting a payroll run or other process *before* any post–run processing takes place. The original run results are deleted and the process is run again.

Revenue Canada Department of the Government of Canada which, amongst other responsibilities, administers, adjudicates, and receives remittances for all taxation in Canada including income tax, Employment Insurance premiums, Canada Pension Plan contributions, and the Goods and Services Tax (legislation is currently proposed to revise the name to the Canada Customs and Revenue Agency). In the province of Quebec the equivalent is the Ministere du Revenu du Quebec.

Reviewer (SSHR) A person invited by an appraising manager to add review comments to an appraisal.

Reversal Method of correcting payroll runs or QuickPay runs *after* post–run processing has taken place. The system replaces positive run result values with negative ones, and negative run result values with positive ones. Both old and new values remain on the database.

Rollback Method of removing a payroll run or other process *before* any post–run processing takes place. All assignments and run results are deleted.

S

Search by Date An SSHR sub–function used to search for a Person by Hire date, Application date, Job posting date or search by a Training event date.

Salary Basis The period of time for which an employee's salary is quoted, such as hourly or annually. Defines a group of employees assigned to the same salary basis and receiving the same salary element.

Scheduled Enrollment A benefits plan enrollment that takes place during a predefined enrollment period, such as an open enrollment. Scheduled enrollments can be administrative, open, or unrestricted.

Security Group Security groups enable HRMS users to partition data by Business Group. Only used for Cross Business Group Responsibility security. See also: *Responsibility*, *Security Profile*, *User Profile Options*

Security Profile Security profiles control access to organizations, positions and employee and applicant records within the Business Group. System administrators use them in defining users' responsibilities. See also: *Responsibility*

Self Appraisal Part of the SSHR Appraisal function. This is an appraisal undertaken by an employee to rate their own performance and competencies.

SMP See: *Statutory Maternity Pay*

Social Insurance Number (SIN) A unique number provided by Human Resources Development Canada (HRDC) to each person commencing employment in Canada. The number consists of 9 digits in the following format (###–###–###).

Source Deductions Return (TP 1015.3) A Ministere du Revenu du Quebec form which each employee must complete. This form is used by the employee to reduce his or her taxable income at source by claiming eligible credits and also provides payroll with such important information as current address, birth date, and SIN. These credits determine the amount of provincial tax to withhold from the employee's wages.

Special Information Types Categories of personal information, such as skills, that you define in the Personal Analysis key flexfield.

SSHR Oracle Self–Service Human Resources. An HR management system using an intranet and web browser to deliver functionality to employees and their managers.

SSP See: *Statutory Sick Pay*

SSP Qualifying Pattern In the UK, an SSP qualifying pattern is a series of qualifying days that may be repeated weekly, monthly or some other frequency. Each week in a pattern must include at least one qualifying day. Qualifying days are the only days for which Statutory Sick Pay (SSP) can be paid, and you define SSP qualifying patterns for all the employees in your organization so that their entitlement to SSP can be calculated.

Standard Link Recurring elements with standard links have their element entries automatically created for all employees whose assignment components match the link. See also: *Element Link*, *Recurring Elements*

Statement of Commissions and Expenses for Source Deduction Purposes (TP 1015.R.13.1) A Ministere du Revenu du Quebec form which allows an employee who is paid partly or entirely by commissions to pay a constant percentage of income tax based on his or her estimated commissions for the year, less allowable business expenses.

Statement of Remuneration and Expenses (TD1X) In Canada, the Statement of Remuneration and Expenses allows an employee who is paid partly or entirely by commission to pay a constant percentage of income tax, based on his or her estimated income for the year, less business-related expenses.

Statutory Maternity Pay In the UK, you pay Statutory Maternity Pay (SMP) to female employees who take time off work to have a baby, providing they meet the statutory requirements set out in the legislation for SMP.

Standard HRMS Security The standard security model. Using this security model you must log on as a different user to see a different Business Group. See: *Multiple Responsibility Security*

Statutory Sick Pay In the UK, you pay Statutory Sick Pay (SSP) to employees who are off work for four or more days because they are sick, providing they meet the statutory requirements set out in the legislation for SSP.

Succession Planning An SSHR function which enables a manager to prepare a succession plan.

Suitability Matching An SSHR function which enables a manager to compare and rank a persons competencies.

T

Tabbed Regions Parts of a window that appear in a stack so that only one is visible at any time. You click on the tab of the required region to bring it to the top of the stack.

Task Flows A sequence of windows linked by buttons to take you through the steps required to complete a task, such as hiring a new recruit. System administrators can create task flows to meet the needs of groups of users.

Terminating Employees You terminate an employee when he or she leaves your organization. Information about the employee remains on the system but all current assignments are ended.

Termination Rule Specifies when entries of an element should close down for an employee who leaves your enterprise. You can define that entries end on the employee's actual termination date or remain open until a final processing date.

Tips An SSHR user assistance component that provides information about a field.

U

User Assistance Components SSHR online help comprising tips and instructions.

User Balances Users can create, update and delete their own balances, including dimensions and balance feeds. See also: *Balances*

User Profile Options Features that allow system administrators and users to tailor Oracle HRMS to their exact requirements. See also: *Responsibility, Security Profile*

V

Viewer (SSHR) A person with view only access to an appraisal. An appraising manager or an employee in a 360 Degree Self appraisal can appoint view only access to an appraisal.

W

WCB Account Number In Canada, this is the account number of the provincially administered Worker's Compensation Board that the employer would use to make remittances. There would be a unique number for each of the provincially controlled boards i.e. Workplace Safety & Insurance Board of Ontario, CSST, etc.

Waiting Days In the UK, statutory Sick Pay is not payable for the first three qualifying days in period of incapacity for work (PIW), which are called waiting days. They are not necessarily the same as the first three days of sickness, as waiting days can be carried forward from a previous PIW if the linking interval between the two PIWs is less than 56 days.

Work Choices Also known as Work Preferences, Deployment Factors, or Work Factors. These can affect a person's capacity to be deployed within an enterprise, such willingness to travel or relocate. You can hold work choices at both job and position level, or at person level.

Worker's Compensation Board In Canada, this is a provincially governed legislative body which provides benefits to employees upon injury, disability, or death while performing the duties of the employer. Worker's Compensation Board premiums are paid entirely by the employer.

Workflow An Oracle application which uses charts to manage approval processes and in addition is used in SSHR to configure display values of sections within a web page and instructions.

Work Structures The fundamental definitions of organizations, jobs, positions, grades, payrolls and other employee groups within your enterprise that provide the framework for defining the work assignments of your employees.

Index

D

E

F

G

I

J

K

L

M

O

P

Q

R

S

T

U

V

Reader's Comment Form

Implementing Oracle HRMS
A73313–01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information we use 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? What did you like least about it?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

_____

_____

_____

_____

_____

_____

_____

_____

Please send your comments to:

Oracle Applications Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
Phone: (650) 506–7000 Fax: (650) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

_____

_____

_____

Thank you for helping us improve our documentation.