Oracle HRMS™ US Implementation Guide

Release 11.0

Part No. A58332–01

ORACLE®

Enabling the Information Age™

Oracle HRMS US Implementation Guide, Release 11.0

Part No. A58332–01

Major Contributors: Louise Raffo and Robert Rose

Contributors: Peter Attwood, John Cafolla, Juliette Fleming, Michael O'Shea and John Thuringer

This software was not developed for any use in nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It is the customer's responsibility to take all appropriate measures to insure the safe use of such applications if the programs are used for such purposes.

Contents

Preface

Welcome to Release 11.0 of the *Oracle HRMS US Implementation Guide.*

This guide contains a summary of the steps you should follow to implement all of the functionality available in Oracle HRMS. The detailed information you need to support your implementation decisions is contained in your product user's guide

This preface explains how this guide is organized and introduces other sources of information that can help you.

About This User's Guide

There are two chapters and three appendixes in this guide:

- Chapter 1 is designed to help you plan your implementation. It contains flowcharts showing the major stages of an implementation and a summary checklist for each stage, listing the implementation steps.
- Chapter 2 contains a step–by–step implementation sequence summarizing the decisions and tasks required for each stage.
- Appendix A includes essays that address important technical issues relating to your HRMS implementation.
- Appendix B includes an essay on several topics specifically relating to your Payroll implementation.
- Appendix C describes any post install steps you must perform before you implement Oracle HRMS.

The HRMS user's guides are available online

All Oracle Applications user's guides are available online, in both HTML and Adobe Acrobat format. Most other Oracle Applications documentation is available in Adobe Acrobat format.

The HTML version of the HRMS user's guides are optimized for onscreen reading, and enable you to follow hypertext links for easy access to guides across our entire library. You can also search for words and phrases if your national language is supported by Oracle's Information Navigator. The HTML documentation is available from the Oracle Applications toolbar, or from a URL provided by your system administrator. Note that the HTML documentation is translated into over twenty languages.

You can order an Oracle Applications Documentation Library CD containing Adobe Acrobat versions of each guide in the Oracle Applications documentation set. Using this CD, you can search for information, read it onscreen, and print individual pages, sections, or entire books. When you print from Adobe Acrobat, the resulting printouts look just like pages from an Oracle Applications hardcopy guide.

Finding the Latest Information

For information about any new features that were not available when this guide was printed, look at the What's New? section on the main help menu. This information is updated for each new release of Oracle HRMS HTML help.

Assumptions

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 Education.
- the Oracle Applications graphical user interface

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

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

Do Not Use Database Tools to Modify Oracle Applications Data

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 like 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 may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications 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.

Consequently, 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.

Finding Your Way Around the System

Oracle HRMS is delivered with a set of default menus and task flows that give you access to all windows on the system. A full listing of the default menus is provided in appendix A of your product user's guide. Read the *Customizing Windows and Menus* chapter of that guide to find out how to set up your own menus and task flows for users at your site.

To read more about the interface and standard features of Oracle HRMS, refer to your product user's guide.

Other Information Sources

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

Most Oracle Applications documentation is available in Adobe Acrobat format on the *Oracle Applications Documentation Library* CD. We supply this CD with every software shipment.

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

Oracle Applications User's Guide

This guide explains how to navigate, enter data, query, run reports, and introduces other basic features of the graphical user interface (GUI)

available with this release of Oracle HRMS (and any other Oracle Applications product). This guide also includes information on setting user profiles, as well as running and reviewing reports and concurrent requests.

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

Related User's Guides

Oracle HRMS shares business and setup information with other Oracle Applications products. Even if you have not installed them as separate products, your Oracle HRMS application includes some forms and functionality from other Oracle Applications. Therefore, you may want to refer to other user's guides when you set up and use Oracle HRMS.

If you do not have the hardcopy versions of these guides, you can read them by choosing Library from the Help menu, or by reading from the Oracle Applications Document Library CD, or by using a web browser with a URL that your system administrator provides.

Oracle Human Resources US User's Guide

This guide contains the information you need to set up Oracle Human Resources to meet the requirements of your enterprise. It describes how you can represent your enterprise structures, policies, and people on the system and use this information to manage your human resources.

It should be read by anyone involved in setting up or managing Oracle Human Resources.

Oracle Payroll US User's Guide

This guide contains the information you need to set up Oracle Payroll to meet the requirements of your enterprise. It describes how to use the earnings, deductions, and tax calculations that come with the system, how to initiate additional earnings and deductions in accordance with your own compensation and benefits policies, and how to manage payroll runs and post–run processing.

Oracle Applications Flexfields Guide

This guide provides flexfields planning, setup, and reference information for the Oracle HRMS implementation team, as well as for

users responsible for the ongoing maintenance of Oracle Applications product data. This guide also provides information on creating custom reports on flexfields data.

Oracle 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.

Oracle Alert User's Guide

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

Oracle Applications Implementation Wizard User's 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 Developer / 2000 forms so that they integrate with Oracle Applications.

Oracle Applications User Interface Standards

This manual 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 4.5.

Installation and System Administration

Oracle Applications Installation Manual

This manual and the accompanying release notes provide information you need to successfully install Oracle Financials, Oracle Government Financials, Oracle Manufacturing or Oracle Training Administration in your specific hardware and operating system software environment.

Oracle Applications Upgrade Manual

This manual explains how to prepare your Oracle Applications products for an upgrade. It also contains information on finishing the upgrade procedure for each product. Refer to this manual and the *Oracle Applications Installation Manual* when you plan to upgrade your products.

Oracle Applications Product Update Notes

These notes describe the new features added to Oracle HRMS, and also describe the changes made to database objects, seed data and profile options for the same 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 Technical Reference Manual

The *Oracle HRMS Technical Reference Manual* contains database diagrams and a detailed description of Oracle HRMS and related applications database tables, forms, reports, and programs. 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 manual for any product you have licensed. Technical reference manuals are available in paper format only.

Other Information

Training

Oracle Education offers 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. 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.

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, an integrated suite of more than 45 software modules for financial management, 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 140 countries around the world.

Thank You

Thank you for using Oracle HRMS and this user's guide.

We value your comments and feedback. At the end of this guide is a Reader's Comment Form you can use to explain what you like or dislike about Oracle HRMS or this user's 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.

CHAPTER

1

Planning Your Implementation

This chapter contains flowcharts and check lists to help you plan and manage your implementation of Oracle Human Resources and Oracle Payroll.

Each flowchart and check list corresponds to the implementation of one area of the system. Some of these areas are required; others are optional. You can use this information to plan a phased implementation.

Implementing Oracle HRMS

Implementation takes place after a successful installation and post installation of the system, and describes the process you follow to customize Oracle HRMS to meet your own specific business needs.

Warning: 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 must confirm the startup data is installed before you use Elements, Payment Methods or Legislation Specific Flexfield Structures.

For more information on installing Oracle HRMS, consult your *Oracle Applications Installation Manual* and *Post Install Steps*: page C – 2.

Following a Plan

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.

The flexibility of Oracle HRMS lets you develop an implementation project plan which meets your own specific business needs for both Oracle Human Resources and Oracle Payroll.

Attention: 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.

Oracle Applications Implementation Wizard

You may want to use the Oracle Applications Implementation Wizard to manage 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. The Wizard also identifies steps you can complete in parallel to help you manage your implementation process most efficiently. For Oracle HRMS, the Wizard follows the steps documented in the following Implementation Steps topic.

The Implementation Wizard also gives you a graphical overview of the setup steps and lets you keep a record of comments with each step for future reference and review.

Implementation Flowcharts

The flowcharts provide you with a summary of the logical sequence you might follow in any implementation of all the functional areas of Oracle Human Resources and Oracle Payroll. It is not the only sequence you could follow but this one takes account of all the dependencies which exist in the system.

The functional areas of the system that you implement depend on your own specific business needs. Steps that are required for all implementations are marked as *required*.

Some functions have been *seeded* with default data. The steps where you can use data supplied with the system are marked as *seeded*.

After you complete your implementation, you are ready to enter your personal, assignment and pay and benefits information into your system.

Figure 1 – 1
Legend for Following Setup Diagrams

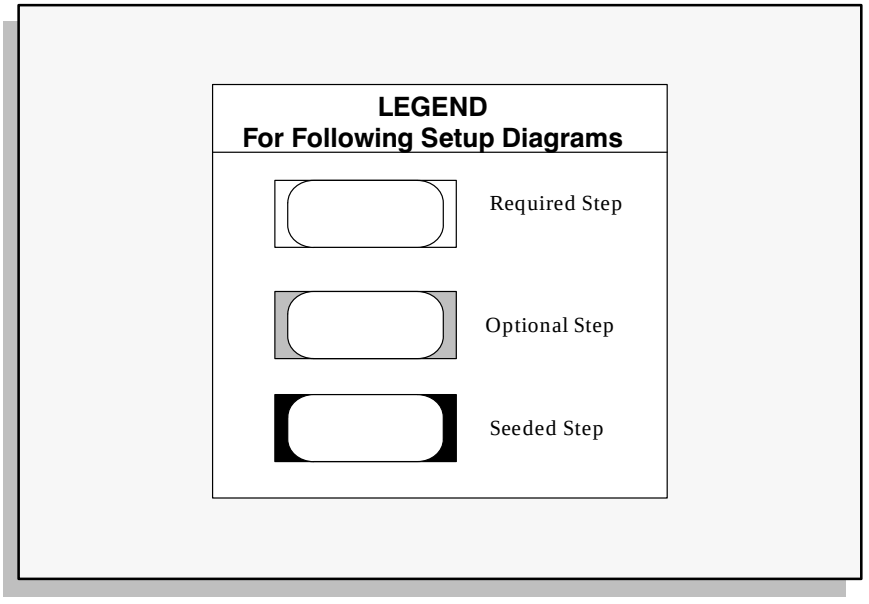


Figure 1 – 2
Implementation Flowchart for Basic Administration

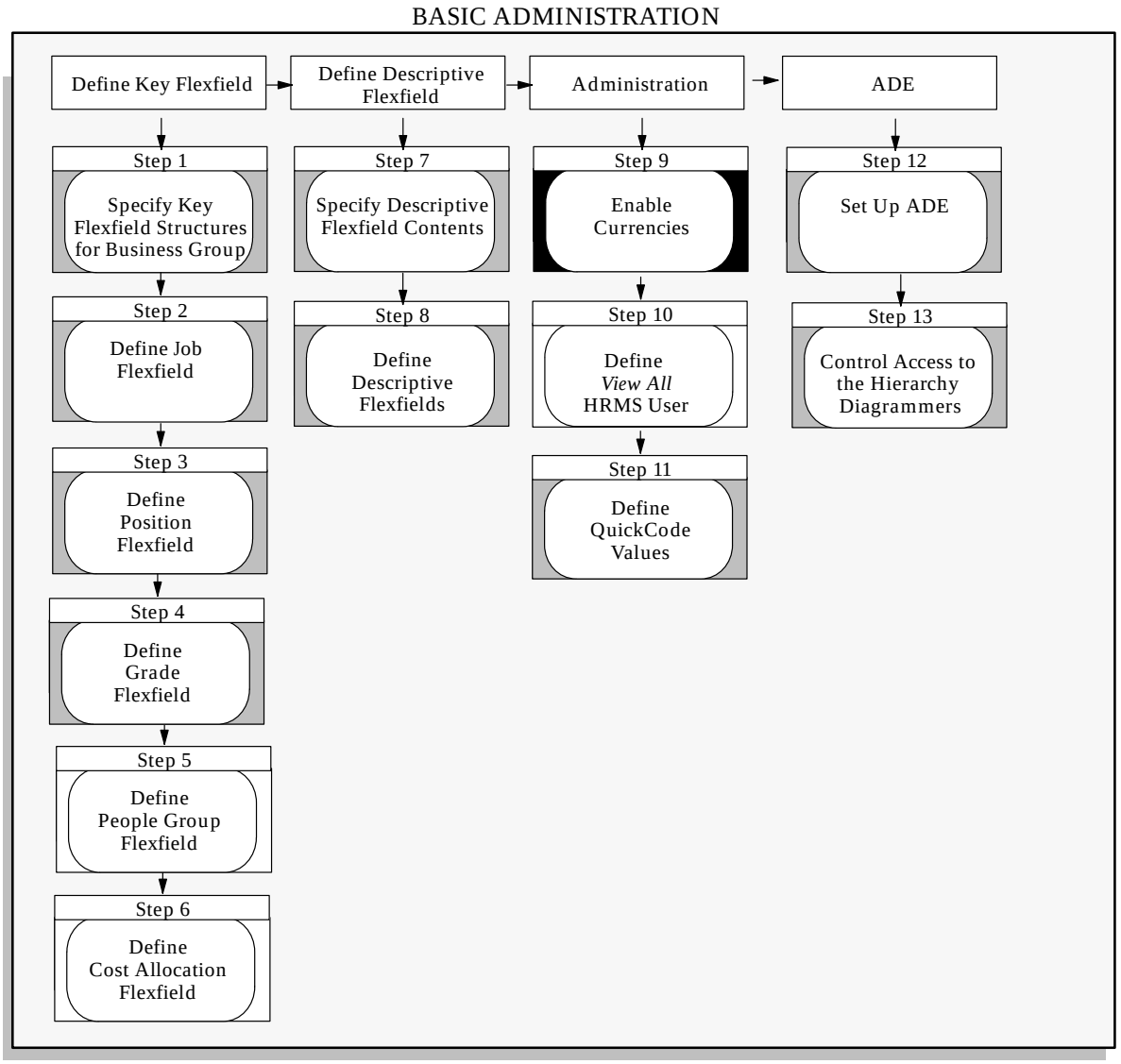


Figure 1 – 3
Implementation Flowchart for Work Structures

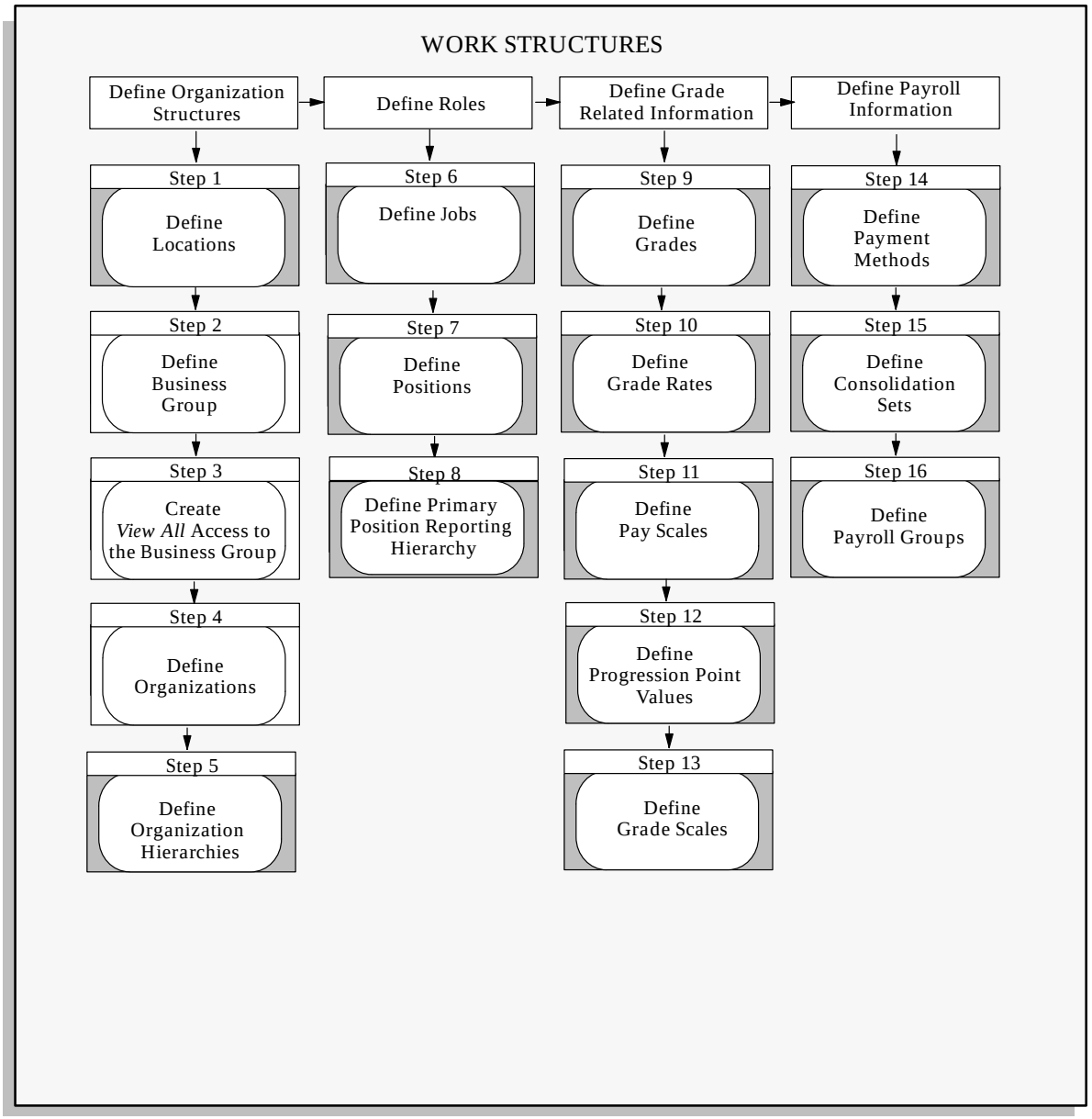


Figure 1 – 4
Implementation Flowchart for Compensation and Benefits

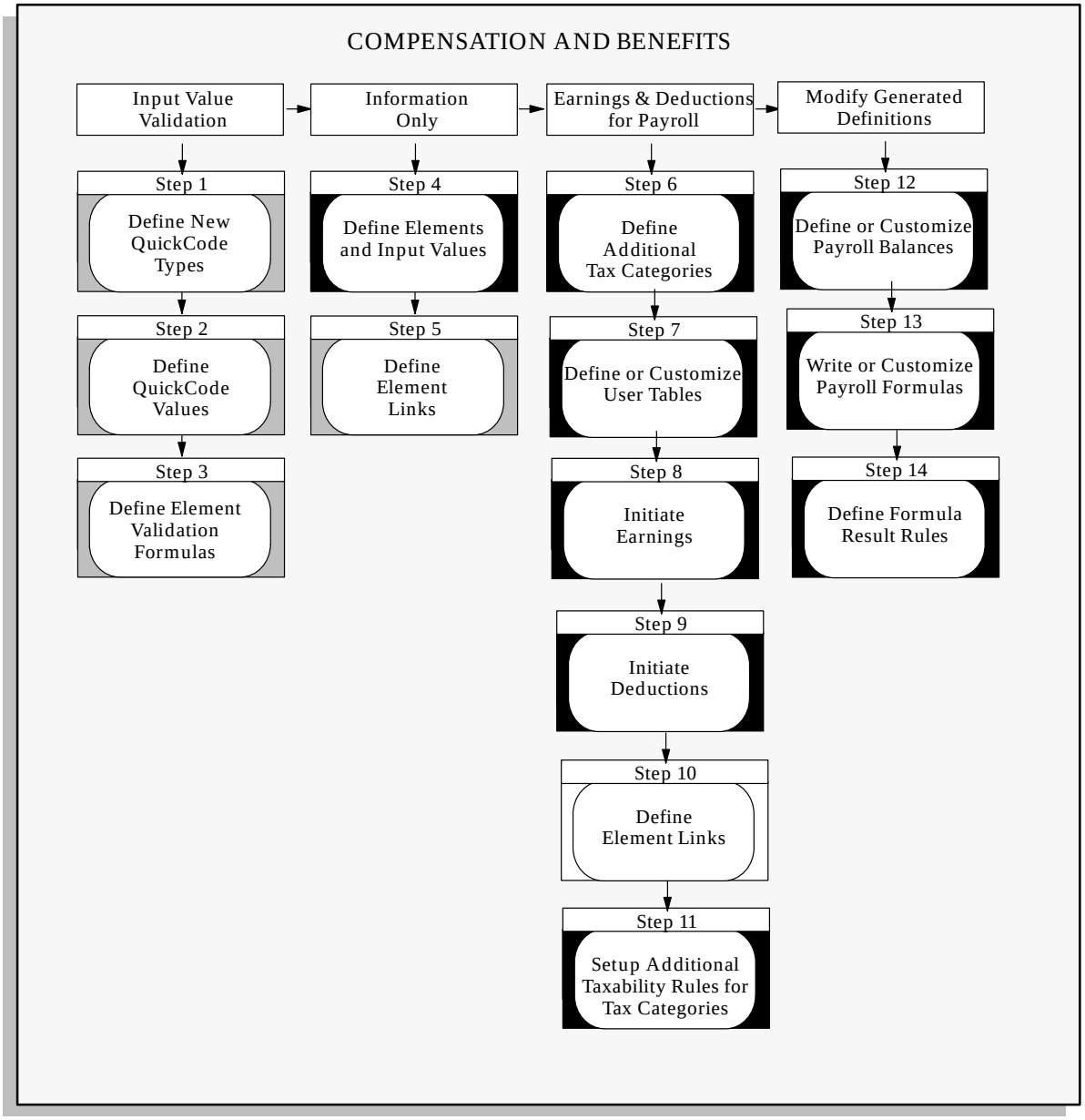


Figure 1 – 5
Implementation Flowchart for
Salary and Benefits Administration

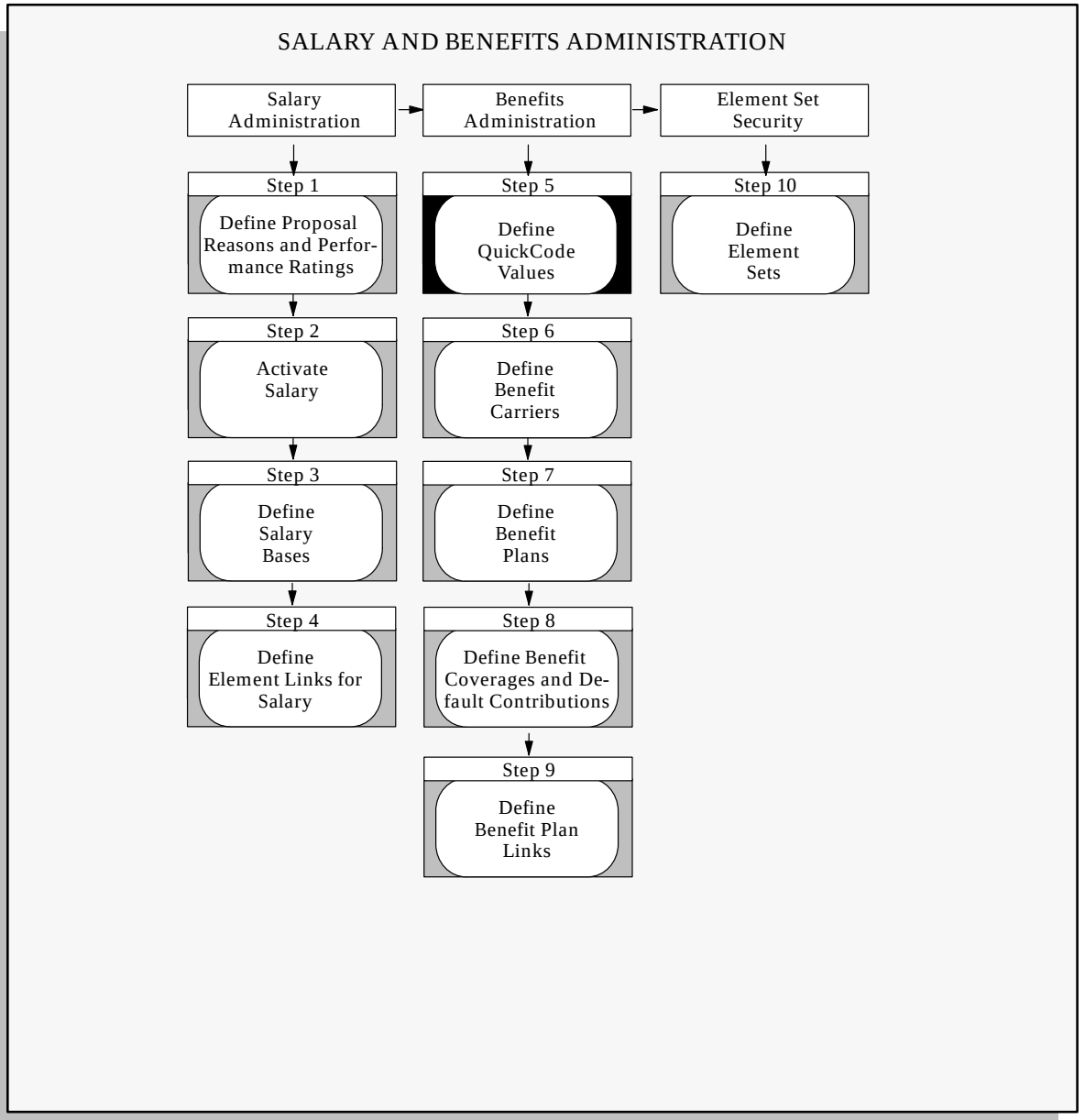


Figure 1 – 6
Implementation Flowchart for People and Assignments

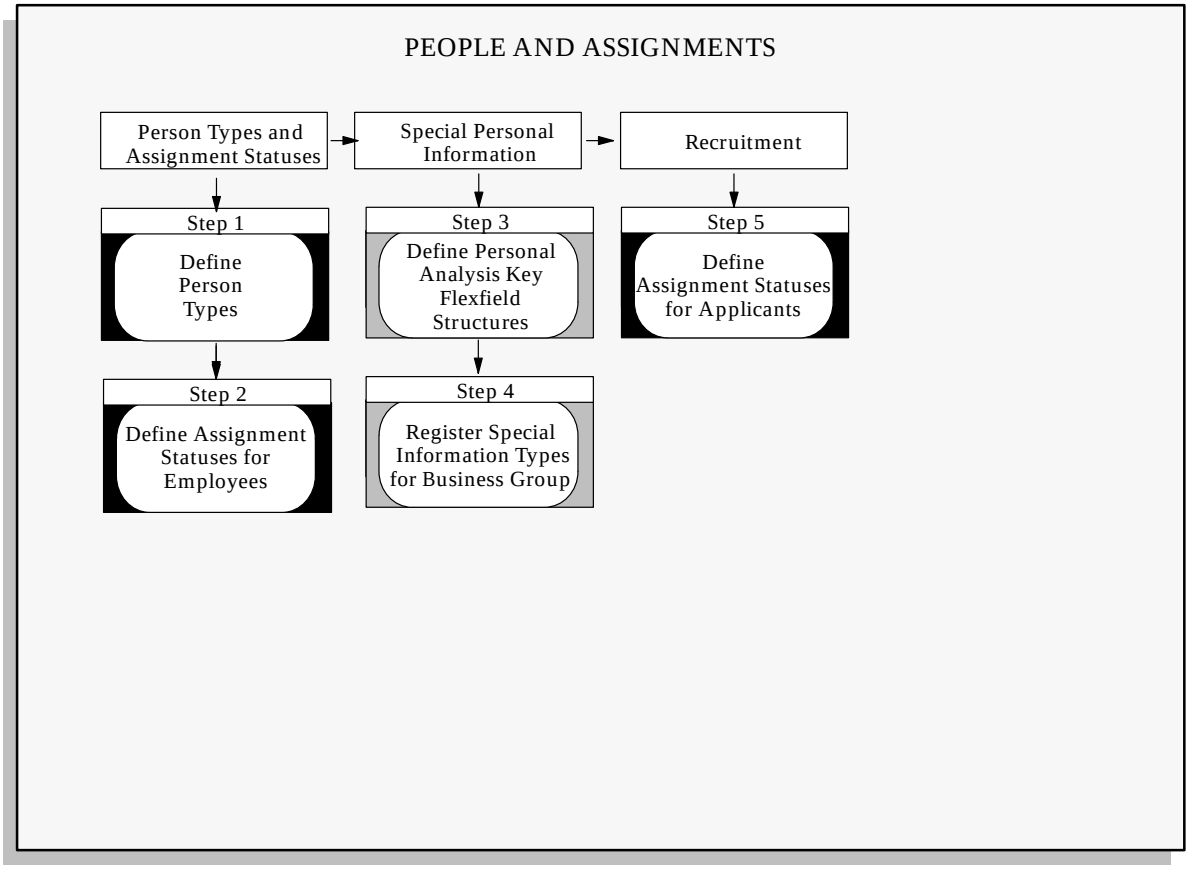


Figure 1 – 7
Implementation Flowchart for Career Management

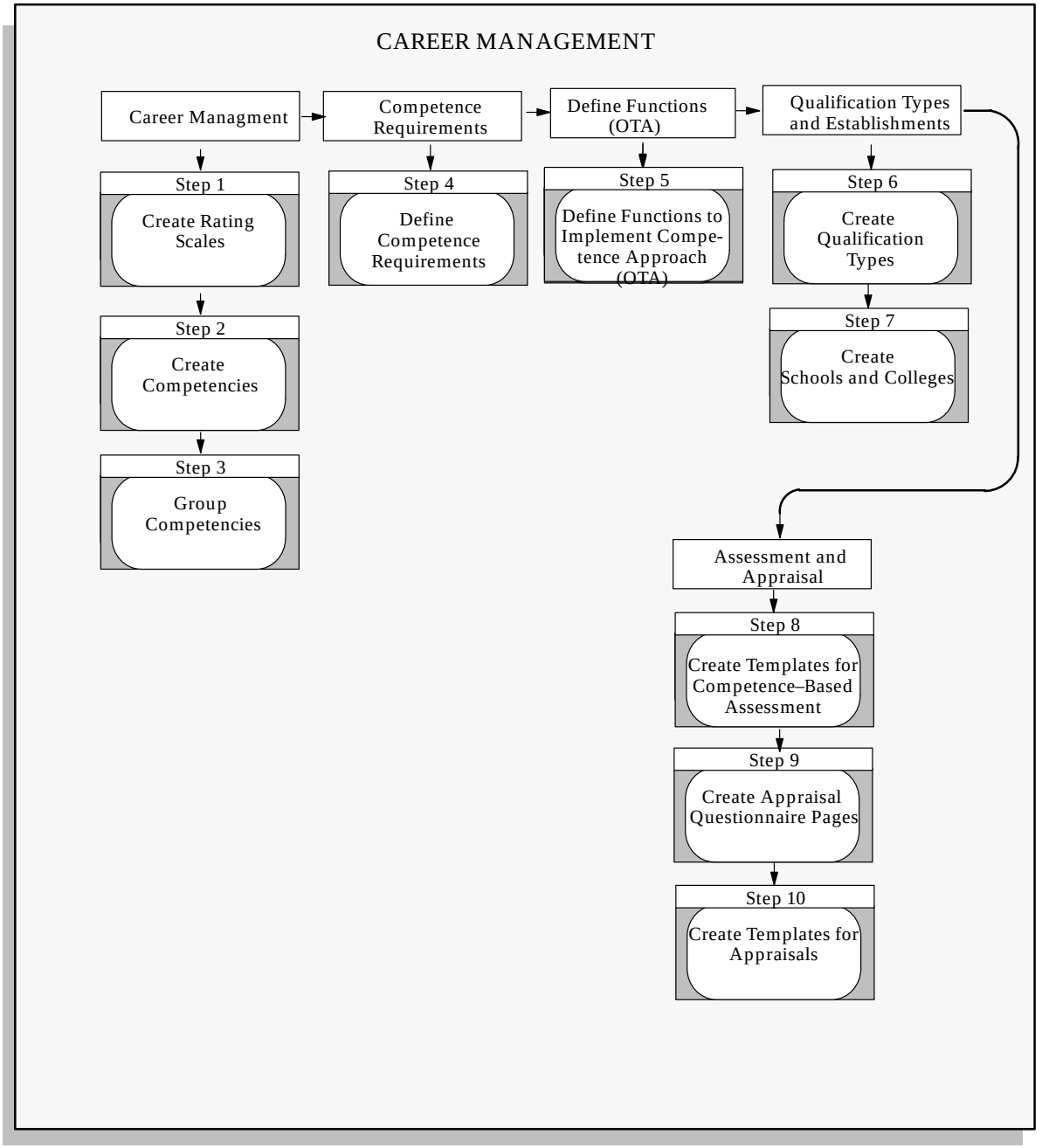


Figure 1 – 8
Implementation Flowchart for
New Hire Reporting

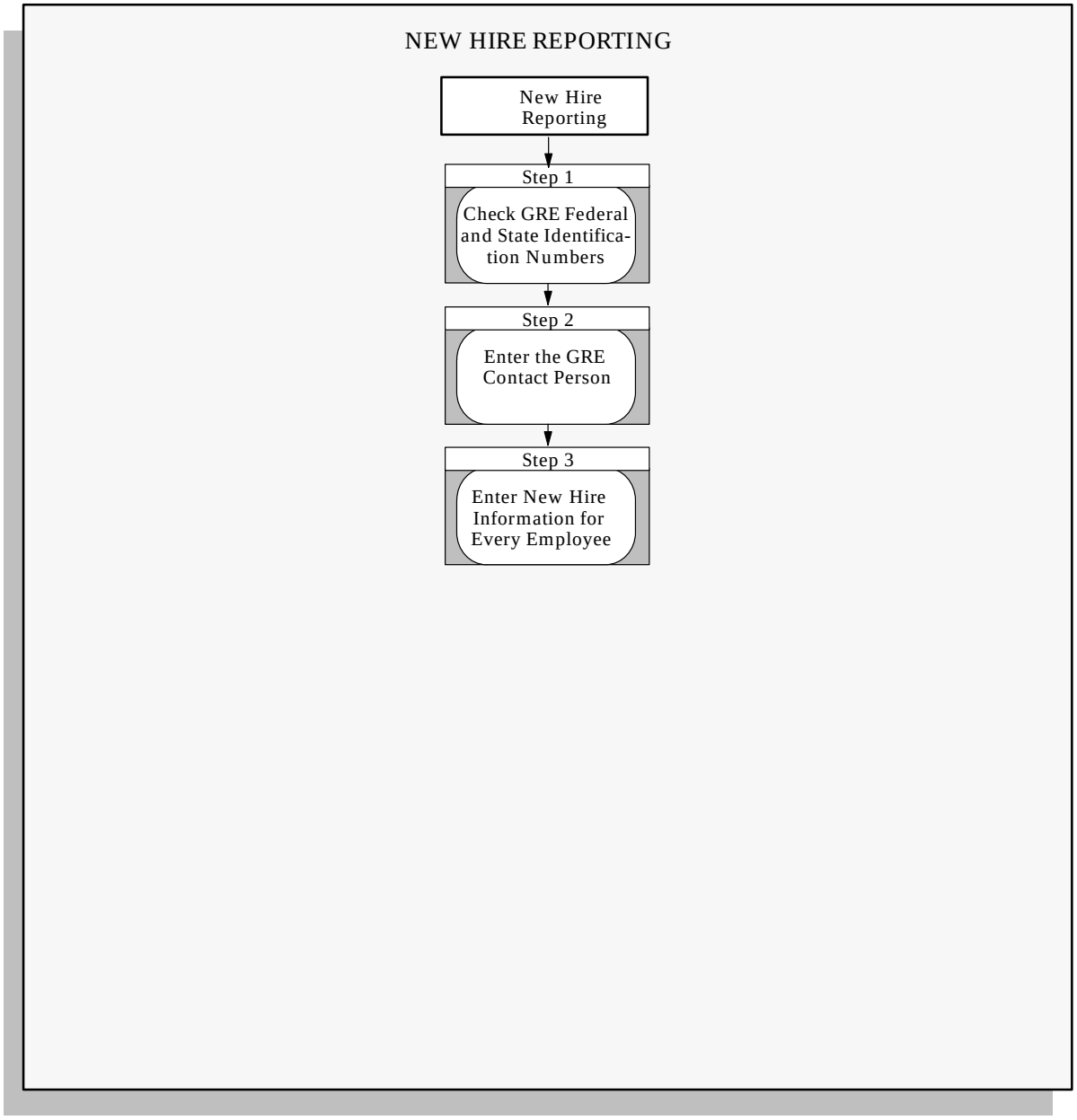


Figure 1 – 9
Implementation Flowchart for Specific Business Functions 1

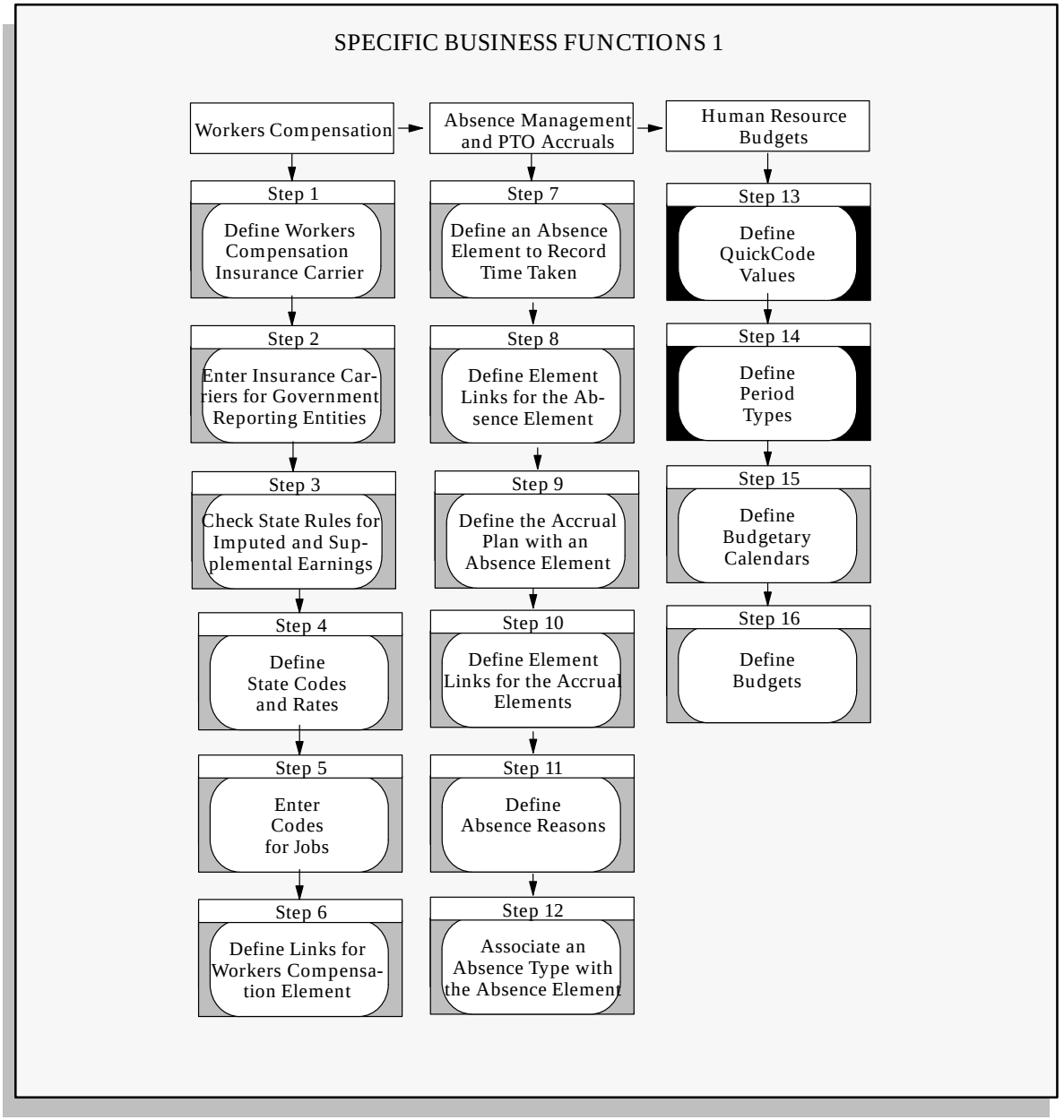


Figure 1 – 10
Implementation Flowchart for Specific Business Functions 2

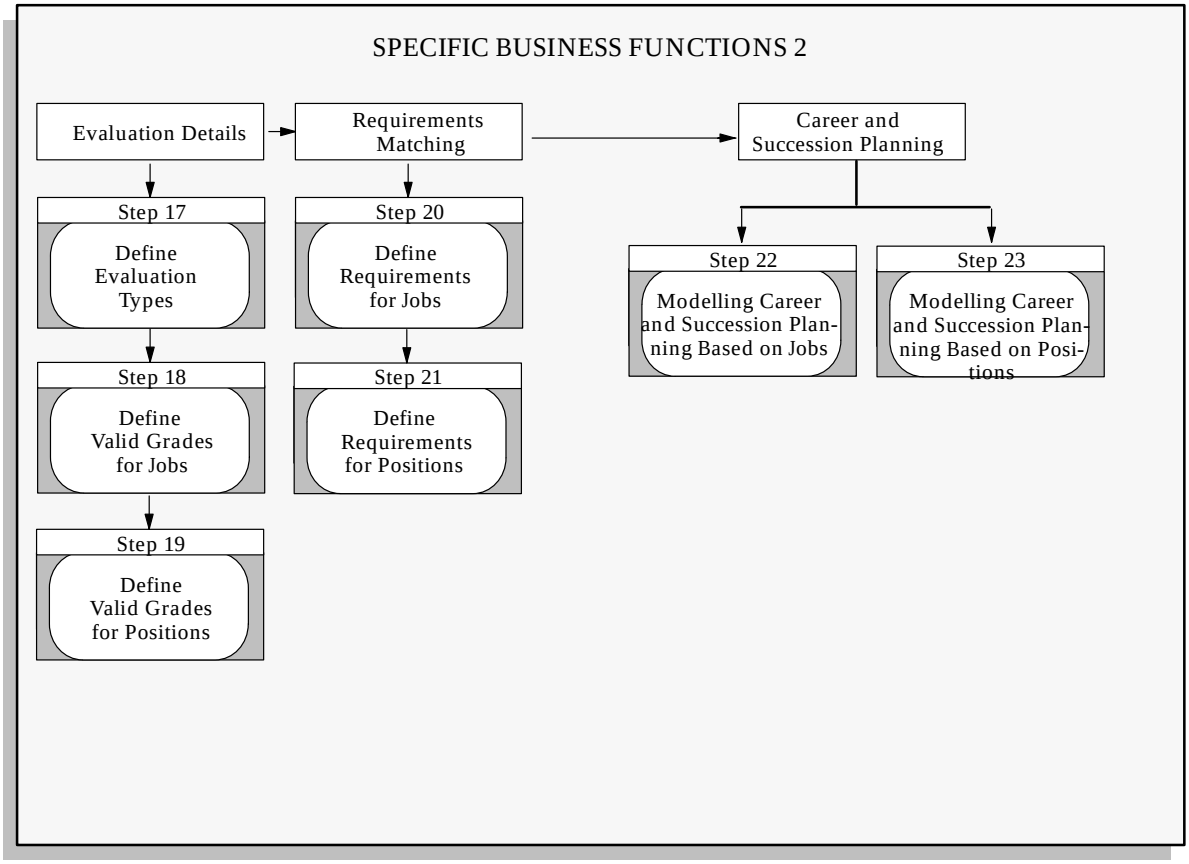
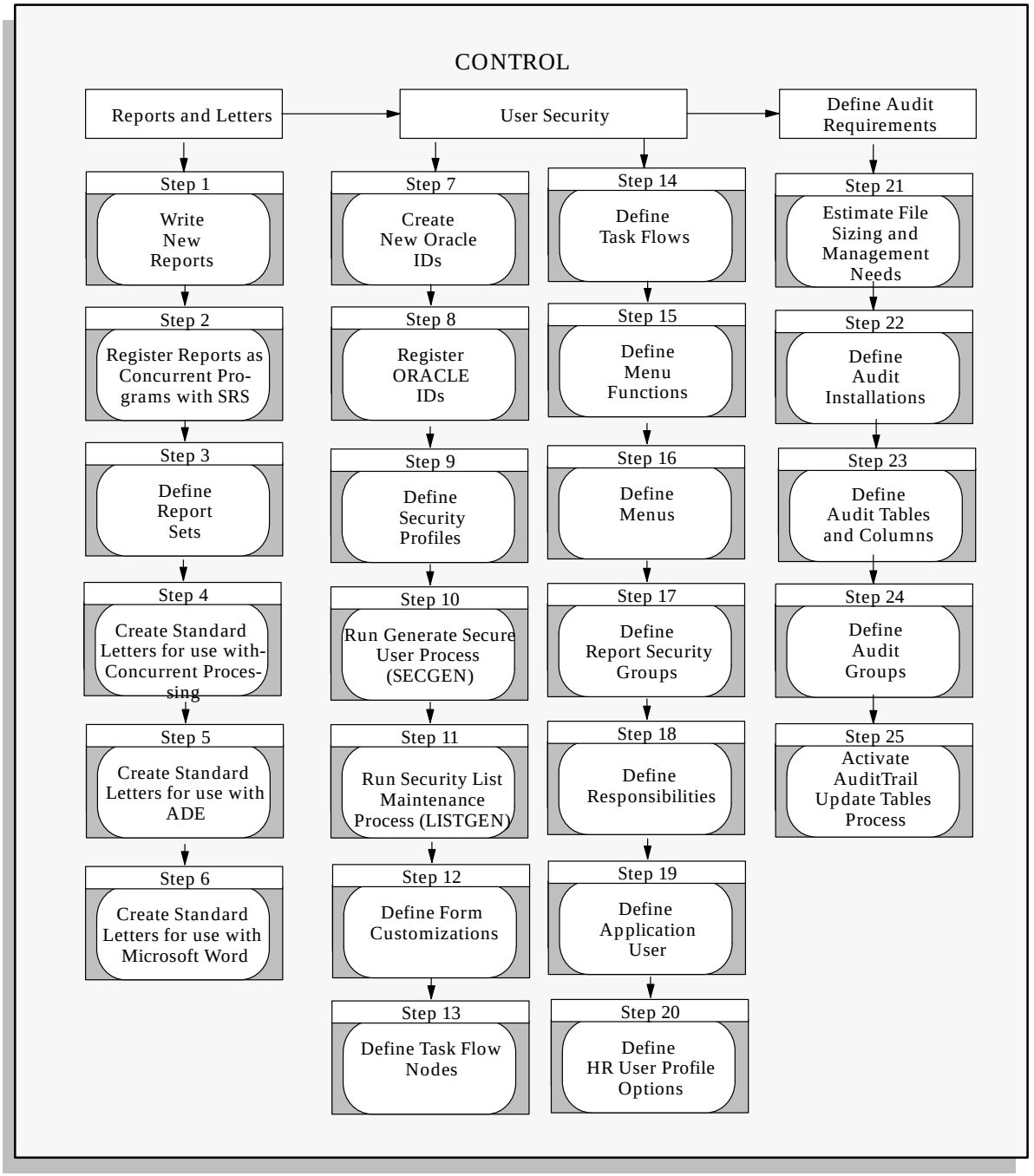


Figure 1 – 11
Implementation Flowchart for Control



Implementation Checklists

Use the following checklists to record which parts of Oracle HRMS you want to use. This will help you to complete the appropriate steps in the correct order for your implementation.

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

Basic Administration Checklist

- ❑ **Define Key Flexfields**
  - ❑ 1 Specify Key Flexfield Structures for Business Group
  - ❑ 2 Define Job Flexfield
  - ❑ 3 Define Position Flexfield
  - ❑ 4 Define Grade Flexfield
  - ❑ 5 Define People Group Flexfield
  - ❑ 6 Define Cost Allocation Flexfield
- ❑ **Define Descriptive Flexfields**
  - ❑ 7 Specify Descriptive Flexfield Contexts
  - ❑ 8 Define Descriptive Flexfields
- ❑ **Administration**
  - ❑ 9 Enable Currencies
  - ❑ 10 Define 'View All' HRMS User
  - ❑ 11 Define QuickCode Values
- ❑ **Application Data Export (ADE) and Hierarchy Diagrammers**
  - ❑ 12 Set Up ADE
  - ❑ 13 Control Access to the Hierarchy Diagrammers

Work Structures Checklist

- ❑ **Define Organization Structures**
  - ❑ 14 Define Locations
  - ❑ 15 Define Business Group
  - ❑ 16 Create 'View–All' access to the Business Group
  - ❑ 17 Define Organizations and Other Reporting Groups
  - ❑ 18 Define Organization Hierarchies
- ❑ **Define Roles**
  - ❑ 19 Define Jobs
  - ❑ 20 Define Positions
  - ❑ 21 Define Primary Position Reporting Hierarchy
- ❑ **Define Grade Related Information**
  - ❑ 22 Define Grades
  - ❑ 23 Define Grade Rates
  - ❑ 24 Define Pay Scales
  - ❑ 25 Define Progression Point Values
  - ❑ 26 Define Grade Scales
- ❑ **Define Payroll Information**
  - ❑ 27 Define Payment Methods
  - ❑ 28 Define Consolidation Set
  - ❑ 29 Define Payroll Groups

Compensation and Benefits Checklist

- ❑ **Define Input Value Validation**
  - ❑ 30 Define New QuickCode Types
  - ❑ 31 Define QuickCode Values
  - ❑ 32 Define Element Validation Formulas
- ❑ **Define Compensation and Benefits for Information**
  - ❑ 33 Define Elements and Input Values
  - ❑ 34 Define Element Links
- ❑ **Define Earnings and Deductions for Payroll Processing**
  - ❑ 35 Define Additional Tax Categories
  - ❑ 36 Define or Customize User Tables
  - ❑ 37 Initiate Earnings
  - ❑ 38 Initiate Deductions
  - ❑ 39 Define Element Links
  - ❑ 40 Setup Additional Taxability Rules for Tax Categories
- ❑ **Customize the Generated Definitions**
  - ❑ 41 Define or Customize Payroll Balances
  - ❑ 42 Write or Customize Payroll Formulas
  - ❑ 43 Define Formula Result Rules
- ❑ **Salary Administration**
  - ❑ 44 Define Proposal Reasons and Performance Ratings
  - ❑ 45 Activate Salary
  - ❑ 46 Define Salary Bases
  - ❑ 47 Define Element Links for Salary
- ❑ **Set up Benefits Administration**
  - ❑ 48 Define QuickCode Values
  - ❑ 49 Define Benefit Carriers
  - ❑ 50 Define Benefit Plans

- ❑ 51 Define Benefit Coverages and Default Contributions
- ❑ 52 Define Benefit Plan Links

❑ **Element Sets**

- ❑ 53 Define Element Sets

People and Assignments Checklist

- ❑ **Person Types and Assignment Statuses**
  - ❑ 54 Define Person Types
  - ❑ 55 Define Assignment Statuses for Employees
- ❑ **Special Personal Information**
  - ❑ 56 Define Personal Analysis Key Flexfield Structures
  - ❑ 57 Register Special Info Types for the Business Group
- ❑ **Recruitment**
  - ❑ 58 Define Assignment Statuses for applicants

Career Management Checklist

- ❑ **Methods of Measurement and Creating Competencies**
  - ❑ 59 Create Rating Scales
  - ❑ 60 Create Competencies
  - ❑ 61 Group Competencies into Types
- ❑ **Competence Requirements**
  - ❑ 62 Create Competence Requirements
- ❑ **Define Functions to Implement the Competence Approach (OTA)**
  - ❑ 63 Define Functions to Implement the Competence Approach (OTA)
- ❑ **Qualification Types and Establishments**
  - ❑ 64 Create Qualification Types
  - ❑ 65 Create Establishments
- ❑ **Assessment and Appraisal**
  - ❑ 66 Create Assessment Templates
  - ❑ 67 Create Appraisal Questionnaires
  - ❑ 68 Create Appraisal Templates

New Hire Reporting

- ❑ **New Hire Reporting**
  - ❑ 69 Check GRE Federal and State Identification Numbers
  - ❑ 70 Enter the GRE Contact Person
  - ❑ 71 Enter New Hire Information for Every Employee

Specific Business Functions Checklist

- ❑ **Workers Compensation**
  - ❑ 72 Define Workers Compensation Insurance Carrier
  - ❑ 73 Enter Insurance Carriers for GREs
  - ❑ 74 Check State Rules for Earnings
  - ❑ 75 Define State Codes and Rates
  - ❑ 76 Enter Codes for Jobs
  - ❑ 77 Define Element Links for WC Element
- ❑ **Absence Management /Accruals of Paid Time Off (PTO)**
  - ❑ 78 Define an Absence Element to Record Time Taken
  - ❑ 79 Define Element Links for the Absence Element
  - ❑ 80 Define the Accrual Plan with an Absence Element
  - ❑ 81 Define Element Links for the Accrual Elements
  - ❑ 82 Define Absence Reasons
  - ❑ 83 Associate an Absence Type with the Absence Element
- ❑ **Human Resource Budgets**
  - ❑ 84 Define QuickCode Values
  - ❑ 85 Define Period Types
  - ❑ 86 Define Budgetary Calendars
  - ❑ 87 Define Budgets
- ❑ **Evaluation Systems**
  - ❑ 88 Define Evaluation Types
  - ❑ 89 Define Valid Grades for Jobs
  - ❑ 90 Define Valid Grades for Positions
- ❑ **Requirements Matching**
  - ❑ 91 Define Requirements for Jobs
  - ❑ 92 Define Requirements for Positions
- ❑ **Career and Succession Planning**

- ❑ 93 Modelling Career and Succession Planning Based on Jobs
- ❑ 94 Modelling Career and Succession Planning Based on Positions

Control Checklist

- ❑ **Define Reports and Generate Standard Letters**
  - ❑ 95 Write New Reports
  - ❑ 96 Register Reports as Concurrent Programs with SRS
  - ❑ 97 Define Report Sets
  - ❑ 98 Create Standard Letters for use with Concurrent Processing
  - ❑ 99 Create Standard Letters for use with Application Data Export (ADE)
  - ❑ 100 Create Standard Letters for use with Microsoft Word
- ❑ **Define User Security**
  - ❑ 101 Create Oracle IDs
  - ❑ 102 Register Oracle IDs
  - ❑ 103 Define Security Profiles
  - ❑ 104 Run *Generate Secure User* Process
  - ❑ 105 Run *Security List Maintenance* Process
  - ❑ 106 Define Form Customizations
  - ❑ 107 Define Task Flow Nodes
  - ❑ 108 Define Task Flows
  - ❑ 109 Define Menu Functions
  - ❑ 110 Define Menus
  - ❑ 111 Define Report Security Groups
  - ❑ 112 Define Responsibilities
  - ❑ 113 Define Application Users
  - ❑ 114 Define HR User Profile Options
- ❑ **Define Audit Requirements**
  - ❑ 115 Estimate file sizing and file management needs
  - ❑ 116 Define Audit Installations
  - ❑ 117 Define Audit Tables and Columns

- ❑ 118 Define Audit Groups
- ❑ 119 Activate AuditTrail Update Tables Process

CHAPTER

2 Implementation Steps

This chapter expands the implementation steps in the checklist and provides a summary of what you can set up in each functional area in the sequence that you should follow. Every step gives you the names of any forms or processes that you should use for that step.

Detailed task information for each step is included in the *Oracle Human Resources User's Guide* and the *Oracle Payroll User's Guide*. You can easily access this information using the online help system.

Attention: If you try to implement functionality in Oracle HRMS before you are familiar with it you may make mistakes. Before you do each step you should read any topic information that is referenced.

Implementation Steps: Basic Administration

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

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

See:Appendix B Key and Descriptive Flexfields,
Costing in Oracle HRMS,
(Oracle Payroll or Human Resources User's Guide)

Labor Costs in Oracle HRMS,
(*Oracle Payroll User's Guide)*

Step 1 Specify Key Flexfield Structures for Business Group

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

The sequence which you follow to implement each Flexfield is:

- Define Flexfield Value Sets
- Define Key Flexfield Segments
- Define Flexfield Segment Values
- Define Key Flexfield Cross–Validation Rules

- Define Key Flexfield Aliases
- Freeze and Compile Key Flexfield Structure

When you have completed the definition of a key flexfield you can run a special 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.

- Run Create Key Flexfield Database Items process

Step 2 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:

1. **Define Job 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.

    Define Value Set

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.

    Define Key Flexfield Segments

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.

   
   Define Segment Values

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.

   
   Define Cross–Validation Rules

5. **Define Job Flexfield Aliases**

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

   
   Define Shorthand Aliases

6. **Freeze and Compile Your Job Flexfield Structure**

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

   
   Define Key Flexfield Segments

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.

   
   Submit a New Request

Step 3 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:

1. **Define Position 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.

    Define Value Set

2. **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.

    Define Key Flexfield Segments

3. **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.

    Define Segment Values

4. **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.


Define Cross–Validation Rule

5. **Define Position Flexfield Aliases**

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


Define Shorthand Aliases

6. **Freeze and Compile Your Position Flexfield Structure**

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


Define Key Flexfield Segments

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.


Submit a New Request

Step 4 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:

1. **Define Grade 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.

Define Value Set

2. **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.

Define Key Flexfield Segments

3. **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.

Define Segment Values

4. **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.

Define Cross–Validation Rule

5. **Define Grade Flexfield Aliases**

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

Define Shorthand Aliases

6. **Freeze and Compile Your Grade Flexfield Structure**

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

    Define Key Flexfield Segments

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.

    Submit a New Request

Step 5 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:

1. **Define People Group 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.

    Define Value Set

2. **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.

    Define Key Flexfield Segments

3. **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.

    Define Segment Values

4. **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.

    Define Cross–Validation Rule

5. **Define People Group Flexfield Aliases**

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

    Define Shorthand Aliases

6. **Freeze and Compile Your People Group Flexfield Structure**

   You are now ready to freeze your People Group Flexfield definition. Navigate to the Define Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your

changes. Oracle Human Resource Management Systems now freezes and compiles your People Group Flexfield definition. Compiling the flexfield definition enables the People Group Flexfield window with the defaults, values and rules that you have defined.

 Define Key Flexfield Segments

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.

 Submit a New Request

Step 6 Define Cost Allocation Flexfield

See: Labor Costs in Oracle HRMS, *Oracle Payroll User's Guide*

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.

See: Costing in Oracle HRMS,
(Oracle Payroll or Oracle Human Resources User's Guide)

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 forms with the flexfield window.

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:

1. **Define Cost Allocation 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.


Define Value Set

2. **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.


Define Key Flexfield Segments

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.

Table 2 – 1

3. **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.


Define Segment Values

4. **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.


Define Cross–Validation Rule

5. **Define Cost Allocation Flexfield Aliases**

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


Define Shorthand Aliases

6. **Freeze and Compile Your Cost Allocation Flexfield Structure**

   You are now ready to freeze your Cost Allocation Flexfield definition. Navigate to the Define 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.


Define Key Flexfield Segments

Define Descriptive Flexfields

See: User Definable Descriptive Flexfields,
(Oracle Payroll or Oracle Human Resources User's Guide)

Step 7 Specify Descriptive Flexfield Contexts for Additional Details

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.

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

See: *Oracle HRMS Technical Reference Manual*

Step 8 Define Descriptive Flexfields

The sequence which you follow to implement each descriptive flexfield is:

- Register any field in the same block of the window as the Context Reference Field. This field will supply the context for any context sensitive segments.
- Define Flexfield Values Sets
- Define Descriptive Flexfield Segments
- Define Flexfield Segment Values
- Run the Create Descriptive Flexfields Database Items Process

The sub–steps that follow show you how to perform all these steps.

1. **Register a Reference Field**

   You must use the *Application Developer* Responsibility to update the definition of the descriptive flexfield.

   
   Register Descriptive Flexfields

   - Query the flexfield you want to update.
   - Navigate to the Reference Fields block and enter the name of the Reference Field you want to use.
   - Save your choices.

   
   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.

2. **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.

   
   Define Value Set

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

3. **Define Descriptive Flexfield Segments.**

   Define the segments of your descriptive flexfield for each Context.

   - Use Global Context to define any segments which will always appear in the flexfield window.
   - Enter your own Context Name to define segments which will appear only for that context.

   Freeze and compile your descriptive flexfield definitions.

   
   Define Descriptive Flexfield Segments

   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.

   
   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.

4. **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.

   
   Define Segment Values

5. **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.

   
   Submit a New Request

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.

Step 9 Enable Currencies

This is a task for your System Administrator.

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.

 Currencies

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

Step 10 Define 'View All' HRMS User

This is a task for your System Administrator.

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

 Users

1. Define a new Username and Password.
2. Pick the default Responsibility with a Start Date and save your choice.

Note: After you have completed this step you can sign on and use the new responsibility to run the Grant Permissions to Roles process, define QuickCodes, Locations and your new Business Group.

Step 11 Define QuickCode Values

See: QuickCodes, *(Oracle Payroll or Oracle Human Resources User's Guide)*

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

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

QuickCode 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 QuickCodes 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.

Suggestion: You should use Start and End Dates to control when values appear in a list. Using DateTrack you can change your effective date to view and update information at any point in time.

Also, when you add or change QuickCode Values you should sign on again to use the new value.

QuickCodes

Implementation Steps: Application Data Export (ADE) and Hierarchy Diagrammers

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

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

Step 12 Set Up ADE

You can use ADE to launch another application within Oracle HRMS. Applications are invoked and run in another window on your desktop.

You can invoke ADE in three different ways:

- **Standalone mode**. ADE is invoked from your PC desktop and run independently, without accessing your application.
- **Application mode**. ADE is invoked from a button on your application toolbar and run within your application.
- **Letter request mode**. ADE is invoked from the Merge button in the Request Letter window in Oracle HRMS or Oracle Training Administration.

ADE comes with its own set of documentation and online help.

See: Setup Overview,
Oracle Application Data Export User's Guide.

Step 13 Control Access to the Hierarchy Diagrammers

If you want to graphically create and maintain your Organisation and Position hierarchies, you must control access to the Hierarchy diagrammers.

The Hierarchy diagrammers come with their own set of documentation and online help.

> **Note:** We provide online help only for this version of the Hierarchy diagrammers. Refer to the online help to enable you to control access to the Hierarchy diagrammers.

Implementation Steps: Work Structures

See: Work Structures, *(Oracle Payroll or Oracle Human Resources User's Guide)*

Define Organization Structures

Step 14 Define Locations

Use this window to define 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.


Location

This window contains information that is shared with users of other Oracle Applications, such as, 'Inventory Organization'.


Attention: If you have installed Oracle Payroll, you must enter a valid City and Zip Code for every site address. The system automatically uses this information to generate the taxation codes it needs for the calculation of state and local taxes.

Step 15 Define 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.

Note: 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.


Organization

- Enter Business Group identification information and enter the appropriate key flexfield structures.
- Enter any defaults that apply to the whole business group
  - Establish employment categories and assignment statuses used for VETS–100 reporting

– If you intend to use PayMix, specify the Cost Allocation flexfield segments you want to use for Cost Center and Labor Distribution codes. These will appear in the PayMix window.

- When you save a new business group the system will generate a 'view all' security profile for this new group. This is linked to the *base–user* of the Oracle HRMS tables.

Warning: If you intend to process payrolls in your business group it is essential that you select a valid legislation code and base currency. The system uses these values to copy in the data it needs to comply with your payroll legislative requirements.

You cannot change these definitions after they have been saved.

Step 16 Create 'View All' Access to the Business Group

This is a task for your System Administrator. If you are using the Setup Business Group supplied with Oracle HRMS, you can omit this step.

See: Security in Oracle HRMS, *(Oracle Payroll or Oracle Human Resources User's Guide)*

System Profile Values

1. Select the default responsibility.
2. Query the HR:Security Profile option and enter the name of the view–all security profile for your Business Group. By default this is the name of your Business Group.

Warning: If you define a new Business Group you must change the user profile option *HR:Security Profile* for your default Responsibility to point at the view–all security profile for your new group.

If you use more than one Business Group you must set up a separate responsibility and user profile for each group.

Step 17 Define Organizations and Other Reporting Groups

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, or to represent carriers for benefits and Workers Compensation, or for third party payments.

In addition to these internal organizations you can define other organizations for tax and government reporting purposes, or 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 QuickCode Type ORG_CLASS, and provide options for all users of the *Organization* window.

You can disable the QuickCode values you will not use in your implementation in the *QuickCodes* window.

Organization

- Enter a unique name for every organization in the business group.
- Enter classifications as appropriate
  - Use HR Organization if you want to include this in any employee assignment
  - Enter default Work Schedule details to prorate pay for employees who do not submit timecards for payroll.
- For Cost Center reporting you can enable a segment of the *Cost Allocation* flexfield with a qualifier of Organization.
- For government reporting purposes you must define at least one Government Reporting Entity. You can associate the following types of information with a GRE:
  - Employer Identification Number
  - EEO–1 Filing
  - Federal Tax Rules
  - Local Tax Rules
  - Multiple Worksite Reporting
  - NACHA Rules
  - State Tax Rules
  - W2 Reporting Rules
  - VETS–100 Filing

See: Overview of Government–mandated Reporting, *(Oracle Human Resources User's Guide)*

- You can place GREs into Tax Groups by associating them with a Tax Group in the Federal Tax Rules window

Note: A Tax Group can include any number of GREs you have defined. Tax Groups are useful when one GRE should be a Common Paymaster for others in the same Tax Group.

See: Varieties of Establishment Hierarchies, (*Oracle Human Resources User's Guide*)

- You can define benefits carriers and carriers for Workers Compensation as external organizations

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–1901. This will simplify your data–entry.

Step 18 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.

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

For government reporting such as EEO–1 or VETS–100 reporting, you must define one or more Reporting Establishments to represent your work sites. If you have more than one such establishment, you must build establishment hierarchies to determine the employees a particular report will cover.

– You will also need to define FLSA codes for jobs, and EEO codes, salary codes and Job Groups for EEO–1 and AAP reporting

Organization 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.

Define Roles

Oracle HRMS lets you define the roles that employees perform as Jobs or Positions, or a combination of both.

Step 19 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.

 Job

- 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.
- Enter the FLSA codes for every job you define. For EEO–1 and AAP reporting, you must also enter EEO–1 codes, salary codes and Job Groups.

If you intend loading historic assignment details into Oracle HRMS, make sure you enter valid start dates for your jobs.

You cannot assign an employee to a job before the start date of the job.

Suggestion: Consider using a fixed date as a default for your initial setup, for example, 01–JAN–1901. This simplifies your data entry.

Step 20 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.

 Position

- You must define jobs before you define 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.

- If you intend loading historic assignment details into Oracle HRMS, make sure you enter valid start dates for your positions.
- You cannot assign an employee to a position before the start date of the position.

Suggestion: Consider using a fixed date as a default for your initial setup, for example, 01–JAN–1901. This will simplify your data–entry.

Step 21 Define Primary Position Reporting 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.

Each position can belong to any number of hierarchies at the same time, but can only appear once in any hierarchy.

You should define the primary reporting hierarchy as part of your implementation of positions. The first version of your hierarchy should show your reporting structures when you implement Oracle HR.

Position Hierarchy

You can generate a new version of a position hierarchy at any time with start and end dates. This allows you to keep the history of your reporting structures, and to use the system to plan future changes. The dates you enter are used to identify the 'current' hierarchy for all reporting and inquiry purposes. When you use DateTrack you see the 'current' hierarchy.

Suggestion: You may find it easier to define the Hierarchy using the top position and one other. Then you can add other positions into the hierarchy when you make your definitions in the Position window.

You always build a new hierarchy from the top down.

Define Grade Related Information

See:

Representing Grade Structures,
Grade Relationships to Compensation and Benefits,
(Oracle Payroll or Oracle Human Resources User's Guide)

Step 22 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*.


Grades


Attention: If you intend loading historic assignment details into Oracle HRMS, make sure you enter valid start dates for your grades. You cannot assign an employee to a grade before the start date of the grade.


Suggestion: Consider using a fixed date as a default for your initial setup, for example, 01–JAN–1901. This will simplify your data–entry.

Step 23 Define Grade Rates

See: Relating Pay to Grades Directly: Using Grade Rates, *(Oracle Payroll or Oracle Human Resources User's Guide)*

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.


Grade Rate


Attention: Grade rate values are DateTracked and you must ensure you use the correct date to create your initial set of values.

If you intend loading historic grade rate details into Oracle HRMS, make sure you enter the correct start date for all your history.

Step 24 Define Pay Scales

See: Relating Pay to Grades Indirectly: Using Pay Scales, *(Oracle Payroll or Oracle Human Resources User's Guide)*

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'. Characteristics of this functionality are:

- A single scale of points and values is used to establish the actual pay for a grade group
- Each point in the pay scale has a single value
- Grades can have a number of distinct *steps*, with each step given a single point in the pay scale
- An employee assignment to a grade includes a point, or step value, and the point value determines the actual pay of the employee

The first step in this process is to define the *Pay Scales* you want to use. You can have any number of different pay scales in Oracle HRMS. Each scale has its own set of points which may be characters or numbers.


Pay Scale

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.

Step 25 Define Progression Point Values

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


Scale Rate


Attention: You must use the correct date to create your initial set of values. If you intend loading historic pay scale values into Oracle HRMS, you must use correct dates for all your history.

Step 26 Define Grade Scales

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

A grade can have any number of steps. Steps do not always have the same interval as the pay scale points. For example, you may have a pay scale with points from 1 to 10, and a Grade which has 5 steps with points in the sequence 3, 5, 7, 8, and 9.

Grade Scale

Note: 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.

Define Payroll Information

See: Representing Payrolls, *(Oracle Payroll or Oracle Human Resources User's Guide)*

In Oracle HRMS payrolls define groups of employees who share a common frequency of payroll processing.

Attention: 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 27 Define Payment Methods

Standard categories of payment methods such as 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.

Organizational Payment Method

- You can define multiple payment methods for the same category. For example, you might have different source bank accounts for payments.

  Note: After you define your Payment Methods you can enter the available types for each payroll you define.

  After you assign an employee to a payroll you can enter payment details for each employee in the *Personal Payment*

Method window. For example, for employees who work overseas, you may want to record more than one payment method with different percentages, and currencies.

Step 28 Define Consolidation Set

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.


Consolidation Sets

Step 29 Define Payroll Groups

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.


Payroll

- Select the Period Type and an End Date for the First Period
- Enter the number of years you want to generate
  - You can increase this number when you need to generate new calendar years
- The system automatically generates your payroll calendar with the correct start and end dates for each pay period and year. These dates are correct for the legislation of your Business Group and you can modify the generated dates to take account of any special holidays.
- Cut–off and payment dates are for information and reference only. They do not determine when your payroll is processed or when employees are paid. You control this when you set up and administer your payroll processing.
- Enter all valid payment methods for employees on each payroll. The method must be valid for the payroll before you can use it for an employee.

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

Implementation Steps: Compensation and Benefits

See: Overview of Elements, *Oracle Human Resources User's Guide*

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

If you need to define an element for information only, you do this in the Element window.

Warning: If you intend to process an element in a payroll run you **MUST** use the Earnings or Deductions windows to initiate the type of earnings or deduction. The system will generate the elements, input values, formulas, results and balance definitions you need for accurate payroll processing.

You can modify these definitions to meet your specific business needs.

Define Input Value Validation

Step 30 Define New QuickCode Types

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

QuickCode Types

Note: You can also use QuickCode 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.

Step 31 Define QuickCode Values

You add new QuickCodes Values at any time. You can set the *Enable Flag* for a value to 'No', so that value will no longer appear in any list of values.

Suggestion: Use Start and End Dates to control when values appear in a list. With DateTrack you change your effective date to view and update information at any point in time. If you set the Enable Flag to 'No' the value is never valid.

When you add or change QuickCode Values you should sign on again to use the new value.

QuickCodes

Step 32 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 = ' ... '

 Formula

Define Compensation and Benefits for Information

See: Overview of Elements: The Building Blocks of Pay and Benefits, *(Oracle Payroll or Oracle Human Resources User's Guide)*

Step 33 Define Elements and Input Values

Elements are the basic components of compensation types, benefits and benefit plans, accrual plans and deductions. You can also use elements to represent tangible items distributed to employees, such as tools or safety equipment. When you install Oracle HRMS you will automatically have elements defined for:

- Regular Salary
- Regular Wages

At sites including Oracle Payroll, other predefined elements are installed to represent the earnings and deductions that process in the payroll run.

At sites without Oracle Payroll, you can define any number of additional elements to hold information about benefits, compensation types and other items.

 Element

Suggestion: Before you start defining an element you should have made all of your decisions about the definition and the rules of eligibility.

For each element you can:

- define up to 15 input values
- 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.
- set validation options for each value
  - a fixed value
  - a value range
  - a list of values using QuickCodes
  - a formula
- set *Hot* and *Cold* Defaulting Rules
- use the Balance Feed window to modify the individual balances that an element will feed.

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

You cannot enter an element for an employee before the start date of the element.

Step 34 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.


Element Link

- A *Standard* link will automatically start entries for all eligible employees. This will happen as soon as the employee assignment matches the link.

- If the link is not standard, employees are eligible but you must enter elements manually or through a batch process.
- An employee cannot be eligible for an element in two different ways at the same time. For example, you cannot define a link between Salary and Job and then define another link between Salary and Grade.

Note: You can date effectively end a set of element links and define a new set of links which take effect the next day. You cannot enter an element for an employee before the start date of the element link.

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

Define Earnings and Deductions for Payroll Processing

Note: 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.

See: Earnings and Deductions Overview, *Oracle Payroll User's Guide*

Warning: If you have installed Oracle Payroll you must use the *Earnings* and *Deductions* windows to initiate all of your earnings and deductions. This will simplify your setup for payroll processing.

Step 35 Define Additional Tax Categories

When you install Oracle HRMS you are given a predefined set of classifications and categories for elements.

Note: You cannot change or add to the list of classifications, but you can add more tax categories if you need these.

Categories determine the specific processing and tax rules that are applied to each element. These are essential for accurate payroll processing. You can add further tax categories by using the Quickcodes window to enter additional categories for these Quickcode types:

- US_EARNINGS
- US_IMPUTED_EARNINGS
- US_INVOLUNTARY_DEDUCTIONS
- US_PAYMENT

- US_PRE_TAX_DEDUCTIONS
- US_SUPPLEMENTAL_EARNINGS
- US_VOLUNTARY_DEDUCTIONS

QuickCodes

Suggestion: Use Start and End Dates to control when values appear in a list. With DateTrack you change your effective date to view and update information at any point in time. If you set the Enable Flag to 'No' the value is never valid.

Note: When you add or change QuickCode Values you should sign on again to use the new value.

Step 36 Define or Customize User Tables

See: User–Defined Tables in Oracle HRMS, *(Oracle Human Resources User's Guide)*

With Oracle HRMS you can set up any number of 'User–Defined Tables' to hold additional enterprise–level information in a tabular format. You can access this information using the *GET_TABLE_VALUE* function in any fastformula.

When you install the system you should find the following predefined tables ready to receive values:

- Company Work Schedules
- GTL Premiums
- Shift Differentials
- Wage Rates

Table Values

Additionally, you can set up and use your own table structures, with rows and columns. For example, you may want to initiate a deduction with an amount rule based on your own table of values.

A user–defined table is a 'matrix' of columns that hold different values for the same row. You can define exact row values or an inclusive range of values.

Table Structure

- Define table with *Match* or *Range* comparison for row values

- Enter row values
- Define column headings

Attention: You do not need to define additional tables for benefit plans with coverage levels that determine employee and employer contributions. When you install Oracle HRMS you will find a Benefits Table is already set up to hold information on benefit plans with coverage levels.

Step 37 Initiate Earnings

See: Initiating Earnings and Non–payroll Payments: *(Oracle Payroll User's Guide)*

A number of predefined earnings types are provided when you install Oracle Payroll. To activate the following predefined earnings you need only define links for them:

- Company Car
- GTL Imputed Income
- Overtime
- Regular Salary
- Regular Wages
- Shift Pay

See: Reviewing Earnings and Deductions, *(Oracle Payroll User's Guide)*

For any other type of earnings or non–payroll payment you can activate the types you require.

Attention: When you activate a type of earnings the system generates all the elements, input values, balance feeds, formulas and result rules you will need to process that type of earnings. You can customize any of these definitions to allow additional data capture or to modify the default calculations.

Earnings

You cannot enter history of an earnings type before the start date of that earnings type. You must set your effective date to allow you to load employee history.

Suggestion: Consider using a fixed date as a default for your initial setup, for example, 01–JAN–1901. This will simplify your data–entry.

Step 38 Initiate Deductions

See: Initiating Non–Tax Deductions: *(Oracle Payroll User's Guide)*

All Federal, State and Local Tax Levies are provided when you install Oracle Payroll. Oracle Corporation has an agreement with Vertex Inc. to provide and maintain this data. To activate these deductions you need only define links for the following elements:

- VERTEX
- Workers Compensation

A number of predefined non–tax deductions are also provided with Oracle Payroll. To activate the following predefined deductions you need only define links for them:

- Child Support
- Creditor Garnishment

For any other deduction you can activate the types you require.

Attention: You can customize any of the definitions generated by the system.

Deduction

You cannot enter history of a deduction for an employee before the start date of that deduction. You must set your effective date to allow you to load employee history.

Suggestion: Consider using a fixed date as a default for your initial setup, for example, 01–JAN–1901. This will simplify your data–entry.

Step 39 Define Element Links

Before you can enter any of your earnings or deductions for an employee you must define your element links.

Element Link

Attention: You must remember to define links for the predefined elements before you can use them with Oracle HRMS:

- Regular Salary and Regular Salary Special Inputs
- Regular Wages and Regular Wages Special Inputs
- Overtime
- Shift Pay

- GTL Imputed Income
- Company Car and Company Car Special Inputs
- VERTEX
- Workers Compensation
- Child Support and Child Support Special Inputs
- Creditor Garnishment and Creditor Garnishment Special Inputs
- Sick Pay
- Vacation Pay

Step 40 Setup Additional Taxability Rules for Tax Categories

See: Taxability Rules for Earnings and Deductions Categories: *(Oracle Payroll User's Guide)*

Oracle Payroll comes with the current rules for the federal and state–level taxability already in place for regular earnings and deductions. You can update these rules as necessary.

> **Note:** The system does not come with rules in place regarding the inclusion of supplemental and imputed earnings categories in states' payroll exposure.

Taxability Rules

- Use this window to maintain and review the taxability of supplemental and imputed earnings, and pre–tax deductions at the federal and state level.
- You also enter rules regarding the inclusion of categories of supplemental and imputed earnings in a state's payroll exposure for Workers Compensation.
- Enter all the rules you need for any additional Tax Categories you have setup.

See: Workers Compensation, *(Oracle Payroll User's Guide)*

Customize The Generated Definitions

When you initiate a type of earnings or deduction Oracle Payroll will generate all the elements, input values, balance feeds, formulas and result rules you will need to process that type of earnings or deduction in a payroll run. Additionally, it also associates a Skip Rule (formula) with the elements generated.

You can customize any of these definitions to meet your specific business needs.

See: Defining an Element to Hold Information, (*Oracle Human Resources User's Guide*)

Step 41 Define or Customize Payroll Balances

Attention: Oracle Payroll has many predefined balances installed with the system. To protect the integrity of the payroll processes you cannot change any of these balances.

You can customize any generated balance to change the feeds. A payroll balance has 'feeds' and 'dimensions'. You can modify the feeds and also add your own dimensions.

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.

Balance

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

Step 42 Write or Customize Payroll Formulas

See: Oracle FastFormula Overview: *Oracle Human Resources User's Guide.*

You can edit any of the generated 'Oracle Payroll' formulas to change the default calculations. You can also write your own formulas if you have special calculations that are very different from the defaults.

Formula

Warning: Remember that formula definitions are also 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.

Step 43 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.

When you activate any earnings or deduction type the system will generate the formula results and the rules for each result. If you customize the formula you may also have to customize the results.


Formula Result Rules

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

Note: 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.

Salary Administration

See: Salary Administration and Performance Reviews, *(Oracle Payroll or Oracle Human Resources User's Guide)*

You can choose to administer 'Salary' as a special type of entry using the Salary Administration window to make use of your Grade Rate and other definitions.

Note: If you want to use the predefined earnings types of Regular Salary and Regular Wages to administer 'salary' you must define element links before you can enter salary values.

Step 44 Define Proposal Reasons and Performance Ratings

You can associate salary administration with an employee evaluation and performance review process. You enter this information in the Salary Administration window.

- Define QuickCode values for PROPOSAL_REASON
- Define QuickCode values for PERFORMANCE_RATING


QuickCodes

Step 45 Activate Salary

Attention: You only need to perform this step if you decide not to use the predefined Regular Salary or Regular Wages types of earnings.

If you have installed Oracle Payroll then you can activate your Salary definition using the Earnings window.

Earnings

If you have not installed Oracle Payroll then you can activate your Salary definition using the Element window.

Element

Step 46 Define Salary Bases

A Salary Basis defines a relationship between the input value of a salary element and a grade rate. It also establishes the period for which a salary is quoted.

Salary Basis

- You can have any number of salary administration groups with different administration bases.
- Use meaningful names, such as *Salaried* or *Waged*.
- You can record Grade Rates at one frequency and salary amounts at another. For example you might want to enter employee salary as an annual amount but record rates as monthly values.

Step 47 Define Element Links for Salary

You can use different elements to administer salary for different groups of employees, for example 'hourly–paid' and 'salaried'. You define the eligibility rules for your different salary elements to make sure the right element is always used.

You must enter the salary basis in the *Assignment* window for an employee before you can use the Salary window to enter the value.

Note: You cannot enter a salary for an employee who is not eligible for the salary element.

Element Link

- Define links for your predefined Regular Salary or Regular Wages elements.
- If appropriate enter costing information for each link.

Warning: Do not use the salary basis to define links for salary unless you want to end the salary entry on a change of salary basis.

Do not use the Standard checkbox for these links. It is unlikely that you will have a default salary value for all employees.

Benefits Administration

See: Benefits Administration Overview, *(Oracle Human Resources User's Guide)*

Note: When you have completed the setup of COBRA–eligible plans you enter information about dependents of employees in the Contact window.

Oracle HRMS provides standard letters for COBRA notification and termination. You can modify these letters using Oracle Reports or you can define your own MSWord letters.

See: Generating a Microsoft Word Letter: *(Oracle Human Resources User's Guide)*

Step 48 Define QuickCode Values

QuickCode Values

There are four predefined values for the levels of dependent coverage for which you can define benefit contributions. You can modify these values or add any others that you may need.

- Default values for US_BENEFIT_COVERAGE are:
  - Employee Only
  - Employee and Children
  - Employee and Family
  - Employee and Spouse

There are two predefined values for COBRA termination reasons. Define any additional reasons for termination of continued coverage.

- Default values for US_COBRA_TERM_REASON are:
  - End of Coverage

- Non–Payment

There are five predefined values for COBRA status.

- Default values for US_COBRA_STATUS are:
  - Awaiting Notification
  - Elected
  - Notified
  - Rejected
  - Terminated

Step 49 Define Benefit Carriers

You define your Benefit Carriers as external organizations with a classification of 'Benefits Carrier'.

 Organization

Step 50 Define Benefit Plans

If you have installed Oracle Payroll then you can activate your benefit plans using the *Deductions* window.

> **Note:** If you have already implemented 'Benefits Administration' using Oracle HR before you install Oracle Payroll you should update your definitions using the *Element* window to make sure your definitions are suitable for calculating payroll deductions.

 Deductions

If you have not installed Oracle Payroll then you can activate your benefit plans using the *Element* window.

 Element

See: Defining an Element for a Medical, Dental or Vision Benefit Plan, *(Oracle Human Resources User's Guide)*

Step 51 Define Benefit Coverages and Default Contributions

 Benefit Contributions

See: Establishing Health Care Benefit Coverages and Default Contributions, *(Oracle Human Resources User's Guide)*

Step 52 Define Benefit Plan Links

Define eligibility rules for each of your benefit plans.

> **Note:** You cannot enroll employees in a benefit plan if they are not eligible for that plan.

Element Link

Element Sets

Step 53 Define Element Sets

In Oracle HRMS you can define a set of elements:

- to restrict access to elements using *Form Customization*
- to distribute costs across a *Distribution Set* of elements
- to process a restricted set in a *Payroll Run*

Element and Distribution Set

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*.

See:

Person Types,
Restricting the Data Displayed in a Window,
(Oracle Payroll or Oracle Human Resources User's Guide)

Step 54 Define Person Types

Oracle HRMS lets you define your own names to identify the 'types' of people in your system. These include all types of employees, applicants and contacts, as well as current and 'ex–' types.

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

A number of System Types have been predefined. These are:

User Name	System Name	Default
Applicant	Applicant	Yes
Applicant and Ex–applicant	Applicant and Ex–applicant	Yes
Contact	External	Yes
Employee	Employee	Yes
Employee and Applicant	Employee and Applicant	Yes
Ex–applicant	Ex–applicant	Yes
Ex–employee	Ex–employee	Yes
Ex–employee and Applicant	Ex–employee and Applicant	Yes
External	External	No

Table 2 – 2

Person Types

You can change these default names or define any number of new user types. For example, you might want to use Person Type to identify employees who are on a fixed term contract, or you might want to record Special Information for dependants of employees who are a special category of *External* Person Type.

Step 55 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.

Five employee user statuses are predefined. These are:

User Name	System Name	Default
Active Assignment	Active Assignment	Yes
End	End	Yes
Suspend Assignment	Suspend Assignment	Yes
Terminate Assignment	Terminate Assignment	Yes
Terminate Process Assignment	Terminate Assignment	No

Table 2 – 3

For each system status, you define as many user statuses as you require. These user statuses help you track the current employment circumstances of all your employees. You can also define secondary user statuses having no associated system statuses. You can use these for reporting purposes.

Assignment Statuses

Note: The User Statuses you define here will provide the list of values for Status in the Assignment window for an employee. If you want to change any of the seeded default values you must overtype the User Name.

Special Personal Information

See: Entering Special Information,
(Oracle Payroll or Oracle Human Resources User's Guide)

Step 56 Define 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

To take best advantage of the flexibility of Oracle HRMS for recording and reporting special personal information in your enterprise the implementation sequence which you follow is:

- Design your Personal Analysis Flexfield Structures.
- Define Flexfield Value Sets.
- Define Personal Analysis Flexfield Segments.
- Define Personal Analysis Flexfield Segment Values.
- Define your Personal Analysis Flexfield Cross–Validation Rules.
- Define your Personal Analysis Flexfield Aliases.
- Freeze and Compile your Personal Analysis Flexfield Structures.

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

The sub–steps that follow show you how to perform all these steps.

1. **Design your Personal Analysis Flexfield Structures**

   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.

2. **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.

   Define Value Set

3. **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.

    Define Key Flexfield Segments

4. **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.

    Define Segment Values

5. **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.

    Define Cross–Validation Rules

6. **Define Personal Analysis Flexfield Aliases**

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

    Define Shorthand Aliases

7. **Freeze and Compile Your Personal Analysis Flexfield Structure**

   You are now ready to freeze your flexfield definition. Navigate to the Define Key Flexfield Segments window. Enter Yes in the Freeze Flexfield Definition field and save your changes. Oracle Human Resource Management Systems now freezes and compiles your

Personal Analysis Flexfield definition. Compiling the flexfield definition enables the flexfield window with the defaults, values and rules that you have defined.

 Define Key Flexfield Segments

Step 57 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.

 Special Information Types

- 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 illnesses.

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.

Required Information for Employees

Attention: The following employee information is required for Government Mandated Reports:

- Gender
- Ethnic Origin
- Veteran Status

– Related special information for OSHA and ADA.

Attention: The following employee information is required for payroll processing:

– Primary Address

– Work Location and Payroll

– GRE / Legal Entity

– Employment Category and Job.

– Salary Information

– W–4 Information

Note: Additionally, you may need to enter exception information for the following overrides:

– Reporting Establishment Override

– WC Override Code

– Work Schedule Override

See: The Recruitment Cycle,
(Oracle Human Resources User's Guide)

Step 58 Define Assignment Statuses for Applicants

See: Tracking Applicant Progress,
(Oracle Human Resources User's Guide)

Assignment Statuses for applicants let you 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*.

Four system statuses for applicants have been predefined. These are:

User Name	System Name	Default
Accepted	Accepted	Yes
Active Application	Active Application	Yes
Offer	Offer	Yes
Terminate Application	Terminate Application	Yes

Table 2 – 4

Assignment Statuses

- You can change the user names of the predefined statuses
- You can define as many new user statuses as you require to describe the progress of an applicant in your enterprise.

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

Implementation Steps: Career Management

Note: Oracle Government Human Resources customers should skip these steps concerning Career Management and go to the next section.

If you are developing the competence approach as part of your performance management system, you must set up your methods of measurement, create your competencies and create your assessment and appraisal templates.

See: Introducing Career Management Activities, *(Oracle Human Resources User's Guide)*

You must also perform other implementation activities, such as configuring Oracle Workflow.

See: *HRDA Configuration Guide*.

Attention: This software should not be used as the sole method of assessment for making judgements about hiring, performance or deployment. Your company may be held liable if you rely on incorrect computer data or computerized rules to make such judgements.

It is the customer's responsibility to take all appropriate measures to comply with the Data Protection and Privacy laws of the countries in which they operate.

All personal information that you store or use with this software must be up to date, accurate and relevant. You should confirm the details of the restrictions that apply to the computerized storage and use of personal information with your own legal department or representative.

Methods of Measurement and Creating Competencies

Step 59 Create Rating Scales

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

 Rating Scales

See: *General Rating Scales*, *(Oracle Human Resources User's Guide)*

Step 60 Create Competencies

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

 Competencies

See: *Competencies,*
(Oracle Human Resources User's Guide)

Step 61 Group Competencies

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

1. **Create Competence Types**

   Create the competence types you require using the QuickCode COMPETENCE_TYPE.

    QuickCodes

   See: *Adding QuickCode Values,*
   (Oracle Human Resources User's Guide)

2. **Group Competencies into Types**

    Competence Types

   See: *Competence Types and Groupings,*
   (Oracle Human Resources User's Guide)

Competence Requirements

Step 62 Define Competence Requirements

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

1. **Define Your Competence Requirements**

   Create your competence requirements to meet the needs of your enterprise.

    Competence Requirements

   See: *Competence Requirements,*
   (Oracle Human Resources User's Guide)

2. **Enter Work Choices for a Job**

   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

See: *Entering Work Choices for a Job or Position, (Oracle Human Resources User's Guide)*

3. **Enter Work Choices for a 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

   See: *Entering Work Choices for a Job or Position, (Oracle Human Resources User's Guide)*

Define Functionss to Implement the Competence Approach (Oracle Training Administration)

Step 63 Define Functions to Implement the Competence Approach (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.

 Form Functions

See: *Defining Functions to Implement the Competence Approach, (Oracle Human Resources User's Guide)*

Qualification Types and Establishments

Step 64 Create Qualification Types

You can enter all the qualification types your enterprise recognises.

 Qualification Types

See: *Creating Qualification Types, (Oracle Human Resources User's Guide)*

Step 65 Create Schools and Colleges

You need to create schools and colleges that deliver the qualifications your enterprise recognises. 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.


Schools and Colleges

See: *Creating Schools and Colleges,*
(Oracle Human Resources User's Guide)

Assessment and Appraisal

Step 66 Create Templates for Competence–Based Assessments

You can create assessment templates for all the different assessments your enterprise performs to enable users to perform assessments using the web.


Assessment Template

See: *Competence–Based Assessments,*
(Oracle Human Resources User's Guide)

Step 67 Create Appraisal Questionnaire Pages

See: *Appraisal Templates and Questionnaire Pages,*
(Oracle Human Resources User's Guide)

Oracle provides you with an easy to use method of designing and formatting your own questionnaire pages for the appraisal using the Web. You can identify the questions to use in the appraisal, and the number, size and type of boxes in which to record the answers. You can create as many different appraisal questionnaire pages as your enterprise requires.


Attention: You create your appraisal questionnaire pages using the Web, and not using Oracle forms.

The implementation sequence which you follow is:

1. **Create the Questionnaire**

Create the questionnaire which contains the categories, sub–categories, sub–headings, questions and answer layouts.

See: *Creating the Questionnaire*, *(Oracle Human Resources User's Guide)*

2. **Creating Categories**

   Create the heading text that you want to appear on your appraisal questionnaire.

   See: *Creating Categories*, *(Oracle Human Resources User's Guide)*

3. **Assembling the Categories Within the Questionnaire**

   Assemble the categories you just created within the questionnaire. You can update the questionnaire, if required.

   See: *Assembling the Categories Within the Questionnaire*, *(Oracle Human Resources User's Guide)*

4. **Define Answer Layout and Create Questions**

   Name your questions and to define how you want the answers to questions to be recorded. For example, you can create text answer boxes, checkboxes, and such. Once you have named your questions and defined your answer layout, you can create your questions.

   See: *Defining Answer Layout and Creating Questions*, *(Oracle Human Resources User's Guide)*

Step 68 Create Templates for Appraisals

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.

 Appraisal Template

See: *Creating or Changing an Appraisal Templates*, *(Oracle Human Resources User's Guide)*

Implementation Steps: New Hire Reporting

Step 69 Check GRE Federal and State Identification Numbers

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


Organization

See: *Entering Tax Rules for GREs, (Oracle Human Resources User's Guide).*

Step 70 Enter the GRE Contact Person


Organization

See: *Entering a New Hire Report Contact for a GRE, (Oracle Human Resources User's Guide).*

Step 71 Enter New Hire Information for Every Employee

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


Person

See: *Entering New Hire Report Information for Employees, (Oracle Human Resources User's Guide).*

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


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

Do this manually, or as part of your data loading.

Workers Compensation

See: Workers Compensation, *(Oracle Payroll User's Guide)*

If you have not installed Oracle Payroll you should skip the section on Workers Compensation.

Workers Compensation programs are legislated in each state to provide employees with insurance coverage for work–related injuries. The following steps cover all aspects of setting up Workers Compensation in Oracle Payroll.

> **Note:** Some of these steps overlap with the setup of Work Structures and Compensation and Benefits.

Step 72 Define Workers Compensation Insurance Carrier

Define both state and private insurance carriers as external organizations.

 Organization

- Enter and enable the classification of Workers Compensation Carrier.
  - This will provide the list of values when you define the Government Reporting Entity.

Step 73 Enter Insurance Carriers for Government Reporting Entities

When you define your GREs you can enter the insurance carrier in the State Tax Rules window.

 Organization

- For each state enter the name of the Workers Compensation Carrier and any rates that may affect calculation of the GRE's liability.

Step 74 Check State Rules for Imputed and Supplemental Earnings

For each state in which you are liable for WC payments check the categories for Imputed and Supplemental Earnings that represent earnings included in employee's payroll exposure.

 Taxabiliity Rules

Note: You can define additional categories using the QuickCode type US_SUPPLEMENTAL_EARNINGS or US_IMPUTED_EARNINGS.

See: *Adding QuickCode Values, Oracle Human Resources User's Guide.*

Step 75 Define State Codes and Rates

For each carrier **or** carrier and location in the state, enter the state work classification codes and associated rates.

 WC Codes and Rates

Step 76 Enter Codes for Jobs

For every state where you need to calculate WC you must enter the appropriate codes for your jobs. You can also enter the Executive Weekly Maximum, if one exists for this state, and the state rules regarding inclusion of overtime earnings in the payroll exposure. You can also enter any standard surcharges for the state.

 Workers Compensation

- Include the overtime categories that contribute to WC payroll exposure

Attention: When you assign an employee to a job the system will default the WC code for that job. You can override the default code in the Miscellaneous region of the Employee Assignment.

Note: This override information is datetracked.

Step 77 Define Element Link for Workers Compensation Element

With Oracle Payroll there is a predefined element and formula to calculate Workers Compensation. To activate this element you need to define the rules for eligibility.

 Element Link

Absence Management/Accruals of Paid Time Off (PTO)

You can set up as many plans as you need to permit employees to accrue PTO each calendar year, to use for vacation or sick leave. Each

plan has the units of Hours or Days, and can have its own start rules and rules regarding length of service bands, accrual ceilings, and accrual carryovers.

Attention: The additional elements that you set up are used to record the information that is used in the calculation of Sick Pay and Vacation pay.

If you have installed Oracle Payroll, you must check that you have already linked the predefined elements for Sick Pay and Vacation to all employees. You should make these links 'standard' for all employees on a payroll.

Step 78 Define an Absence Element to Record Time Taken

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.

Element

- Use a Classification of 'Information'
- Termination Rule is 'Actual Termination'
- Input Value must be 'Hours' or 'Days'

Step 79 Define Element Links for the Absence Element

You define the eligibility rules for each absence element.

Element Link

Step 80 Define the Accrual Plan with an Absence Element

Accrual Plans

- Define the Accrual Plan and its start rule
- Associate your Absence Element with the Plan
- Define Accrual Bands
- Modify Net Calculation Rules
  - The system will insert the default rules for the associated absence element and for 'Carried Over' absence. You can include other element input values to add or subtract from net according to your specific business needs.

Attention: When you save your plan definition the system will automatically generate 3 additional Accrual Plan elements. These elements are used in the calculation of PTO.

- 'Accrual Plan'
- 'Carried Over'
- 'Residual'

The default effective start date for all of these is 01–JAN–0001.

Step 81 Define Element Links for the Accrual Elements

You define the eligibility rules for the 3 accrual plan elements.


Element Link

Step 82 Define Absence Reasons

If you plan to use absence types and you want to record absence reasons for each occurrence of an absence type, you must define the QuickCode values for ABSENCE_REASON.


QuickCodes

Step 83 Associate an Absence Type with the Absence Element


Attention: If you will always use PayMIX to enter the absence element as part of a timecard entry you may choose to omit this step.

If you decide to use the *Absence Detail* window you can enter more detailed information for each occurrence of an absence. To use this window you must first associate your Absence Element with an Absence Type.


Absence Attendance Type

- Select 'Increasing' to control the accumulation of the absence balance.
  - The system enters a positive value for the nonrecurring element when you enter an absence.

Human Resource Budgets

See: Human Resource Budgets,
(Oracle Human Resources User's Guide)

Step 84 Define QuickCode 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


QuickCode Values

Step 85 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.


Period Types

Step 86 Define Budgetary Calendars

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


Budgetary Calendar

- Once you define the calendar you cannot change the start date. Set the start date to let you enter any budget history information you want to enter.
- You cannot define years with an earlier start date than the start of the calendar.
- In each calendar you define as many years as you require. You do not create a new calendar for each year. You just add new periods to the calendar.

Step 87 Define Budgets

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


Budget

- Define a budget for any combination of organization, job, position or grade.
- Enter a budget value for every time period in your calendar.

  Actual values for each budget type for an assignment are entered in the Assignment Budget Values window.

Suggestion: Consider setting default assignment budget values for the Business Group. If you want accurate values you must make sure that assignments have budget values entered.

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.

Evaluation Systems

See: Evaluating Jobs And Positions,
(Oracle Human Resources User's Guide)

Step 88 Define Evaluation Types

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


QuickCodes

- Define the name of your evaluation system as a value for the QuickCode 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.

Step 89 Define Valid Grades for Jobs

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.


Job

- Query the job and select the Valid Grades button
- Enter and save the valid grades for each job. You can enter a single grade, or a set of grades.

Step 90 Define Valid Grades for Positions

Oracle HRMS lets you define Valid Grades for Positions. These definitions will be used to provide warning messages in the Assignment window when you enter Position and Grade information.

 Position

- Query the position and click on the Valid Grades button
- Enter and save the valid grades for each position. You can enter a single grade, or a set of grades.

Requirements Matching

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

See: Implementation Steps: Career Management: page 2 – 52.

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.

See: Defining Special Information Types
(Oracle Human Resources User's Guide)

For each type, you can choose whether also 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 allows 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.

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

Step 91 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.

 Job

- Query the job and choose the Requirements button
- Enter the combinations of attributes and details that are required for this job. The pop–up flexfield window will apply the defaults, values, and rules that you have defined for each special information type.

Step 92 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 attributs may be Essential or Desirable. The requirements are based on the same personal analysis flexfield structures you have defined for special personal information.


Position

- Query the position and click on the Requirements button
- Enter the combinations of attributes and details which are required for this position. The flexfield window will apply the defaults, values and rules that you have defined for each special information type.

Career and Succession Planning

See: Career and Succession Planning,
(Oracle Human Resources User's Guide)

The flexibility provided by Oracle Human Resources means that 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.

Step 93 Modelling Career and Succession Planning 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.

See: *Modelling Career and Succession Planning Based on Jobs,*
Oracle Human Resources User's Guide

1. **Create the career paths 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.

   
   Career Path Names
   and Map Career Paths

   See: *Defining Career Paths*,
   Oracle Human Resources User's Guide

   
   Attention: For AAP–Workforce Analysis reporting you use the career path functionality to build the *lines of progression* for the jobs included in your AAP plans.

2. **Enter Work Choices**

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

   
   Work Choices

   See: *Entering Work Choices for a Job or Position*,
   Oracle Human Resources User's Guide

   See: *Entering Work Choices*,
   Oracle Human Resources User's Guide

3. **Create Your Appraisal Questionnaire (Line Manager Direct Access users only)**

   If you are using the Web–based Line Manager Direct Access, you can select a career path for a person as part of the appraisal process.

   See: *Creatring Your Appraisal Questionnaire Web Pages*,
   Oracle Human Resources User's Guide

4. **Use Attachments or Special Information Types**

   If you are not doing step 3, consider holding succession plan information against people as attachments or using a special information type.

   See: *Defining Special Information Types*,
   Oracle Human Resources User's Guide

Step 94 Modelling Career and Succession Planning 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.

See: *Modelling Career and Succession Planning Based on Jobs*, *Oracle Human Resources User's Guide*

1. **Create position hierarchies**

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

    Position Hierarchy

   Use the Position Hierarchy form.

   See: Creating Position Hierarchies, *Oracle Human Resources User's Guide*

2. **Enter Work Choices**

   Optionally, enter work choices to help identify a person's career plan.

    Work Choices

   See: *Entering Work Choices for a Job or Position*, *Oracle Human Resources User's Guide*

   See: *Entering Work Choices*, *Oracle Human Resources User's Guide*

3. **Use Succession {Planning (Line Manager Direct Access users only)**

   If you are using the Web–based Line Manager Direct Access, use the Succession Planning option to record one or more next positions for each employee.

4. **Use Attachments or Special Information Types**

   If you are not doing step 3, consider holding succession plan information against people as attachments or using a special information type.

   See: *Defining Special Information Types*, *Oracle Human Resources User's Guide*

Implementation Steps: Control

See: Predefined and User Defined Reports,
(Oracle Payroll or Oracle Human Resources User's Guide)

See:

Overview of Government–mandated Reporting, *Oracle Human Resources User's Guide*

Predefined and User Defined Reports, *Oracle Human Resources User's Guide*

Define Reports

Step 95 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.

See also: Overview of Printers and Printing,
Oracle Applications System Administrator's Guide.

Step 96 Register Reports as Concurrent Programs with SRS

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.


Concurrent Programs

Step 97 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.


Request Set

Standard Letter Generation

You can use standard letters in HRMS to help you to manage your enterprise's recruitment or enrollments. 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:

- Online using Application Data Export (ADE)
- Concurrent Processing

Before you start to set up your standard letters, you need to establish which method best suits your needs.

See: Setting Up Standard Letters, *Oracle Human Resources User's Guide*

Step 98 Create Standard Letters for use with Concurrent Processing

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

- Using word processors
- Using Oracle reports

You can use any word processor to produce standard letters from Oracle HRMS. If you use a word processor, you can submit a concurrent request in the Letter Request window to generate the mail merge file. When the concurrent request is complete, you can use your word processor's mail merge facilities to create the merged letters.

See: Using a Word Processor, *Oracle Human Resources User's Guide*

As an alternative to using a word processor to produce standard letters, you can use the Standard Letter and Label features of Oracle Reports. Use this method if you do not want use word processors to print your letters (or if you do not have word processors).

See: Using Oracle Reports, *Oracle Human Resources User's Guide*

Concurrent Processing – for use with Word Processors

The sub–steps below describe how to set up standard letters using concurrent processing.

1. **Plan**

   One of the first tasks you need to perform is to identify which data you want to extract from the database to include in your standard letters. You need to identify the select statements to use to provide you with the the data as the *content* of your letters. Oracle HRMS

supplies you with SQL*Plus scripts as templates to help you do this. You also need to identify the text that you want to include as the *body* of your letters.

The next decision you need to make is to decide whether or not you want to associate your standard letters with student enrollment or applicant assignment statuses.

See: Planning (for, MultiMate or WordPerfect), *Oracle Human Resources User's Guide*

2. **Write SQL*Plus script**

   The next step is to write the script that extracts the event and enrollment details from the system. This SQL*Plus file must use the correct delimiter for your word processor to recognize in the mail merge routine.

   See:

   Writing a SQL*Plus Script for MultiMate or WordPerfect) *Oracle Human Resources User's Guide*

   Template SQL*Plus Script PERWPMUS, *Oracle Human Resources User's Guide*

3. **Register SQL*Plus script**

   After you have written the file to extract the data, you must register it so that it can be run as a concurrent program. Name the files PERWP***. You must use this prefix for the system to recognise it as a type of letter.

    Concurrent Programs

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

4. **Linking the SQL*Plus script with a letter**

   The next step is to link your SQL*Plus script to one or more applicant or enrollment statuses.

    Letter

   See: Linking the SQL*Plus Script With a Letter for MultiMate or WordPerfect) *Oracle Human Resources User's Guide*

5. **Write skeleton letter**

   You now need to write a skeleton letter using your word processor.

See: Writing a Skeleton Letter for MultiMate or WordPerfect, *Oracle Human Resources User's Guide*

6. **Request Letter**

   You now run the SQL*Plus script to extract the data from the database. You activate the SQL*Plus script through the Letter Request window.

   
   Request Letter

   See: Requesting Letters for MultiMate or WordPerfect, *Oracle Human Resources User's Guide*

7. **Merge Data File with standard letter**

   You need to merge the data in the data file with your skeleton letters to create your standard letters.

   See: Merging the Data File With the Standard Letter for MultiMate or WordPerfect, *Oracle Human Resources User's Guide*

Concurrent Processing – for use with Oracle Reports

The sub–steps below describe how to set up standard letters using concurrent processing.

1. **Plan Content and Layout**

   One of the first tasks you need to perform is to identify which data you want to extract from the database to include in your standard letters. You need to identify the select statements to use to provide you with the the data as the *content* of your letters.

   The next decision you need to make is to decide whether or not you want to associate your standard letters with student enrollment or applicant assignment statuses.

   See: Planning the Content and Layout, *Oracle Human Resources User's Guide*

2. **Write Report**

   The next step is to write the report that extracts the event and enrollment details from the system. You also need to write your skeleton letter text and select statements.

   See: Writing the Report, *Oracle Human Resources User's Guide*

3. **Register Report**

   After you have written the report, you must register it so that it can be run as a concurrent program. Name it PERWP***. You must use this prefix for the system to recognise it.

   
   Concurrent Programs

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

4. **Linking the report with a letter**

   The next step is to link your report to one or more applicant or enrollment statuses.

    Letter

   See: Linking the Report With a Letter, *Oracle Human Resources User's Guide*

5. **Running the Report**

   You now run the report to extract the data from the database. You run the report by creating a pending letter request in the Letter Request window.

    Request Letter

   See: Running the Report, *Oracle Human Resources User's Guide*

Step 99 Create Standard Letters Online for use with Application Data Export (ADE)

You can generate your standard letters online, using ADE. ADE comes with its own set of documentation and online help.

> **Note:** We provide online help only for this version of ADE. Refer to the online help to help you create standard letters online for use with ADE.

Step 100 Create Standard Letters Online for use with Microsoft Word

If you use Microsoft Word as your word processor, not only can you use the concurrent processing method to produce your standard letters, but you can also generate letters online.

You can use either of two methods:

- Generate Microsoft Word letters using Object Linking and Embedding (OLE)
- Application Data Export (ADE)

Attention: In future releases of Oracle HRMS, generating Microsoft Word letters using Object Linking and Embedding (OLE) will be replaced by ADE.

ADE comes with its own set of documentation and online help.

Note: We provide online help only for this version of ADE. Refer to the online help to enable you to set up ADE.

If you are setting up standard letters using the concurrent processing method, follow the same sequence as for MultiMate or WordPerfect.

See Flowchart for Setting Up Standard Letters Using MultiMate or WordPerfect, *Oracle Human Resources User's Guide*

If you are generating Microsoft Word letters using Object Linking and Embedding (OLE), see Writing a SQL*Plus Script for Microsoft Wor , *Oracle Human Resources User's Guide*

Define User Security

See: Security in Oracle HRMS,
(Oracle Payroll or Oracle Human Resources User's Guide)

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.

- A security profile limits access to records and is based on the work structures you define in the system. These restrict access to data using organization hierarchies, position hierarchies, and payrolls.

  Note: You can use security profiles to restrict the data visible to people who use Oracle reporting tools to access the Oracle HRMS database, as well as those who use forms. To do this, you must create a new ORACLE User ID for each security profile.

- The Main Menu determines the forms a user can see in a Responsibility. Forms may be called in Query–Only mode which will allow a user to inquire and see information but will not allow any changes to be made.
- The HR:Query Only Mode user profile option restricts access to view–only for *all* HR and Payroll forms on a menu.
- Form Customization lets you restrict the types of information a user can access in a specific form or window.
- Task Flow lets you define the sequence of windows you want to use when performing specific tasks.

Step 101 Create New 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. For example, reporting users might use Oracle Discoverer to report on HRMS information.

Note: You need to inform Reporting Users of their Reporting Username and Password.

Step 102 Register ORACLE IDs

This is a task for your System Administrator.

After the DBA has created the ORACLE IDs you must register them with Application Object Library.


Oracle Usernames

- Enter the username and password of your ORACLE ID.
- Enter the logical database details, for example, Application Object Library, STANDARD.
- Enter Restricted in the Privilege field.

Step 103 Define Security Profiles

Change to your 'View–All' Responsibility to define restricted security profiles within a Business Group.


Security Profile

The security profile defines the employees AND the applicants you can see in a single responsibility. For example, when you define a profile to have access to the records of one department you can say whether this is for:

- employees to that department and *all* applicants, or
- applicants to that department and *all* employees, or
- employees and applicants to that department only

Step 104 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.


Submit a New Request

- Select *Generate Secure User.*
- In the Parameters window, enter the security profile you created for the ORACLE ID.

- 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.

Step 105 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.

Submit a New Request

When this process has completed successfully you can sign on to the system using the new username and responsibility.

Step 106 Define Form Customizations

Form Customization lets you restrict the types of information a user can access in a specific form or window.

The list of forms that you can customize is given in your User's Guide.

Note: You can define your own form titles for any form customization option. Remember that the user guides and the online help use the default form names to identify forms.

Form Customization

Attention: You can call the customized form in two ways:

- define a customized node in a task flow
- add the customization as an argument to the menu function which calls the form

  HR_CUSTOMIZATION = "*customization name*"

Step 107 Define Task Flow Nodes

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.


Define Task Flow Nodes


Suggestion: If you want to use a customized form as the first node in a task flow you should define this as a new node.

You could add both the task flow and the customization as arguments to a single menu function but this may be more difficult to maintain.

WORKFLOW_NAME = "*task flow name*"

Step 108 Define Task Flows

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


Task Flow


Suggestion: When defining the navigation buttons in your task flows:

– Don't have long sequences where you cannot go back to the previous step

– Use the same button names for the same nodes

– Use different button names for customized nodes

Note: Before you can include the task flow in a menu, you must define a new menu function with the following argument in the Argument field:

Step 109 Define Menu Functions

This is a task for your System Administrator.

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 forms.

Function Security helps you to control the menu options you make available to each responsibility. When you define a responsibility you can restrict the submenus or functions for that responsibility.

In Oracle HRMS a function can be:

- a form
- a form called in query–only mode
- a form called with a customization

- a form called with a task flow

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

Form Functions

Consider whether you want to define your own 'supermenus' to contain all of your task flow and customization 'functions' as well as the standard forms. The alternative is to define many menus.

Step 110 Define Menus

This is a task for your System Administrator.

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:

- HR View Medical

  This function causes the Medical Information alternative region to display in the People window.

- HR View Background

  This function causes the Background Information alternative region to display in the People window.

- HR View Rehire

  This function causes the Rehire Information alternative region to display in the People window.

- Salary Administration: Approve

  This function enables the user to approve salary proposals in the Salary Administration window and the Salary Management folder.

Menus

Warning: Oracle HRMS with DateTrack and task flows does not fully support multiple active forms. When you define a new menu for use with Oracle HRMS the top menu must include the following function as a separate menu option:

Step 111 Define Report Security Groups

This is a task for your System Administrator.

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 report group.


Report Groups

Step 112 Define Responsibilities

This is a task for your System Administrator.

Define a Responsibility to bring together all of your Security definitions:


Responsibilities

- Every responsibility has a unique Name.
- Select your customized Main Menu to restrict access to the forms.
- Select the Report Security Group for the Responsibility.
- Exclude any submenus or functions you want to restrict.

  Note: Before you can log on to Oracle HRMS using your new secure Responsibility you must run the Security List Maintenance process.

Step 113 Define Application User

This is a task for your System Administrator.

You should define every user of the system with a unique username and password. You can give the same responsibility to many different users, but any data changes will be identified by the Application Username.


Users

- To create a new User, enter a username and a password with the correct Start Date.

  Note: The first time you sign on as a new user the system will force you to select your own private password.

- Select the responsibilities you want this user to access. Use the Start and End Dates to control when each responsibility is available to the user.

Step 114 Define HR User Profile Options

This is a task for your System Administrator.

HR User Profile Options control some of the defaults which are used in the online system. You must define the following profile values for every new responsibility:

- HR:Security Profile
- HR:User Type

  Note: The HR:Security Profile option determines the Business Group for the responsibility. If you do not define this profile, the setup business group will be used as the default.

In addition to these profiles you may want to set up other defaults for groups of users or even for an individual user. For example, you may want to set the default for the DateTrack:Prompt to always prompt new users with their effective date.

 System Profile Values

- Select the new responsibility.
- Select the HR:Security Profile option and enter the name of your security profile.
- Select the HR:User Type option and enter the correct user type for this responsibility.
  - *HR User* if you have access to Oracle Human Resources but not Oracle Payroll functions.
  - *HR and Payroll User* if you have access to both Oracle Human Resources and Oracle Payroll functions.
  - *Payroll User* if you have access to Oracle Payroll but not Oracle Human Resources functions.

  Note: This option restricts the regions and fields that a user can use in the windows. For example, an HR User cannot see some of the payroll processing fields on the Element window. Also, if Oracle Payroll is installed, HR Users cannot assign employees to payrolls.

Define Audit Requirements

Step 115 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.

Note: The following tasks are the responsibility of your System Administrator.

Step 116 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.

Audit Installations

Step 117 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.

Define Audit Tables

- Query the Table you want to audit
- Enter the columns you want to audit for that table

Step 118 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.

Audit Groups

Step 119 Activate AuditTrail Update Tables Process

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

Submit a New Request

APPENDIX

A Technical Essays

This appendix contains essays on the following topics:

- How DateTrack Works
- How to Create and Modify DateTrack History Views
- The FastFormula Application Dictionary
- Extending Security in Oracle Human Resources
- Creating Control Totals for the Batch Element Entry Process
- APIs in Oracle Human Resources
- Balances in Oracle Payroll
- Balance Dimensions
- Initial Balance Loading for Oracle Payroll
- Including Balance Values in Reports
- US Legislative Balance Initialization
- US Dependents and Beneficiaries
- US Payroll Tax Subsystem
- PayMIX

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:

- CHANGE 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.

- OVERRIDE – 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 – 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 – This is the total delete. All records matching the key value, whatever their date stamps, are deleted.
- FUTURE CHANGE – 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 – 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 because the inserted record has its effective start date set to today (Effective Date) and its effective end date set to 31–DEC–4712.

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, you must add an extra two columns to every DateTracked table.

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.

Notice that when a DT record has not been deleted in the future (the normal case), then EFFECTIVE_END_DATE for the last row is set to 31–DEC–4712.

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

Table A – 1 Example of DateTracked Table Contents (Page 1 of 1)

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	18,200
02–DEC–1989	** no rows retrieved		

Table A – 2 Example of Query Results for a DateTracked Table (Page 1 of 1)

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 as Application Short Name%F where the appropriate short name is one of PER, PAY, SSP, or FF.

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-JAN-0001','DD-MON-YYYY') EFFECTIVE_START_DATE,
     TO_DATE('31-DEC-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
```

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 *Introduction to Oracle HRMS*.

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 allows 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.

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

What Happens When You Request DateTrack History

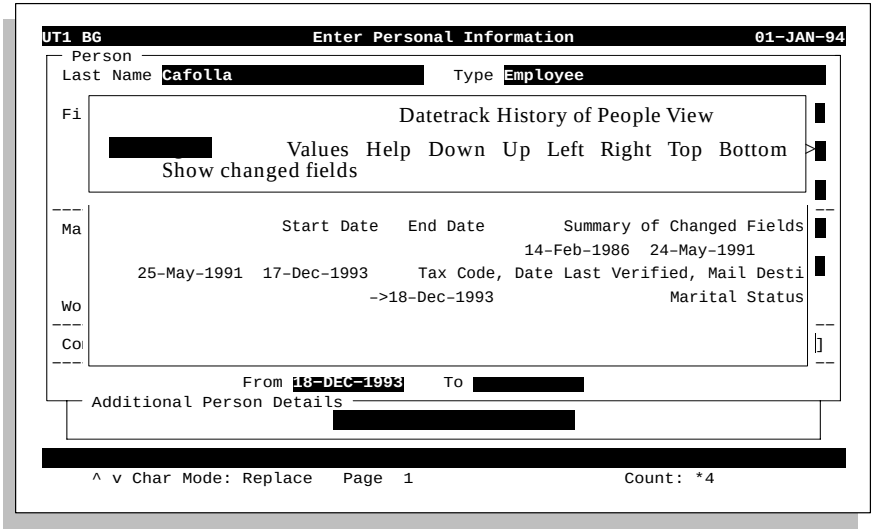
When you request DateTrack History from a DateTracked window, the window passes the name of the base table, the name of the unique id field, and the value of the unique id to DateTrack History.

Before the DateTrack History Change Field Summary window displays, the code checks whether a DateTrack History view exists for the base table. 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_people_f, the code looks for a view called per_people_d. If the view exists, the code uses it; otherwise it extracts information from the base table.

DateTrack History in Forms 2

If the information is extracted from a view, the title of the DateTrack History menu contains the word 'View', as shown in the following figure. The DateTrack History code modifies the column names of the table or view before presenting them in the menu. Underscores are replaced by spaces and the first letter of each word appears in upper case.

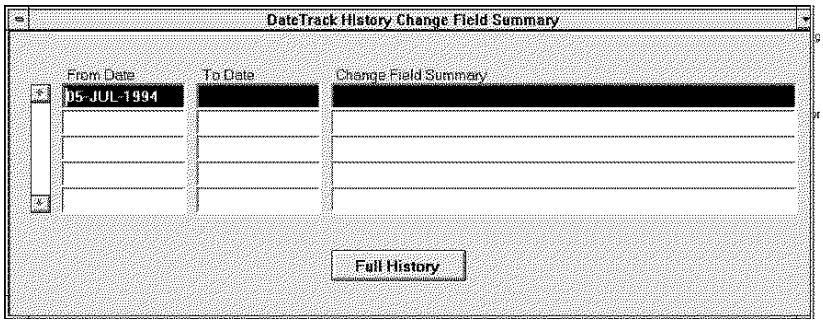
Figure A – 1
DateTrack History of People View



DateTrack History in Forms 4

When a view exists, information about the entity name and column prompts is read from the datetrack tables (DT_TITLE_PROMPTS, DT_DATE_PROMPTS and DT_COLUMN_PROMPTS). If the information cannot be found in these tables or the _f table is used, the information is extracted from the view / table 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.

Figure A – 2
Change Field Summary



Rules for Creating or Modifying DateTrack History Views

All DateTrack History views must adopt the naming convention described above. That is, they must have the same name as the corresponding base table, except that the suffix _f is replaced by _d.

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
```

List of DateTrack History Views

The supplied view creation scripts are as follows:

View Creation Script	*Based on*
pedtasgn.sql	PER_ASSIGNMENTS_F
pedtbbcf.sql	BEN_BENEFIT_CONTRIBUTIONS_F
pedtccbf.sql	PER_COBRA_COVERAGE_BENEFITS_F
pedtgrsp.sql	PER_GRADE_SPINES_F
pedtpepl.sql	PER_PEOPLE_F
pedtsppp.sql	PER_SPINAL_POINT_PLACEMENTS_F
pedtspst.sql	PER_SPINAL_POINT_STEPS_F
pydtbalf.sql	PAY_BALANCE_FEEDS_F
pydtetyp.sql	PAY_ELEMENT_TYPES_F
pydtexrt.sql	PAY_EXCHANGE_RATES_F
pydtfmrr.sql	PAY_FORMULA_RESULT_RULES_F
pydtgrdt.sql	PAY_GRADE_RULES_F
pydtpaym.sql	PAY_ORG_PAYMENT_METHODS_F
pydtpayr.sql	PAY_PAYROLLS_F
pydtppym.sql	PAY_PERSONAL_PAYMENT_METHODS_F
pydtstpr.sql	PAY_STATUS_PROCESSING_RULES_F
pydtucin.sql	PAY_USER_COLUMN_INSTANCES_F
pydtussrr.sql	PAY_USER_ROWS_F

The supplied views are as follows:

View Name	*View Script*
PAY_EXCHANGE_RATES_F	pydtexrt.sql
PAY_USER_COLUMN_INSTANCES_F	pydtucin.sql
PAY_USER_ROWS_F	pydtussrr.sql
PER_COBRA_COVERAGE_BENEFITS_F	pedtccbf.sql
PAY_FORMULA_RESULT_RULES_F	pydtfmrr.sql
PAY_BALANCE_FEEDS_F	pydtbalf.sql
BEN_BENEFIT_CONTRIBUTIONS_F	pedtbbcf.sql

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 may 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 on the QuickPick lists.

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 Oracle FastFormula User Guide (Appendix D of the *Oracle Human Resources Reference Manual*).

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;
/
```

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 your product user's guide, and to the setup steps in the *Oracle HRMS Implementation Guide.*

This essay does not repeat the definitions and description in your user and implementation guides. 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 profile via the **HR:Security Profile** profile option.

There are two types of security profile:

- unrestricted
- restricted

Restricted security profiles are available only to users of Oracle Human Resources and Oracle Payroll. 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
- 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 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)
PER_POSITIONS	Secure view of Positions table
PER_ALL_POSITIONS	Positions table
PER_VACANCIES	Secure view of Vacancies table
PER_ALL_VACANCIES	Vacancies table
PER_ASSIGNMENTS	Secure view of Assignments table, effective at session date
PER_ASSIGNMENTS_F	Secure view of Assignments table
PER_ASSIGNMENTS_X	Secure view of Assignments table, effective at system date
PER_ALL_ASSIGNMENTS	Unrestricted view of Assignments table, effective at session date
PER_ALL_ASSIGNMENTS_F	Assignments table
PER_PEOPLE	Secure view of Person table, effective at session date
PER_PEOPLE_F	Secure view of Person table
PER_PEOPLE_X	Secure view of Person table, effective at system date
PER_ALL_PEOPLE	Unrestricted view of Person table, effective at session date
PER_ALL_PEOPLE_F	Person table
PAY_PAYROLLS	Secure view of Payrolls table, effective at session date
PAY_PAYROLLS_F	Secure view of Payrolls table
PAY_PAYROLLS_X	Secure view of Payrolls table, effective at system date
PAY_ALL_PAYROLLS	Unrestricted view of Payrolls table, effective at session date
PAY_ALL_PAYROLLS_F	Payrolls table

Table A – 3 Secure Tables 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 HOA
WHERE DECODE(HR_SECURITY.VIEW_ALL, 'Y', 'Y',
           HR_SECURITY.SHOW_RECORD
           ('HR_ALL_ORGANIZATION_UNITS',HAO.ORGANIZATIONN_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 taken from the HR:Security Profile profile option.
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 current security_profile_id is set to zero, which corresponds to the View All security profile for the Setup Business Group. In this latter case, you are given unrestricted access to the data in the HRMS tables that are accessible by your Oracle User.

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 and resp_appl_id), passing the IDs of the user, responsibility and application for the sign–on session you want to simulate.

> **Note:** Neither FND_GLOBAL or HR_SECURITY are 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

Table A – 4

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)

If you are not setting up reporting users, you need not run ROLEGEN and SECGEN.

Refer to your Oracle HRMS user's guides for details of how to submit each process from the Submit Requests window. This section describes how the processes work.

ROLEGEN: Grant Secure Role Permission Process

A role is a set of permissions that can be granted to Oracle users or to other roles. Roles are granted to users by the SECGEN process (see below).

Before running SECGEN, you must run the ROLEGEN process. This dynamically grants select permissions on Oracle HRMS tables and views to the HR_REPORTING_USER role. This role must exist before you run ROLEGEN.

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 is run as part of the installation process, you do not need to manually run ROLEGEN. You only need to manually run ROLEGEN if you have applied a patch which adds new HRMS tables or views, and you need to make the tables or views available to reporting users.

ROLEGEN performs the following actions:

- creates public synonyms for HRMS tables and views
- 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

SECGEN is run 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. Although once SECGEN has been run for a particular security profile, you need nor rerun it even if you subsequently rerun ROLEGEN.

SECGEN is a PRO*C process with embedded SQL statements. It is initiated from the Submit Requests window.

LISTGEN – Create Security Lists Process

LISTGEN is run 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 run ROLEGEN. 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;
```

Creating Control Totals for the Batch Element Entry Process

Introduction

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 may be entered, or viewed on the *Batch Element Entry*, (BEE), window.

Setting up Control Totals

This is done in three parts:

1. Create a control total type in QuickCode Values under the type CONTROL_TYPE. (Refer to User Manual for details of setting up QuickCode values)
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.

Task 2 is the most complex and is elaborated below.

Creating the SQL Code

When customising the BEE control total validation code, the code should be added into the following procedure:

- Procedure: check_control
- Package: user_check
- File: pyusrchk.pkb

This procedure is delivered with a null statement in it which will need to be replaced with the appropriate control total validation code.

Parameters

The check_control procedure is executed during the batch validation phase of the MIX 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 MIX Batch Line screen.

For example, if you wished to check the total value of the first column of the lines you could achieve this by using 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;
```

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
- Understanding the Object Version Number (OVN)
- API Parameters
- API Features
- Flexfields with APIs
- Alternative APIs
- API Errors and Warnings
- Example pl/sql Batch Program
- WHO Columns and Oracle Alert
- API User Hooks
- Using APIs as Building Blocks
- Handling Object Version Numbers in Oracle Forms 4.5
- HRMS Table Locking Ladder

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 allows 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 webpages 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 will:

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 will:

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 updating 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 allow 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

Every published API has 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 which still list, the new object version number is passed out.

See:

Understanding the p_datetrack_update_mode control parameter

Understanding the p_datetrack_delete_mode control parameter

Handling Object Version Numbers in Forms 4.5

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.

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 the APIs which were included in the first version of Release 11, the scripts are located in the operating system directories $PER_TOP/admin/sql and $PAY_TOP/admin/sql. Refer to filenames such as pe???api.pkh, hr???api.pkh, and py???api.pkh.

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 which where not included in the first version of Release 11, may be provided in different operating system directories.

Oracle only supports the APIs listed in the Release new features documentation. That list is a reduced set of the server side code which matches all of the following three criteria:

- The database package name ends with "_API".
- The package header creation script filename conforms to pe???api.pkh, hr???api.pkh, or py???api.pkh naming standard.
- 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 by passed. Therefore, corrupting the data held within the Oracle HRMS application suite.

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 which may or may not be specified. Most parameters map onto a database column in the HR

schema. There are some control parameters which affect the processing logic which 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 4.5 code, then all parameters in the API call must be specified. You cannot make use of the pl/sql declared default values because the version of pl/sql in Forms 4.5 does not support this.

Default Parameters with Create Style APIs

For APIs which 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 which do not a 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, you should set the variable used to set it to null before each call.

Attention: Check individual API documentation 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
  );
```

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 individual API documentation 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 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;
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
    );
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;
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
    );
    --
    -- 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 which 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.

Sample Code

```
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 which are of particular interest have been shown.

In the previous example, ellipses (...) indicate where irrelevant parameters to this example have not been listed.

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.

Sample Code

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 API Control Parameter p_object_version_number

Every published API has 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 which still list, the new object version number is passed out.

See:

Understanding the p_datetrack_update_mode control parameter

Understanding the p_datetrack_delete_mode control parameter

Handling Object Version Numbers in Forms 4.5

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.

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 the APIs which were included in the first version of Release 11, the scripts are located in the operating system directories $PER_TOP/admin/sql and $PAY_TOP/admin/sql. Refer to filenames such as pe???api.pkh, hr???api.pkh, and py???api.pkh.

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 which where not included in the first version of Release 11, may be provided in different operating system directories.

Oracle only supports the APIs listed in the Release new features documentation. That list is a reduced set of the server side code which matches all of the following three criteria:

- The database package name ends with "_API".
- The package header creation script filename conforms to pe???api.pkh, hr???api.pkh, or py???api.pkh naming standard.
- 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 by passed. Therefore, corrupting the data held within the Oracle HRMS application suite.

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 which may or may not be specified. Most parameters map onto a database column in the HR schema. There are some control parameters which affect the processing logic which 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 4.5 code, then all parameters in the API call must be specified. You cannot make use of the pl/sql declared default values because the version of pl/sql in Forms 4.5 does not support this.

Default Parameters with Create Style APIs

For APIs which 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 which do not a 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, you should set the variable used to set it to null before each call.

Attention: Check individual API documentation 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
  );
```

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 individual API documentation 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 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;
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
    );
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;
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
    );
    --
    -- 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 which 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.

Sample Code

```
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 which are of particular interest have been shown.

In the previous example, ellipses (...) indicate where irrelevant parameters to this example have not been listed.

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.

Sample Code

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 will all be 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 rollbacking 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 allows 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 which 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 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 which 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 which starts from today.

```
hr_grade_api.create_grade_rate_value
(...
,p_effective_date => trunc(sysdate)
 ...);
```

Example 2

This example creates a new employee who will join 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 which do not modify data in DateTrack entities still have a p_effective_date parameter. The date value is not used to affect when the changes take affect. It is used to validate QuickCode values. Each QuickCodes value can have 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 QuickCode after the end date.

Understanding the p_datetrack_update_mode Control Parameter

Most APIs which update data for at least one DateTrack entity have a p_datetrack_update_mode control parameter. It allows 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 which 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 which applies as of the date for the operation (i.e. p_effective_date).

Example

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

Grade_rule_id	Effective Start_Date	Effective_ End_Date	Version_ Number	Value
12122	01–JAN–1996	20–FEB–1996	2	45
12122	21–FEB–1996	20–JUN–1998	3	50

Also assume that the grade rate value was updated to the wrong value on the 21–FEB–1996. The update from 45 to 50 should have been 45 to 55 and you want to modify 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 which delete data for at least one DateTrack entity have a p_datetrack_delete_mode control parameter. It allows you to define the type of DateTrack change 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 which 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 individual API documentation 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 which applies as of the date for the operation (i.e. p_effective_date).

Example

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

```
                                          Object_
                 Effective_   Effective_   Version_
Grade_rule_id    Start_Date   End_Date     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.

Sample Code

```
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 which 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 which 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:

Grade_rule_id	Effective_ Start_Date	Effective_ End_Date
17392	01–FEB–1996	24–MAY–1996
17392	25–MAY–1996	01–SEP–1997

The update_grade_rate_value API is called to perform a DateTrack mode of UPDATE_CHANGE_INSERT with an effective date of 10–MAR–1996. The API then modifies the database rows to the following:

Grade_rule_id	Effective_ Start_Date	Effective_ End_Date
17392	01–FEB–1996	09–MAR–1996
17392	10–MAR–1996	24–MAY–1996
17392	25–MAY–1996	01–SEP–1997

The API p_effective_start_date parameter is set to 10–MAR–1996 and p_effective_end_date to 24–MAY–1996.

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 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 which 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 which 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.

A summary of what to do follows. You must maintain the order in which table rows are locked. You can only modify the order of the actual updates or deletes which can be modified.

See: HRMS Table Locking Ladder

Flexfields with APIs

Due to technology constraints it has not been possible to use the Flexfield definitions created using the Oracle Application Object Library Forms, to validate customer specific Flexfield data maintained using the APIs.

Oracle Applications expects these constraints to be removed in the near future, so that the APIs will be able to perform Flexfield validation by using the Flexfield definitions set–up using the Oracle Application Object Library Forms. For that reason, it is strongly recommended that you do not directly refer to any form fields in your value set definitions. Those references cannot be resolved in an API call, so the validation will not work in the future.

In the short and long term, APIs will not enforce value security. This can only be done when using the Forms.

For each Descriptive Flexfield, Oracle Applications has defined a structure column. In most cases the structure column name end with the letters, or is called, “ATTRIBUTE_CATEGORY”. It is possible for the implementation team to associate this structure column with a reference field. The structure column value can affect which Flexfield structure is will be used for validation. When reference fields are defined and you

wish to call the APIs, it is your responsibility to populate the attribute_category value with the reference field value.

Until the technology constraints are removed, if you wish to you Flexfields and the APIs to create/update Flexfield data, then you need to perform some additional set–up steps.

Skeleton Flexfield Validation Packages

For each entity which can be maintained using APIs, a skeleton Flexfield validation package is provided. Your implementation team can modify this skeleton code to perform your Flexfield validation using pl/sql.

The supplied validation raises a pl/sql exception if any of the Flexfield attribute parameters are set. Therefore if you wish to use APIs to maintain Flexfield data, you must implement your own validation in place of the supplied validation.

Other example validation code has been listed to help your implementation team. It has been placed between /**/pl/sql comment statements so it is not executed.

The skeleton Flexfield validation package body creation scripts can be found in the operating system directories $PER_TOP/admin/sql and $PAY_TOP/admin/sql. Refer to the filenames such as: pe???fli.pkb, hr???fli.pkb, and py???fli.pkb.

In these filenames, the ??? corresponds to a three letter entity short code. For example, the Person entity Descriptive Flexfield API validation package can be found in the $PER_TOP/admin/sql/peperfli.pkb file.

Skeleton Flexfield validation package creation scripts for new APIs, which where not included in the first version of Release 11, may be provided in different operating system directories.

Only the APIs will call these packages to validate Flexfield data. Forms will still use the definitions set up in the Oracle Application Object Library Forms. Hence, it is important for you to ensure that any changes to the definitions and Flexfield validation packages are synchronized.

Modifying Flexfield Validation Packages

Make a copy of the skeleton file. A procedure called *df* should be edited for Descriptive Flexfields and where applicable a procedure called *kf* should be edited for Key Flexfields.

For Descriptive Flexfields the APIs will call the validation packages after all the other attributes have been validated. Every attribute for the entity

is passed to the *df* procedure in a single pl/sql record structure. The components of the record structure cannot be modified; the values can only be read.

It is recommended that you implement the validation for each Flexfield segment structure in a separate procedure. Then modify the *df* procedure to call the relevant validation procedure depending on the structure column reference value.

For example, assume that the Person Descriptive Flexfield is being used to hold additional information depending on the person's marital status. (The database reference field is per_people_f.marital_status.) If the person is married, it is mandatory to hold the names of the towns where the person was married and born.

First, write a validation procedure to check the segment values when a person is married. Raise a pl/sql exception if there are any errors.

Sample Code

```
procedure val_per_marital_status_m
  (p_rec   in    per_per_shd.g_rec_type
  ) is
begin
  if p_rec.attribute1 is null then
    --
    -- This person is married but the Town of where
    -- the marriage took place has not been entered
    --
    dbms_standard.raise_application_error
      (num => -20999
      ,msg => 'Marriage Town must be entered'
      );
  elsif p_rec.attribute2 is null then
    --
    -- This person is married but the place
    -- of birth has not been entered
    --
    dbms_standard.raise_application_error
      (num => -20999
      ,msg => 'Place of birth must be entered'
      );
  end if;
  --
  -- Implement validation for other Married
  -- Flexfield attributes fields here.
  --
end val_per_marital_status_m;
```

Then modify the *df* procedure to call the validation procedure which corresponds to each marital_status value, as follows:

```
  procedure df
    (p_rec   in per_per_shd.g_rec_type
    ) is
  --
  begin
    --
    -- Check Context/Reference field value, then
    -- call relevant validation procedure.
    --
    if p_rec.marital_status is not null then
      --
      if p_rec.marital_status = 'M' then
        --
        -- Context/Reference field is 'Married'
        --
        val_per_marital_status_m(p_rec => p_rec);
      elsif p_rec.marital_status = 'S' then
        --
        -- Context/Reference field is 'Single'
        --
        val_per_marital_status_s(p_rec => p_rec);
      --
      --  ...repeat for each possible martial_status value which
      --  has a corresponding Flexfield segment structure
      --
    else
      --
      -- Structure/Reference field value is not supported
      --
      hr_utility.set_message(801,
'HR_7438_FLEX_INV_REF_FIELD_VAL');
      hr_utility.raise_error;
    end if;
  end df;
```

It is your responsibility to ensure that:

- The supplied Flexfield validation package body creation scripts are not edited.

  If you want to customize the supplied Flexfield validation packages, copy the scripts and modify the copies. After an upgrade, you should check that your customizations are consistent with the new packages supplied with the upgrade. If so, you can rerun your customized Flexfield package body creation scripts.

- No other API packages are modified.

  This includes the supplied Flexfield validation package headers.
- Any Flexfield attributes or segments that have no corresponding definition are checked to ensure that they contain a null value.
- Any code changes made to these files compile without errors and have been fully tested.
- If the Forms Flexfield definitions are altered, the corresponding pl/sql code change is made to your copies of the Flexfield validation package creation scripts.

Oracle Applications cannot accept any responsibility for Flexfield data created, updated or deleted using APIs with this mechanism.

In the future it will not be necessary to implement Flexfield validation in these server–side packages. When the technology constraints have been removed, Oracle Applications will not supply or execute existing skeleton package creation scripts. The Flexfield segment and value set definition set–up using the Oracle Application Object Library Forms will be used instead. For this reason Form fields must not be refereed to in value set definitions. This style of validation will fail in the future.

These temporary skeleton packages must only be used for implementing Flexfield validation. If you wish perform other field validation or perform Flexfield validation which cannot be implemented in value sets, then utilize API User hooks.

See: API User Hooks

Alternative APIs

Context–specific 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	create_person	create_gb_person

Table Column Name	_address API Parameter Name	_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 it 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 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 which corresponds to your country. If your company operates in various locations and you wish 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 the individual API documentation for further details of where other alternative interfaces are provided.

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 individual API documentation 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 rollback the work the API performed.

Sample Code

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_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;
```

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, a message table called HR_API_BATCH_MESSAGE_LINES is provided with 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 individual documentation.

Batch Message Lines Table

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_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 implicitly 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 return.
        --
        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 which might have been created during the processes can be viewed 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 Application 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 wish 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 let you 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 will stop and any custom logic will be executed. Then, assuming no errors have occurred, the main API processing will continue.

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 you should execute the supplied pre–processor program. This will look at the data you have defined, the package procedure you want to call and will build 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 will know 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 will rebuild the extra code needed to execute your

pl/sql from the specific user hooks without the need for manual edits to Oracle Application or your own source code files.

To implement additional logic in a pl/sql package and have this executed from an API user hook you need to complete the following steps:

1. Identify the apis and user hooks where you want to attach your extra logic.
2. Identify the data values available at the user hooks you intend to use.
3. Implement your extra logic in a pl/sql server–side package procedure.
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.
5. Execute the user hook pre–processor program. This will validate the parameters to your pl/sql server–side package procedure and dynamically generate another package body directly into the database. This generated code will contain pl/sql to call the custom package procedures from the API user hooks.

The following sections explain how to perform these steps in more detail.

Available User Hooks

API user hooks are provided in the HRMS APIs which 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 which lists all the different user hooks.

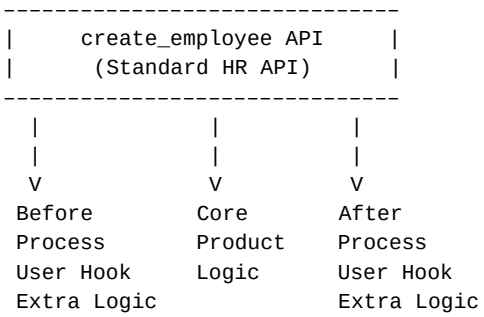
See: API User Hook Support Scripts

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 allows 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 which are going to be made by the API will have been made. Any values provided from these user hooks will have passed the validation checks. Your extra validation can assume the values provided are correct. If the main processing logic did not finish, due to an error, the After Process user hook will not be 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 which 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. With each new release of HRMS products a guide to new features is produced and this includes the list of supported and published HRMS APIs.

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.

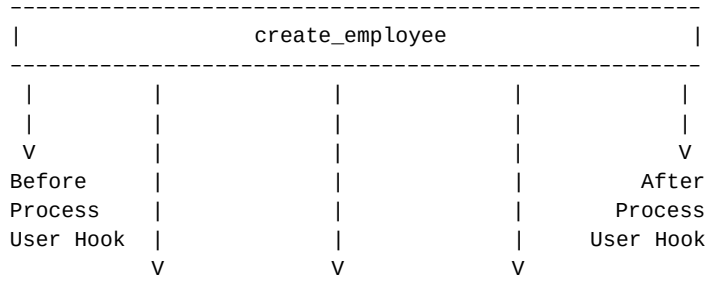
These row handler user hooks are provided after the DML has been completed for two reasons:

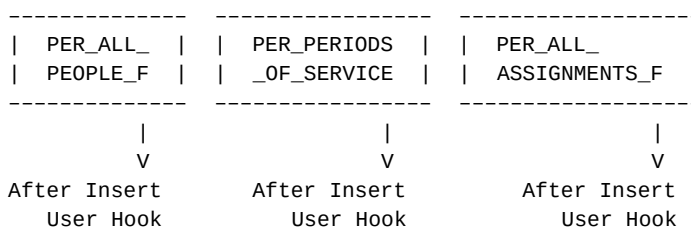
- All core product validation will have been carried out. So you know what the change to that particular table is valid.
- For inserts the primary key value will not be known until the row has actually been inserted.

  Note: Although the update or delete DML statements may have been executed, the previous – before DML, column values will be 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 will be many user hooks available for your use. For example, when the create_employee API is used data can be created 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 will be 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 which 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 will be executed. As a pl/sql exception is raised the 3rd and all remaining user hooks for that API call will not be executed.

> **Note:** When a DateTrack operation is carried out on a particular record only one row handler user hook will be executed. For example, when updating a person record, using the DateTrack mode 'UPDATE' only the *After Update* user hook will be 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.

A pl/sql script is available to list all of the available user hooks and the data values provided.

See: API User Hook Support Scripts

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 will not be 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 will have 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 will be available at the Before Process and After Process user hooks. At the Before Process hook none of the values will be validated.
- OUT parameter values on the published API will only be available from the After Process user hook. They are unavailable from the Before Process user hook because no core product logic will have been executed to derive them.
- IN OUT parameter values on the published API, will be 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 which will be passed out of the published API will be available at the After Process user hook.

From the row handler *After Insert* user hook only column values which 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 which correspond to the updated record and those values which 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 will be p_nationality. Where as the old value parameter name will be p_nationality_o.

Except for the primary key ID, if a database column cannot be updated a new value parameter will not be available. There will still be 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 will be available. But there will be 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 will be available with a parameter which 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 will need to explicitly select them 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

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 will fail and rollback 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 will be rolled back automatically.

Attention: Commits or full rollbacks are disallowed 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 will automatically rollback any database changes carried out by the current call to the published API. This rollback will include 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. 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 which 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

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.

For 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 martial status is 'Married' and sex is 'Female' then 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.

Sample Code

```
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 which contain data used by the API user hook mechanism, HR_API_MODULES and HR_API_HOOKS.

HR_API_MOUDLES

HR_API_MODULES contains a row for every API code module which contains user hooks.

HR_API_MOUDLES Main Columns

Column Name	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

Column Name	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 HRMS. 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.

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.:

Sample Code

```
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.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.
  --
  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 will need 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 will look 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 which 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 a customer does not want to use API user hooks then no further set–up 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, the custom packages should be modified, or the calling definition data in the HR_API_HOOK_CALLS table. Then the pre–processor program should be rerun, 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 created again.

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.

1. Execute the *hrahkall.sql* script in SQL*Plus

   This will create the hook package bodies for all of the different API code modules,.

2. Execute the *hrahkone.sql* script in SQL*Plus

   This will create 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

For 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*.

Error Report

Both pre–processor programs produce an error report. If no text is shown after the 'Created on' statement then all the hook package bodies have been created without any pl/sql errors or application errors.

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 which is not available, the hook package body will still be successfully created. No code will be created to execute that particular customer package procedure. If other custom package procedures need to be executed from the same user hook, then code to perform those calls will still be created – assuming they pass all the standard pl/sql checks and validation checks.

Attention: It is important that you check these error 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 error 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 then 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 will be called anytime 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 Type 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, you should 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 which 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 a particular business event has occurred, or you only want to perform some extra logic for a particular published API, then 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.

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 implementation of generic features. For example, with 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 will be 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 assignment entity, job and payroll values to make sure only 'Machine Workers' are included in the 'Weekly' payroll. There is more than one published API which 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 which are being modified. This prevent 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 which 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 range of dates the record is actually being modified are 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

A complete list of available user hooks and the data values provided can be found 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.

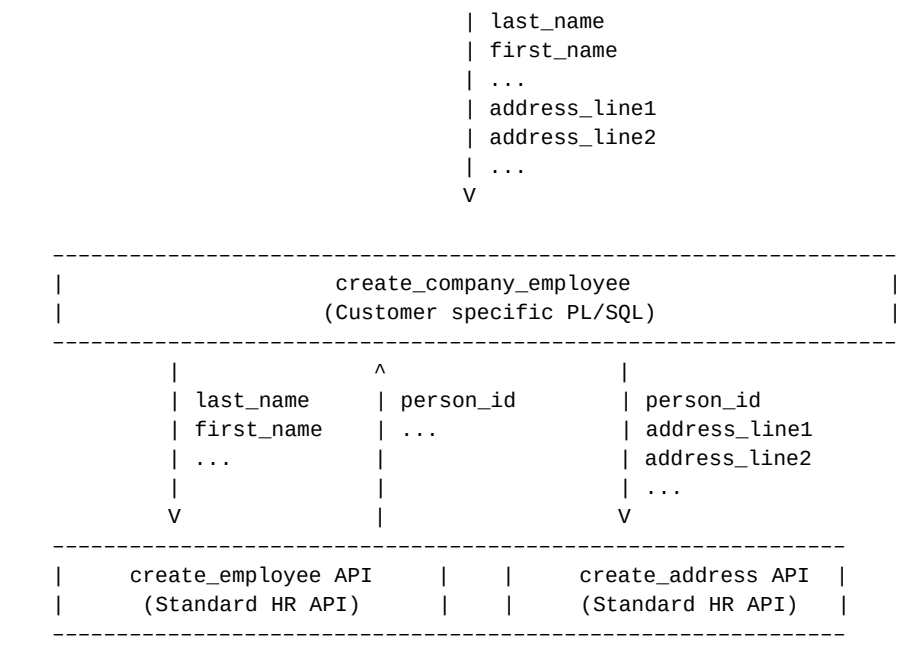
Oracle Applications support direct calls to the published APIs. Direct calls to any other server–side package procedures or functions written as part of the Oracle HRMS product set are not supported, unless explicitly specified.

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.

For example. Suppose that when a new employee joins your company you always obtain their home address at that time. The address details

must be recorded in the HR system because you run reports which 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 4.5

If you intend to write your own Forms which call the APIs, you will need to implement additional Forms logic to correctly manage the object version number. This is required because of the way Forms can process more than one row in the same commit unit.

Example

Consider the following example of what can happen if only one form's block item is used to hold the object version number:

1. The user queries two rows and updates both.

Row	OVN in Database	OVN in Form
A	6	6
B	3	3

2. The user presses commit.

   Row A has no user errors and is validated in the API. The OVN is updated in the database and the new OVN is returned to the form.

Row	OVN in Database	OVN in Form
A	7	7
B	3	3

3. The form calls the API again for row B.

   This time there is a validation error on the user–entered change. An error message is raised in the form and Forms4 issues a rollback to the database. However, the OVN for row A in the form is now different from the OVN in the database.

Row	OVN in Database	OVN in Form
A	6	7
B	3	3

4. The user corrects the problem with row B and commits again.

   Now the API will error when it validates the changes to row A. The two OVNs are different.

Solution

The solution to this problem is to use a non–basetable item to hold the new version number. This item is not populated at query time.

1. The user queries two rows and updates both.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	6	6	
B	3	3	

2. The user presses commit.

Row A is valid, so the OVN is updated in the database and the new OVN is returned to the form.

Note: The actual OVN in the form is not updated.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	6	7
B	3	3	

3. The forms calls the API again for row B.

   The validation fails and an error message is raised in the form. Forms4 issues a rollback to the database.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	6	6	7
B	3	3	

4. The user corrects the problem with row B and commits again.

   The API is called to validate row A again. The OVN value is passed, not the NEW_OVN. There is no error because the OVN in the database now matches the OVN it was passed. The API passes back the updated OVN value.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	6	7
B	3	3	

5. The API is called again to validate row B.

   The validation is successful; the OVN is updated in the database and the new OVN value is returned to the form. The commit in the form and the database is successful.

Row	OVN in Database	OVN in Form	New_OVN in Form
A	7	6	7
B	4	3	4

What would happen when the user updates the same row again without re–querying? Following on from the previous step:

6. When the user starts to update row A, the on–lock trigger will fire.

   The trigger updates the OVN when New_OVN is not null. (Theoretically the on–lock trigger will only fire if the previous commit has been successful. Therefore the New_OVN is the OVN value in the database.)

```
      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 which has to been 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 are 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  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;
```

HRMS Table Locking Ladder

This locking ladder should be used to avoid deadlock situations.

See: Avoiding Deadlocks

Table Name	Processing Order
PER_TIME_PERIOD_TYPES	10
FF_ROUTES	20
FF_CONTEXTS	30
FF_FORMULA_TYPES	40
FF_FORMULAS_F	50
PAY_BALANCE_DIMENSIONS	60
PAY_BALANCE_TYPES	70
PAY_MONETARY_UNITS	80
PAY_DEFINED_BALANCES	90
PAY_EXTERNAL_ACCOUNTS	100
PAY_PAYMENT_TYPES	110
PAY_ORG_PAYMENT_METHODS_F	120
HR_SOFT_CODING_KEYFLEX	130
PAY_COST_ALLOCATION_KEYFLEX	140

HR_LOCATIONS	150
HR_ORGANIZATION_UNITS	160
PER_NUMBER_GENERATION_CONTROLS	170
PER_ORGANIZATION_STRUCTURES	180
PER_ORG_STRUCTURE_VERSIONS	190
PER_ORG_STRUCTURE_ELEMENTS	200
PAY_CONSOLIDATION_SETS	210
PAY_ALL_PAYROLLS_F	220
PAY_ORG_PAY_METHOD_USAGES_F	230
PER_PARENT_SPINES	240
PAY_RATES	250
PER_SPINAL_POINTS	260
PER_GRADE_DEFINITIONS	270
PER_GRADES	280
PER_GRADE_SPINES_F	290
PER_SPINAL_POINT_STEPS_F	300
PAY_GRADE_RULES_F	310
PAY_EXCHANGE_RATES_F	320
PAY_CALENDARS	330
PER_TIME_PERIOD_SETS	340
PER_TIME_PERIOD_RULES	350
PER_TIME_PERIODS	360
PER_BUDGETS	370
PER_BUDGET_VERSIONS	380
PER_STARTUP_PERSON_TYPES	390
PER_PERSON_TYPES	400
PER_ALL_PEOPLE_F	410
PER_APPLICATIONS	420
PER_RECRUITMENT_ACTIVITIES	430
PAY_PEOPLE_GROUPS	440
PER_REQUISITIONS	450
PER_JOB_DEFINITIONS	460
PER_JOBS	470
PER_POSITION_DEFINITIONS	480
PER_POSITIONS	490
PER_VACANCIES	500
PER_RECRUITMENT_ACTIVITY_FOR	510
PER_PERIODS_OF_SERVICE	520
PER_POSITION_STRUCTURES	530
PER_POS_STRUCTURE_VERSIONS	540
PER_POS_STRUCTURE_ELEMENTS	550
PER_CAREER_PATHS	560
PER_CAREER_PATH_ELEMENTS	570
PER_COBRA_QFYING_EVENTS_F	580
PER_COBRA_COV_ENROLLMENTS	590
PER_COBRA_COVERAGE_BENEFITS_F	600
PER_COBRA_COVERAGE_STATUSES	610
PER_SCHED_COBRA_PAYMENTS	620
PER_ADDRESSES	630

PER_CONTACT_RELATIONSHIPS	640
PER_SPECIAL_INFO_TYPES	650
PER_ANALYSIS_CRITERIA	660
PER_PERSON_ANALYSES	670
PER_JOB_EVALUATIONS	680
PER_JOB_REQUIREMENTS	690
PER_VALID_GRADES	700
PER_ASSIGNMENT_STATUS_TYPES	710
PER_LETTER_TYPES	720
PER_LETTER_GEN_STATUSES	730
PER_LETTER_REQUESTS	740
HR_ORG_INFORMATION_TYPES	750
HR_ORGANIZATION_INFORMATION	760
HR_ORG_INFO_TYPES_BY_CLASS	770
PER_SECURITY_PROFILES	780
PAY_SECURITY_PAYROLLS	790
PAY_PAYROLL_LIST	800
PER_ORGANIZATION_LIST	810
PER_PERSON_LIST	820
PER_POSITION_LIST	830
PER_PERSON_LIST_CHANGES	840
HR_WORKFLOWS	850
HR_NAVIGATION_UNITS	860
HR_NAV_UNIT_GLOBAL_USAGES	870
HR_NAVIGATION_CONTEXT_RULES	880
HR_INCOMPATIBILITY_RULES	890
HR_NAVIGATION_NODES	900
HR_NAVIGATION_NODE_USAGES	910
HR_NAVIGATION_PATHS	920
PAY_CUSTOMIZED_RESTRICTIONS	930
PAY_RESTRICTION_PARAMETERS	940
PAY_RESTRICTION_VALUES	950
PER_ALL_ASSIGNMENTS_F	960
PAY_COST_ALLOCATIONS_F	970
PAY_ELEMENT_CLASSIFICATIONS	980
PAY_ELEMENT_TYPES_F	990
PAY_ELEMENT_SETS	1000
PAY_ELE_CLASSIFICATION_RULES	1010
PAY_ELEMENT_TYPE_RULES	1020
PAY_SUB_CLASSIFICATION_RULES_F	1030
PAY_BALANCE_CLASSIFICATIONS	1040
PAY_INPUT_VALUES_F	1050
PAY_BACKPAY_SETS	1060
PAY_BACKPAY_RULES	1070
PAY_BALANCE_FEEDS_F	1080
PAY_ELEMENT_LINKS_F	1090
PAY_LINK_INPUT_VALUES_F	1100
PAY_ASSIGNMENT_LINK_USAGES_F	1110
PAY_PAYROLL_GL_FLEX_MAPS	1120

PER_EVENTS	1130
PER_BOOKINGS	1140
HR_ASSIGNMENT_SETS	1150
PAY_ACTION_CLASSIFICATIONS	1160
PAY_ACTION_PARAMETERS	1170
PAY_PAYROLL_ACTIONS	1180
PAY_ASSIGNMENT_ACTIONS	1190
PAY_ACTION_INTERLOCKS	1200
PAY_PRE_PAYMENTS	1210
PAY_MESSAGE_LINES	1220
PAY_POPULATION_RANGES	1230
PAY_PERSON_LATEST_BALANCES	1240
PAY_ASSIGNMENT_LATEST_BALANCES	1250
PAY_BALANCE_CONTEXT_VALUES	1260
FF_LOOKUPS	1270
FF_HARNESS	1280
FF_FTYPE_CONTEXT_USAGES	1290
FF_QP_REPORTS	1300
FF_COMPILED_INFO_F	1310
FF_FUNCTIONS	1320
FF_FUNCTION_PARAMETERS	1330
FF_FUNCTION_CONTEXT_USAGES	1340
FF_GLOBALS_F	1350
FF_USER_ENTITIES	1360
FF_ROUTE_PARAMETERS	1370
FF_ROUTE_PARAMETER_VALUES	1380
FF_DATABASE_ITEMS	1390
FF_ROUTE_CONTEXT_USAGES	1400
FF_FDI_USAGES_F	1410
PAY_STATUS_PROCESSING_RULES_F	1420
PAY_FORMULA_RESULT_RULES_F	1430
PAY_ELEMENT_ENTRIES_F	1440
PAY_ELEMENT_ENTRY_VALUES_F	1450
PAY_QUICKPAY_INCLUSIONS	1460
PER_SPINAL_POINT_PLACEMENTS_F	1470
PAY_COIN_ANAL_ELEMENTS	1480
PAY_PERSONAL_PAYMENT_METHODS_F	1490
PAY_RUN_RESULTS	1500
PAY_RUN_RESULT_VALUES	1510
PAY_COSTS	1520
PER_ASSIGNMENT_INFO_TYPES	1530
PER_ASSIGNMENT_EXTRA_INFO	1540
PER_ASSIGNMENT_BUDGET_VALUES	1550
PER_LETTER_REQUEST_LINES	1560
PER_BUDGET_ELEMENTS	1570
PER_BUDGET_VALUES	1580
PER_SECONDARY_ASS_STATUSES	1590
PER_ABSENCE_ATTENDANCE_TYPES	1600
PER_ABS_ATTENDANCE_REASONS	1610

PER_ABSENCE_ATTENDANCES	1620
PER_PAY_PROPOSALS	1630
PER_PAY_BASES	1640
HR_ASSIGNMENT_SET_CRITERIA	1650
HR_ASSIGNMENT_SET_AMENDMENTS	1660
PER_QUICKPAINT_INVOCATIONS	1670
PER_QUICKPAINT_RESULT_TEXT	1680
PER_FORM_FUNCTIONS	1690
PAY_USER_TABLES	1700
PAY_USER_ROWS_F	1710
PAY_USER_COLUMNS	1720
PAY_USER_COLUMN_INSTANCES_F	1730
PAY_ACCRUAL_PLANS	1740
PAY_ACCRUAL_BANDS	1750
PAY_ELE_PAYROLL_FREQ_RULES	1760
PAY_FREQ_RULE_PERIODS	1770
PER_ASS_STATUS_TYPE_AMENDS	1780
PER_IMAGES	1790
PAY_NET_CALCULATION_RULES	1800
PAY_GEOCODES	1810
HR_API_MODULES	1820
HR_API_HOOKS	1830
HR_API_HOOK_CALLS	1840

The following tables have not yet been incorporated into the list:

HR_APPLICATION_OWNERSHIPS
HR_COMMENTS
HR_COMPARISON_ROWS
HR_LEGISLATION_SUBGROUPS
HR_OWNER_DEFINITIONS
HR_STU_EXCEPTIONS
HR_STU_HISTORY
PAY_JOB_WC_CODE_USAGES
PAY_LEGISLATION_RULES
PAY_LEGISLATIVE_FIELD_INFO
PAY_PDT_BATCH_CHECKS
PAY_PDT_BATCH_HEADERS
PAY_PDT_BATCH_LINES
PAY_PDT_LINE_ERRORS
PAY_STATE_RULES
PAY_TAXABILITY_RULES
PAY_WC_FUNDS
PAY_WC_RATES
PAY_WC_STATE_SURCHARGES

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 additional 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 Payroll User's Guide* for more information.

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 and allows 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 and in general they 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 with 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 will automatically generate 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 will always refer to the 'defined balance' to return a value.

Latest Balances

In order 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, and in some cases it will clear and then reset 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 xxx balance function. This will always find the latest value of the balance for the dimension you select.

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

These allow the definition of complex balances. 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 are 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.

If you need to define additional balance contexts you should contact your Oracle HRMS Support representative for more information.

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.

Balance Dimension Attributes

The following list provides a description of most attributes of a balance dimension. It does not include attributes that have an obvious meaning.

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 which 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(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, i.e. the current action for this assignment.
   - The ASSACT table is the 'target' assignment action, i.e. the action for those results that feed the balance.
   - The PACT table is the 'target' payroll action, i.e. 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. i.e. 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 PAY_ASSIGNMENT_LATEST_BALANCES.
- **P** – Fed and stored at person level. This dimension type createds a person level latest balance (i.e. stored 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** This indicates that a procedure is called to perform additional feed checking. There is a defined interface that returns a value to indicate feeding will or will not occur.
- **E** This indicates that equality feed checking is done. When a route accepts contexts that are not supported by the core Payroll Run.
- **J** This indicates that Jurisdiction checking is done. This is specific to US legislative balances.
- **S** This indicates that Subject Feed Checking is done. This is specific to US legislative balances.

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 in Release 10SC. 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 is completely different from the steps an enduser follows to adjust a balance, and you should not try to use the balance loading method to make balance adjustments.

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 will calculate and insert the correct run results to set the correct initial values with effect from the upload date.

Differences from Release 9

If you are migrating from an earlier release of Oracle Payroll, it is important to recognize that this area of the system has been completely redesigned. In previous releases all balance values were held explicitly. This meant that many balance items had to be maintained in step with the processing of run results and this produced a large overhead in payroll processing.

In Release 10SC balance items have been dropped and balances are now 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' . (Refer to your *Oracle Payroll User's Guide* for details.) When you make a balance adjustment online, the effect is to create a single processed run result for the element. This run result will automatically feed, or adjust, 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 3 basic steps involved in loading initial balance values:

1. Define an Element and Input Value to feed each specific balance
2. Setup 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

Overview

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 date–effective balance entries, or 'adjustments' to ensure your legislative balances are correct from the upload date. Mainterance of

balance information after this date is managed by the system, or by using the balance adjustments.

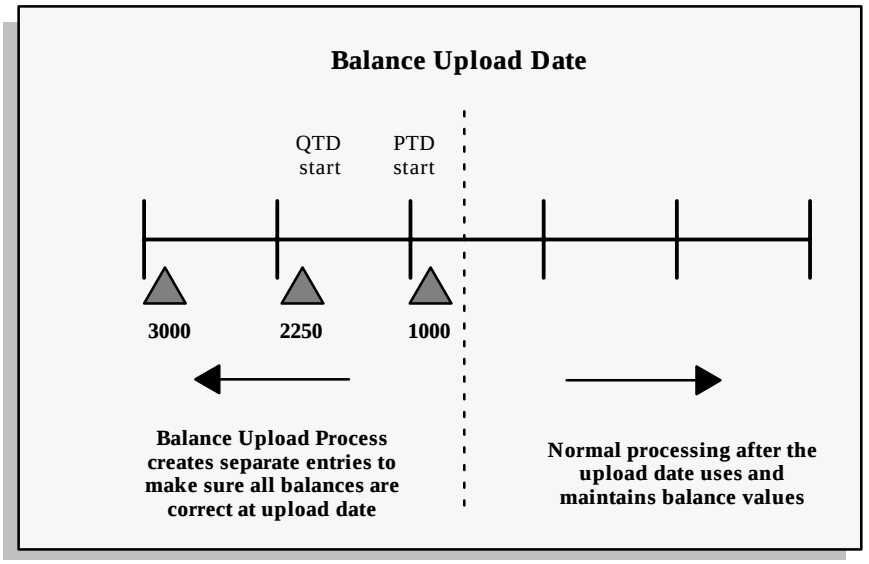
Consider the following example of three dimensions for gross pay balance values for one employee.

- 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 are not permitted to have latest balances, for example balances that are used in–memory but not stored, thus latest balances for these should not be created.

When you are deciding which balances and dimensions you should include in the initial loading process you should 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. In this case, 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 which 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.

Attention: US users should run the special plsql script – paybalup.pkb, to create the elements and inputs you need 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.

UK users need only link the predefined elments that feed the legislative balances that must be initialized. Refer to the essay on UK Legislative Balance Initialization in this manual for a list of these elements and balances: See page NO TAG

All users need to define additional elements and input values to set up initial balance values for their own earnings and deductions.

Setup 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.

Oracle Payroll comes with the predefined elements and input values you need to set initial values for all your legislative balances.

For all other balances you will 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 is always available.

- must have a classification of 'Initial Balance Feed'

  This new classification is excluded from the list of classifications available when you define a balance. You can only setup 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.

Setup 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 the 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 and it also lets you 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)

Name	Null?	Type
BUSINESS_GROUP_NAME		VARCHAR2(60)
PAYROLL_NAME		VARCHAR2(80)

Each batch identifies the payroll which is being uploaded and the date of the upload. Other identifiers can be set to uniquely identify each batch eg.

Batch Name	Batch Ref	Batch Source	Payroll	Upload Date
Weekly Payroll	0001	SQL*Loader	Pay1	01–Jan–1995
Weekly Payroll	0002	SQL*Loader	Pay1	01–Jan–1995
Monthly Payroll	0003	SQL*Loader	Pay2	01–Jan–1995
Semi Monthly Payroll	0001	Screen	Pay3	01–Aug–1995

PAY_BALANCE_BATCH_LINES

Name	Null?	Type
ASSIGNMENT_ID		NUMBER(9)
BALANCE_DIMENSION_ID		NUMBER(9)
BALANCE_TYPE_ID		NUMBER(9)
PAYROLL_ACTION_ID		NUMBER(9)

(Page 1 of 2)

Name	Null?	Type
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)

(Page 2 of 2)

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 or a user ID for each piece of information eg. assignment_id and

assignment_number as this allows maximum flexibility when the user is populating the batch tables.

The rule is that if both are specified then the system ID overrides the user ID. Here's a list of the system ID's and user ID's 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	
JURISDICTION_CODE	

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 are not validated by the initial balance loading process, and it is the responsibility of the user to validate this information. This includes, for example the GRE or Legal Company.

> **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.

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 it ran out of tablespace.

Undo Transfer

This mode will remove 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

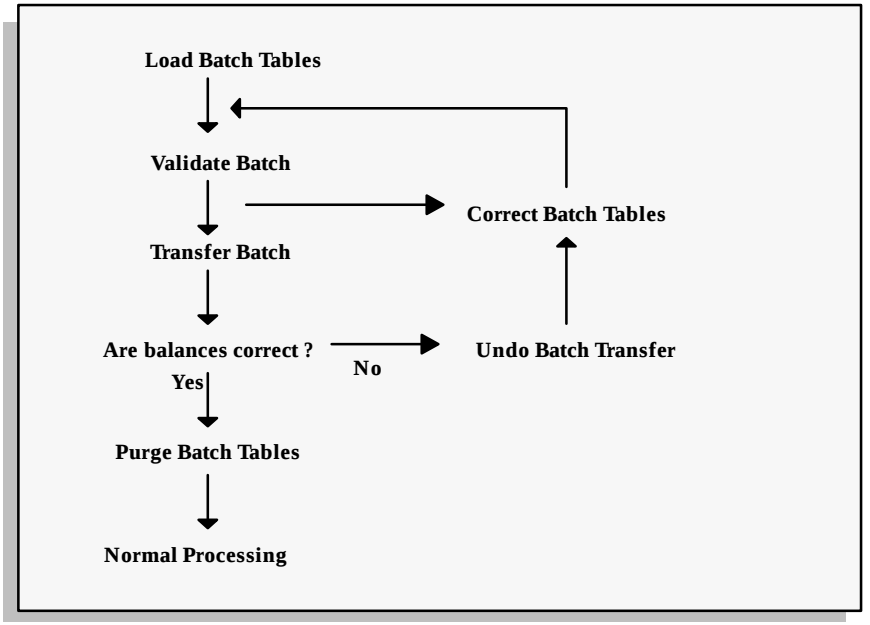
Purges all data in a batch regardless of current status. When a batch is purged all the messages, batch lines and the batch header are removed. This allows you to reclaim space once a batch is successfully transferred.

> **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.

Process Flow

As this implies, there is a defined flow to the loading of the initial balances, this can be seen in the following diagram:–

Process Flow



Warning: Once you have purged a batch, all the entries for that batch are deleted. This action cannot be undone.

Error Statuses

Any errors encountered are recorded in the messages table against the object being validated ie. 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 lines 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, if an assignment has the following upload requests:

- <Balance>_QTD = 1500.00
- <Balance>_YTD = 1000.00

In this example the YTD value is lower than the QTD value. This may be valid, if the balance decreases over time. However, in most instances balances will be increaing, so it is advisable to check a balance if 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, good practice is always to make sure that the 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 setup 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. UK users need only to 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 deduction.
   - follow the requirements listed above
   - 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:

5. 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
```

6. 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 will use 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           200
60001             Tax Paid      QTD           400
60001             Tax Paid      YTD           400
```

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.

7. From the Submit Requests window, run the Initialize Balances 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 may be 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.

> **Note:** Currently the dynamic pl/sql routines that are used in the balance function do not have pragma level of WNPS and WNDS (Write No Package State and Write No DML). This means that the pl/sql function cannot be called as part of a view.
>
> 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:

1. You can easily call the function from a form or SRW2 report.
2. 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:

1. 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 allows the balance function to operate without hard coded knowledge of the contexts, and reduces client–server calls for several balances.
2. The 'C' balance user exit works in 2 modes, date and assignment action. For the balance function rather than pass a mode parameter, this is resolved by using the pl/sql overloading feature. This simplifies the interface.

The pl/sql code resides in 1 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 will update the value.

> **Note:** The context name can be specified in any case. The routine will convert 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 still 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);
```

US Legislative Balance Initialization

This essay supplements the technical essay entitled *Balances in Oracle Payroll*. Specifically, it covers the following topics:

- Balance initialization elements
- Supported dimensions
- Overview of tax–related balances
- Balances that require initialization
- Required US legislative balances

Balance Initialization Elements

US users should run the "Initial Balance Structure Creation" process to create the elements and input values required to feed the predefined legislative balances.

You may need to define additional elements and input values to set up initial balances for your own earnings and deductions.

Supported Dimensions

The following balance dimensions (BD) are currently supported:

- ASG_GRE_YTD — Assignment within GRE Year to Date
- ASG_GRE_QTD — Assignment within GRE Quarter to Date
- ASG_GRE_PTD — Assignment within GRE Period to Date

At the Federal level, the SUBJECT_TO_TAX dimension is supported for balances that hold Earnings / Deduction amounts that are subject to Tax:

- SUBJECT_TO_TAX_ASG_GRE_QTD
- SUBJECT_TO_TAX_ASG_GRE_YTD
- SUBJECT_TO_TAX_ASG_GRE_PTD

For Jurisdiction (State, County, City, School District) level taxes, the dimension of Jurisdiction (JD) is supported:

- ASG_GRE_YTD_JD
- ASG_GRE_QTD_JD

- ASG_GRE_PTD_JD

Overview of Tax–related Balances

In accordance with the tax related information entered in the Oracle HRMS database, the payroll run can build tax–related balances for each of the following tax types.

Federal

The federal tax types follow:

- FIT
- FUTA
- SS
- Medicare
- Earned Income Credit

State

The state tax types follow:

- SIT
- SUI
- SDI

Local

The local tax types follow:

- City Tax
- County Tax
- Head Tax
- School District Tax

Balances Created for Each Assignment

The following tax–related balances are created for each employee assignment:

- Gross Earnings
- Exempt Earnings
- Gross Earnings Subject to Tax
- Gross Earnings Subject to Tax and not Withholdable
- Gross Earnings Subject to Tax and Withholdable
- 401k, 125 and Dependent Care Reductions
- Reduced Subject to Tax and Withholdable Earnings
- Employee withheld
- Reduced Subject EIC
- EIC advance
- Employer Liability

Balances Created for Taxes with Upper Earnings Limits

For taxes with upper earnings limits, the following balances are created.

- Taxable Earnings
- Excess Earnings

Where Tax–related Balances Exist in the System

Tax–related balances exist in the system in the following locations:

- SOE
- View Tax Balances
- W2
- 941
- Tax Summary Report

Balances That Require Initializing

Following is a list of US balances that should be initialized in Oracle Payroll in order to obtain accuracy and consistency in the different areas of the system that use and report such balances.

Earnings and Deductions

The US earnings and deductions balances that should be initialized follow:

- Individual balances

  Oracle Payroll does not require initialization of individual balances for Earnings and Deductions in order to derive the necessary tax–related balances. However, if there is a requirement to initialize these balances for reporting or any other reason, the system does support loading of such balances. Initialization of individual balances does *not* impact calculation of tax related cumulative balances.

- Gross Earnings (GROSS_EARNINGS_*BD*)

  Gross Earnings must be initialized; it *cannot* be derived from individual Earnings type balances.

Federal Level Tax–related Balances

For each Federal Tax type, you must load the following appropriate balances in order to get accurate values for derived balances:

- Gross Earnings (GROSS_EARNINGS_*BD*)
- Regular Earnings (REGULAR_EARNINGS_*BD*)
- Supplemental Earnings for *Tax* (SUPPLEMENTAL_EARNINGS_FOR_*TAX_BD*)
- Supplemental Earnings for NW*Tax* (SUPPLEMENTAL_EARNINGS_FOR_NW*TAX_BD*)

Oracle Payroll differentiates between Earnings Types for tax calculation rules and as such holds balances for the different types (Regular Earnings and Supplemental Earnings in particular). Supplemental Earnings and Imputed Earnings that are subject to tax are rolled into a single balance: Supplemental Earnings for *Tax***.** If your current system does not maintain these split balances, you could roll your single "Subject to Tax" Earnings balance into either bucket. Additionally, a bucket, Supplemental Earnings for NW*Tax*, is provided for your Supplemental and Imputed Earnings that are subject to tax, but are not withholdable.

- Def Comp 401K (DEF_COMP_401K_*BD*)
- Def Comp 401K for *Tax* (DEF_COMP_401K_FOR_*TAX_BD*)
- Section 125 (SECTION_125_*BD*)

- Section 125 for *Tax (*SECTION_125_FOR_*TAX_BD*)
- Dependent Care (DEPENDENT_CARE_*BD*)
- Dependent Care for *Tax*
- (DEPENDENT_CARE_FOR_*TAX_BD*)

Tax Calculation requires the amount that is reduced from the Subject to Tax Earnings amount. This reduced amount is derived from subtracting the amount of Pre–Tax Deductions that is subject to Tax from the total Pre–Tax Deduction amount. Example: Def Comp 401K is the total pre–tax amount. Def Comp 401K for Tax is the amount of pre–tax deduction that does NOT reduce Subject to Earnings, and as such must be initialized only if it is nonzero. The same is true for Section 125 and Dependent Care.

- Employee *TAX* Withheld (*TAX*_WITHELD_*BD*)
- EIC Advance (EIC_ADVANCE_*BD*)
- Taxable Earnings for taxes with upper limits (*TAX*_TAXABLE_*BD*)

Taxable Earnings must be set for FUTA, SS and Medicare. The balance initialization process does not include validation for upper limit and hence the amount initialized must NOT exceed the upper limit.

> **Note:** Medicare Taxable will be equal to the subject amount following recent legislation changes.

- Employer Liability Balances

The following balances are DERIVED from the values entered in the above mentioned balances.

> **Note:** *Do not* initialize these explicit balances for Federal level taxes.

- Subject Witholdable = Regular Earnings + Supplemental Earnings for *Tax*
- Subject Nwhable = Supplemental Earnings for NW *Tax*
- 401 Reductions = Def Comp 401K – Def Comp 401K for *Tax*
- 125 Reductions = Section 125 – Section 125 for *Tax*
- Dep. Care Reductions = Dep. Care – Dep. Care for *Tax*
- Subject Earnings = Subj Whable + Subj NWhable
- Exempt = Gross – Subject Earnings

- Reduced Subj Whable = Subj Whable – 401 Reductions – 125 Reductions –
- Dep Care Reductions
- Excess = Reduced Subj Whable – Taxable

State, County, and City (Jurisdiction) Level Tax–related Balances

For each Jurisdiction Tax type, the following appropriate balances must be initialized:

> **Note:** At the jurisdiction level, the subject balances do not require the "SUBJECT_TO" dimension.

- Gross Earnings (*TAX*_GROSS_BD_JD)
- Total Gross Earnings earned within a particular jurisdiction

Because of the earnings accumulations rules (due to possibility of a percentage of time worked in different jurisdictions), for nonfederal taxes, the "Subject" and "Reduced" tax–related balance values are not derivable directly from the cumulative Earnings / Deductions type balances. Hence the "Subject" and "Reduced" tax–related balance values must be initialized explicitly.

Do not initialize the Supplemental Earnings, Withholdable and Notwithholdable, balances for jurisdiction level taxes. Instead, initialize the following explicit balances:

- Subject Withholdable (*TAX*_SUBJ_WHABLE_*BD*_*JD*)
- Subject Notwithholdable (*TAX*_SUBJ_NWHABLE_*BD*_JD)
- 401 Reductions (*TAX*_401_REDNS_*BD*_JD)
- 125 Reductions (*TAX*_125_REDNS_*BD*_JD)
- Dep Care Reductions (*TAX*_DEP_CARE_REDNS_*BD*_JD)
- Employee Withheld (*TAX*_WITHHELD_*BD*_*JD*)
- Taxable Earnings for taxes (SDI, SUI) with upper limits (*TAX*_TAXABLE_*BD*_*JD*)
- Employer Liability

The following balances are derived:

- Subject Earnings = Subj Whable + Subj NWhable
- Exempt = Gross – Subject Earnings

- Reduced Subj Whable = Subj Whable – 401 Reductions – 125 Reductions –
- Dep Care Reductions
- Excess = Reduced Subj Whable – Taxable

Other Balances

Other balances to consider follow:

- Net (NET_*BD*)

  This Balance holds Net Pay, *exclusive* of non–payroll payments.

- Payments (PAYMENTS_*BD*)

  This balance holds Net Pay, *inclusive* of non–payroll payments.

You must initialize the above two balances in order to see accurate YTD Net Pay figures on the on–line Statement of Earnings. Currently the YTD Net on the check stub and deposit advice is derived and does not use these balances explicitly. However, a design change has been scheduled for the near future to make these reports consistent, so that all of them will use the explicit Payments / Net Balance.

- W2 BOX 13A through W2 BOX 13P
- W2 BOX 14A through W2 BOX 14J
- W2 Pension Plan

The W2 Balances hold the specific W2 Box related balances; as such, they must must be initialized based on your reporting requirements.

For ongoing maintenance of these balances, you must set up the Balance feeds explicitly for these balances. To do so, use the Balance / Balance Feeds window. W2 related balances do *not* have pre–defined feeds.

Required US Legislative Balances

The following table lists required US legislative balances.

> **Note:** The Balance Names and Dimension Names are case sensitive and MUST be loaded in Initcaps. This is similar to how they are stored in the Balance Type and Dimension tables in Oracle Payroll.

Balance	Dimension
Gross Earnings	Assignment within GRE Year to Date
Net	Assignment within GRE Year to Date
Payments	Assignment within GRE Year to Date
Regular Earnings	Assignment within GRE Year to Date
Supplemental Earnings for *Federal Tax*	Subject to Tax for Assignment within GRE Year to Date
Supplemental Earnings for NW *Federal Tax*	Subject to Tax for Assignment within GRE Year to Date
Def Comp 401K	Assignment within GRE Year to Date
Def Comp 401K for *Federal Tax*	Subject to Tax for Assignment within GRE Year to Date
Section 125	Assignment within GRE Year to Date
Section 125 for *Federal Tax*	Subject to Tax for Assignment within GRE Year to Date
Dependent Care	Assignment within GRE Year to Date
Dependent Care for *Federal Tax*	Subject to Tax for Assignment within GRE Year to Date
Federal Tax Withheld	Assignment within GRE Year to Date
EIC Advance	Assignment within GRE Year to Date
Federal Upper Limit Tax Taxable	Assignment within GRE Year to Date
Federal Tax Withheld	Assignment within GRE Year to Date
Federal Tax ER	Assignment within GRE Year to Date
Juristiction Tax Gross	Assignment in JD within GRE Year to Date
Jurisdiction Tax Subj Whable	Assignment in JD within GRE Year to Date
Jurisdiction Tax Subj Nwhable	Assignment in JD within GRE Year to Date
Jurisdiction Tax 401 Redns	Assignment in JD within GRE Year to Date
Jurisdiction Tax 125 Redns	Assignment in JD within GRE Year to Date
Jurisdiction Tax Dep Care Redns	Assignment in JD within GRE Year to Date
Jurisdiction Tax Withheld	Assignment in JD within GRE Year to Date
Jurisdiction Upper Limit TAX Taxable	Assignment in JD within GRE Year to Date
Jurisdiction Tax ER	Assignment in JD within GRE Year to Date
W2 BOX 13A through W2 BOX 13P	Assignment within GRE Year to Date
W2 BOX 14A through W2 BOX 14J	Assignment within GRE Year to Date
W2 Pension Plan	Assignment within GRE Year to Date

Balances Reported on W2 and 941

The following table shows balances reported on W2 and 941 forms.

Balance Name	On W2	On 941
Reduced Subject Withholdable (Derived)	Yes	Yes
Subject not Withholdable	Yes	Yes
FIT Wittheld	Yes	Yes
SS EE Taxable	Yes	Yes
SS EE Withheld	Yes	Yes
Medicare EE Taxable	Yes	Yes
Medicare EE Withheld	Yes	Yes
EIC Advance	Yes	Yes
Dependent Care	Yes	
* W2 BOX 13A through W2 BOX 13P	Yes	
* W2 BOX 14A through W2 BOX 14J	Yes	
* W2 Pension Plan	Yes	
SIT Subj. Whable	Yes	
SIT Withheld	Yes	
County Subj Whable	Yes	
County Withheld	Yes	

* Balance feeds for these balances are not pre–defined and must be defined by the user based on specific reporting requirements.

US Dependents and Beneficiaries

This essay discusses implementing dependents and benficiaries in the US for basic benefits administration. Specifically, it covers the following topics:

- Overview
- Design
- Windows and data entry
- Additional notes

Overview

Oracle HRMS maintains basic benefit enrollments. A benefit enrollment gives rise to an employee contribution, which is deducted from the employees pre–tax or post–tax pay, and an employer contribution. A benefit enrollment has a coverage type.

A benefit enrollment may have a number of covered dependents. Details of the covered dependents are stored on the database. Covered dependents may be entitled to continued coverage through COBRA upon termination of the employee enrolled.

A benefit enrollment may have designated beneficiaries. Multiple beneficiaries may be designated for a benefit enrollment. Each designated beneficiary must have a proportion of the benefit assigned to them. Beneficiaries may also have a *level* specified for them. Benefit goes to the highest surviving level only.

Details of beneficiaries are stored on the database. There should, at least, be sufficient information to allow the beneficiary to be contacted in the case of payment. Beneficiaries may be either people or organizations.

Design

Benefit plans are defined as element types. The benefit plan has a benefits classification. Benefit plans have input values of coverage, employee contribution and employer contribution. At creation time the user is prompted to specify if the units of measure of the contributions are to be *money* or *units* (usually percentages). Both contribution input values have the same unit of measure.

The benefits classification indicates whether the benefit plan requires or allows beneficiaries or dependents.

The benefit classification includes a flag to indicate if the benefit contributions table is to be used to determine benefit contributions. If the benefit contribution table is to be used, then benefit coverages for a benefit plan and the contributions for them are defined in the benefits contributions table. There is an entry for each coverage type with the default employer and employee contribution.

An element entry is given to an assignment to enroll in a benefit plan. Enrollments should always be for the primary assignment. Deductions will be made for the benefit plan.

Depending on the dependents allowed flag for the benefit classification, a benefit plan may have various coverage types, such as employee only, employee and spouse, employee and children and employee and family. The coverage type is held on the coverage input value for the benefit enrollment.

If the benefit classification specifies that dependents are not allowed for this benefit plan, the coverage input value is set to Employee Only. The employee and employer contribution input values are copied from the contribution table but can be overwritten for individual employees.

If the benefit plan benefit classification allows dependents or beneficiaries, it is possible to work flow from the Element Entries window to the Covered Dependents window (dep) and the designate beneficiaries window (ben). (It is possible for a benefit plan to allow both dependents and beneficiaries).

Dependents for a benefit plan are stored on the database in the people tables. The user enters dependents using the define contact window. A valid dependent type is any contact type with a row existing in the valid dependent types table. Entries in the valid dependent type table are provided as startup data and specify which contact types are covered by a coverage type.

If the benefit enrollment coverage is other than Employee Only, then the covered dependents table is populated for the benefit enrollment element entry.

The Covered Dependents window allows the user to modify the covered dependents for a benefit enrollment. Allowed dependents are restricted to those who had a dependent flag set when entered, and who are of a valid dependent type for the coverage. Covered dependents are date effective. It is possible to display the date tracked history of dependents covered by a benefit enrollment.

The beneficiaries allowed flag on the benefit classification specifies whether beneficiaries may be designated for a benefit plan.

Beneficiaries can be organizations or people. People may be entered using the define contact screen. The contact should have the beneficiary flag set. The relationship to the plan enrollee is not significant.

Organizations may be entered using the Organization window. The organization is given a classification of benefit beneficiary. The benefit beneficiary organization type is provided with the startup data. Beneficiary organization addresses may be entered using the location screen. The location screen includes a field into which a contact at the organization may be entered.

Beneficiaries are specified using the designate beneficiary window. This window allows the user to enter multiple beneficiaries for a benefit enrollment. For each designated beneficiary a beneficiary type (person / organization) is selected. The beneficiary type restricts the values displayed in the beneficiary list. A beneficiary person is selected from all contacts for the plan enrollee with the beneficiary flag set. A beneficiary organization may be selected from all organizations of type benefit beneficiary.

Each designated beneficiary is given a level (the benefit goes to the highest surviving level) and a proportion (the benefit is divided according to the proportions of surviving beneficiaries at a beneficiary level). There is a text field to allow the enrollee to enter special instructions against the beneficiary. Designated beneficiaries are date effective. It is possible to display the date–tracked history of the beneficiaries of a benefit enrollment.

Windows and Data Entry

The user can define benefit plans using the Deduction window for US Payroll or the Element Entries window for Oracle HRMS–only installations.

If the benefit classification indicates that the benefit contributions table is to be used, then the user uses the Benefit Contributions window to enter benefit coverages for a benefit plan and the contributions for them. A benefit plan is selected for the window. Available coverage types are entered for it. For each coverage type the employer and employee contribution are entered. If there is no employer or employee contribution required for a certain coverage type the user enters a contribution of zero.

Element Entries Window

The user uses the Element Entries window to enroll an assignment in a benefit plan. The user then queries the deduction for the benefits plan and selects the correct coverage type for the coverage input value for the element entry. The employee and employer contributions default from the contribution table, but may be overwritten for the enrollment. The user can workflow from this window to the cover dependents window and the designate beneficiary window (depending on the benefit classification of the benefit plan).

Designate Beneficiary Window

The user enters beneficiaries using the designate beneficiary window. This window allows the user to enter multiple beneficiaries for a benefit enrollment. The user can select a beneficiary person from all contacts for the plan enrollee with the beneficiary flag set. A beneficiary organization may be selected from all organizations of type benefit beneficiary.

Each beneficiary is given a beneficiary type (person / organization), a beneficiary level (benefit goes to the highest surviving level) and a proportion (benefit is divided according to the proportions of surviving beneficiaries at a beneficiary level). The user may also enter a text string for special instructions to be considered with each beneficiary.

The start and end dates for the designated beneficiary are displayed.

Covered Dependents Window

The Covered Dependents window is used to enter or modify dependents covered by an benefit enrollment. Dependents may be selected from the contacts specified with the dependent flag set. When the covered dependents are entered they are tested to ensure that the contact type is one of the valid dependent types for the coverage specified for the benefit enrollment. A test is made to ensure that the number of contacts of the type does not exceed the maximum specified for the valid dependent type.

The start and end date for coverage of the dependent is displayed.

Organization Window

The Organization window is used to enter organizations. The organization is given a classification benefit beneficiary.

Contact Window

The Contact window is used to enter people. The window allows entry of the name, address, telephone number and social security number. Flags indicate whether the contact is a dependent of the employee, a beneficiary of the employee, or a series EE bond recipient. If the contact is a dependent of the employee, then the contact type must be a valid dependent type.

The following warnings are displayed:

- When the user attempts to leave the window without specifying a social security number.
- When the user makes changes to contacts who are assigned as beneficiaries or dependents to benefit enrollments. The warning indicates that coverage should be checked.
- When a new contact is entered of a dependent type which would be covered by an existing plan enrollment. The warning suggests that the user check the coverage.

Additional Notes

When changes to dependents are made, covered dependents for a benefit are not updated. Each time there is a change to the details of a covered dependent, or a new dependent is entered, the user must ensure that the dependents for a plan remain valid.

A warning message is displayed when dependent / beneficiary details are altered to warn that the person is assigned to benefit plans and coverage details should be checked.

Dependents are validated only against the dependent contact type and the maximum number of the dependent contact type for the coverage. There is no attempt to ensure that the dependent meets more sophisticated criteria (children must be under 18 or in school full time, for example). Changes to dependents eligibility due to age or other time dependent features is not detected.

Changes to the plan enrollment such as coverage result in revalidation of covered dependents. A change to a plan enrollment should be given a reason. There is no attempt to ensure that enrollment changes result from valid life events.

This document does not address COBRA coverage issues and responsibilities. It is assumed that existing COBRA reports will continue to satisfy requirements.

It is not proposed to seed workflows from the element entry screen, the enter covered dependent screen or the designate beneficiary screen to the enter contact screen. However, the user may wish to define workflows to do so.

Date track history is available for covered dependents and designated beneficiaries. However, it should be noted that these are multirow tables and the history relates to the row rather than to the person / organization concerned.

US Payroll Tax Subsystem

This essay introduces the tax subsystem in Oracle Payroll for US legislations. It is assumed that you understand the use of elements, entries, formulas, run results and balances in Oracle Payroll. Specifically, this essay covers the following topics:

- Installed tax system
- Earnings and deductions
- Taxes
- Tax balances
- Database design
- Other forms
- Tax implementation

Installed Tax System

When you install Oracle Payroll with a US legislation, you automatically receive the predefined components you need to calculate the legislated Federal, State and City taxes within the US. These components include the specific Classifications and Categories of elements, Taxability Rules, standard Formulas, Balances and Balance Dimensions you need. As part of your install process you will also install the Vertex tax tables used by Oracle Payroll for all US jurisdictions.

GEOCODES

The interface between Oracle Payroll and the Vertex tax subsystem makes use of a key called the GEOCODE to identify every jurisdiction in the US.

The geocode has a numeric format *nn–nnn–nnnn* and is used to identify the combination of State, City and Locality for tax liability. The geocode is not normally visible to users of Oracle Payroll and special validation is provided on entry of the address for a location and the primary address for an employee to ensure accurate geocodes for both home and work addresses.

Earnings and Deductions

Oracle Payroll uses a combination of classifications and categories of elements to determine the taxability rules of every earnings and deduction.

Regular Earnings

For tax purposes, regular earnings are always subject to taxes.

When you define a type of Regular Earnings, the system automatically generates an input value to hold a jurisdiction value. This lets you enter a Vertex geocode for a specific jurisdiction. For example, you might want to create a regular earnings type of ”Time Entry Wages – Houston” that always has a ”Jurisdiction” entry value of 44–201–1440 for Houston, Texas.

Regular Earnings for which the jurisdiction element entry value is null are considered untagged. This distinction is important for the Earnings Accumulation Rules.

Supplemental Earnings

Supplemental earnings include Bonuses, Commissions or Cash Awards. These earnings are subdivided into several categories and each category may be subject to or exempt from any given tax in any jurisdiction.

Imputed Earnings

Imputed Earnings are non–cash earnings, such as personal use of a company car. These earnings have a dollar value but are not paid directly to the employee.

Income taxes for imputed earnings may not be withheld from the employee’s paycheck, but the earnings may be reported as income subject to tax and the tax paid at tax return time.

Pre–Tax Deductions

Pre–Tax Deductions reduce the gross subject to tax before tax is calculated. Examples include Deferred Comp 401k plans and Health Care 125 plans.

However, not all taxes may be reduced by every pre–tax deduction. For example, for FIT gross is reduced by 401k and 125 deductions, but for

Social Security Tax, gross is only reduced by 125 deductions. Similarly, rules for the various state taxes vary from state to state.

> **Note:** For the production–3 release, the Section 125 balances accumulate only the Health Care 125 values, NOT Dependent Care 125. There is no overall ”Section 125” balance.

Taxes

Oracle Payroll supports both income and limit taxes. Income taxes have no maximum limit. Limit taxes apply to earnings up to a maximum limit after which all successive income is not taxable.

Oracle Payroll also supports both employee and employer taxes, indicating which party is liable for the tax. For example, Social Security has both an employee and employer component.

Income Taxes

All income taxes are paid fully by the employee. Income taxes follow:

- Federal Income Tax (FIT)
- State Income Tax (SIT)
- County Income Tax (COUNTY)
- City Income Tax (CITY)
- School District Tax (SCHOOL)

Limit Taxes

Oracle Payroll supports the following limit taxes:

- Social Security Tax (FICA)
- Federal Unemployment (FUTA)
- State Unemployment Insurance (SUI)
- State Disability Insurance (SDI)

FICA and Medicare have both employee and employer components. FUTA is purely an employer tax. SUI and SDI may have both employee and employer components (or may not exist) depending on the state.

Other Taxes

Head Tax (HT) does not fall into the above categories since it is paid as a flat amount by the employee on a periodic basis.

Tax Credits

Earned Income Credit (EIC) is supported as the companion tax credit to FIT. The taxability rules for EIC are held separately from those for FIT, but they are (and should remain) identical.

Tax Rules

The term Tax Rules refers to the set of filing information for a given tax. One set of information for each relevant taxing jurisdiction is required, at the individual and assignment level.

Employee Tax Rules Form

The Employee Tax Rules form is used to default and then maintain tax information for an employee's assignment. Tax information is maintained at the Federal, State and Local levels.

Maintaining Tax Information For Government Reporting Entities

You use the Define Organization form for this purpose. For the GRE in question, you select the organization classification Government Reporting Entity. Then Others button shows a list of the various contexts defined for the Org Developer DF.

Choosing State Tax Rules allows you to define additional organization information per state such as SUI Company State ID, SIT Company State ID, SUI Self Adjust Method, SDI Self Adjust Method, SUI ER Experience Rate 1, and SUI ER Experience Rate 2, for example.

Taxability Rules

As indicated above, Oracle Payroll maintains rules for deciding whether a particular supplemental or imputed earning is taxable for a given tax, and whether a pretax deduction may reduce taxable gross for a given tax. This information is stored in the table PAY_TAXABILITY_RULES. Given an element classification, its category, and the tax in question (and the locality if necessary) a row in the table means that the earning is taxable, or that the deduction may *not* reduce taxable gross. Regular

earnings are not included, since they are always subject to tax, so no table lookup is necessary.

Considering earnings only, for limit taxes, a row in the table means taxable.

However, for income taxes, there are two kinds of taxability:

- Subject–and–Witholdable
- Subject–and–Not–Withholdable

Subject–and–Withholdable means that tax is withheld on payment of the earning. Subject–and–Not–Witholdable means that the earning is taxable, but tax is not withheld, so it must be paid at the end of the year. Imputed earnings, for example, can be Subject–and–Not–Withholdable for FIT, but are either Subject–and–Witholdable or not subject to FUTA.

Earnings Accumulation Rules

The obvious prerequisite for calculating tax is finding the gross applicable to the tax, the value before applying taxability rules on the earnings components and before applying pretax deductions. For the non–federal taxes, in those cases where an employee works and lives in multiple states, the rules for distributing earnings are not trivial. This section outlines the rules followed in the tax system.

For each employee, tax filing information such as Primary Work Location, Resident Location, and SUI State has been set up.

Federal

For federal taxes, all earnings regardless of where they were earned are included in the gross figure.

SDI

In any given period (in which the Primary Work Location is unchanged), all earnings are considered gross for SDI in the Primary Work Location only, and nowhere else. For example, if John Doe lives in New York for the first six months of 1995, and in New Jersey for the second six months, then he will pay SDI tax to New York only on his first six months' pay, and will pay New Jersey SDI on his second six months pay.

SUI

SUI works similarly to SDI, except that it is referenced to the employee's SUI State (which is often the Primary Work Location). An assignment's SUI State is designated on the Employee Tax Rules Form, and is defaulted to their work state.

In any given period (in which the SUI State is unchanged), all earnings are considered gross for SUI in the SUI State only, and nowhere else.

Non–federal Income Taxes

For the non–federal income taxes, we need a method of apportioning earnings among the states where the employee works and lives. For this purpose, tagged earnings (which explicitly indicate where they were earned) are treated differently from untagged earnings. The following steps are used find the applicable gross for one of these taxes (SIT, County Tax, City Tax) in a given jurisdiction.

1. If an employee's Resident Location is contained within the jurisdiction for which tax is to be calculated, then all the employee's earnings, regardless of where they were earned, are considered as gross, and the following steps below are disregarded.
2. All supplemental earnings are considered to be implicity tagged in the Primary Work Location. Commission is an exception; it is considered untagged and does follow the percentage scaling rule in step 3.
3. All untagged earnings are scaled by the percentage of time the employee spent working in the given jurisdiction. For example, if John Doe worked 30% in California, and 70% in Texas, and has a Salary of 1000, then he would have a gross of 300 for California state tax, and 700 for Texas.
4. All tagged earnings which are contained within the given jurisdiction are included to give a final value.

The special status of the Resident Location is explained as follows. Some states enter into reciprocity agreements with other states. This fact means that if John Doe works in state A and lives in state B, ordinarily, he is taxed in both states for the amount he earned in the work state, A. A reciprocation agreement between state A and state B would specify that the earnings would be taxed in just one state, say the employee's resident state. (To get the tax exemption from this type of agreement, the employee would need a nonresident certificate.)

In a more complex scenario, an employee works in five states. In this case, one considers each work state as part of a pair with the resident state, since reciprocity agreements are not relevant between work states, only between work states and the resident state. Taxes are then calculated independently between each pair of states. Thus, all the employee's earnings are potentially subject to tax in that single resident state. Each work state can either tax the earnings in the state or choose not to do so, but ulinke the resident state, a work state cannot tax earnings from other states.

Deductions

While apportioning the earnings among states, we must also split up the pre–tax deduction amounts as well. The following rule suffices. Within any given time period, for a given tax, total deductions are scaled by the applicable gross for the tax in the relevant jurisdiction, divided by the employee's total gross earnings. For example, if John Doe's total gross was $2000, and his gross for SIT in Colorado was $1500, and he had a total of $200 of 401k deductions, then the portion of his deductions that would apply to SIT in Colorado would be 200 * 1500 / 2000 = $150.

Example

Consider the following situation. The employee lives in Oregon in January, but moves to California in February. He works 50% in California and 50% in Nevada. His primary Work State is California in January and Nevada in February.

Earnings for January and February follow:

1000 Salary

100 Timecard, tagged in Nevada

100 Bonus

100 Commission

———

1300 = Total

Deductions for January and February follow:

100 401(k)

For January:

FIT Gross = 1300

Deductions = 100

SDI:

In California, Gross = 1300 since California is the primary work state.

Deductions = 100

In Nevada, Gross = 0 ”

Deductions = 0

SIT:

In California, Gross: 500 for the California half of salary

100 for bonus (supp earning) in primary work state (California)

50 for the California half of commission

650 = Total

Deductions: 50 for the CA portion of 401 (100 * 650 / 1300)

In NV, Gross: 500 for the Nevada half of salary

100 for the Nevada–tagged timecard earnings

50 for the Nevada half of commission

650 = Total

Deductions: 50 for the NV portion of 401 (100 * 650 / 1300)

For Feb:

FIT Gross = 1300

Deductions = 100

SDI:

In California, Gross = 0 since Nevada is the primary work state.

Deductions = 0

In Nevada, Gross = 1300 ”

Deductions = 100

SIT:

In California, Gross = 1300 due to California = resident state

Deductions: 100 since CA = resident state (100 * 1300 / 1300)

(Since there isn't a reciprocity rule between CA and NV,

NV earnings are reported in CA as gross, but not taxed)

In Nevada, Gross: 500 for the NV half of salary

100 for bonus in primary work state (Nevada)

100 for the Nevada–tagged timecard earnings

50 for the Nevada half of commission

___

750 = Total

Deductions: 57.69 for the NV portion of 401 (100 * 750 / 1300)

Year–to–date:

FIT: Gross = 2600, Deductions = 200

SDI:

In CA, Gross = 1300, Deductions = 100

In NV, Gross = 1300, Deductions = 100

SIT:

In CA, Gross = 1950, Deductions = 150

In NV, Gross = 1400, Deductions = 107.69

Tax Balances

The starting gross is converted by applying taxability rules and pre–tax deductions, involving several steps and intermediate values. This process results in the final amount on which tax is calculated. These intermediate values (tax balances) are also useful for reporting and other purposes.

Definitions

The various tax balances are defined as follows:

- "Gross" is the gross earnings to which the tax applies in the given jurisdiction.

- ”Subject Earnings” is the portion of gross subject to the tax. This value is equal to Subject Withholdable + Subject Not Withholdable.
- ”Subject Withholdable” is the portion of gross earnings subject to the tax and withholdable.
- ”Subject Not Withholdable” is the portion of the gross earnings that are subject to the tax but not withholdable. This figure is applicable only to income taxes.
- ”Exempt” is the portion of gross earnings not subject to the tax (that is Exempt + Subject Withholdable + Subject Not Withholdable = Gross.)
- ”401k / 125 / Dependent Care Reductions” refers to the 401k / 125 / Dependent Care amounts applicable as pretax reductions on the gross for the given tax. A pretax reduction is the amount of the pretax deduction that is allowed to reduce gross according to taxability rules.
- Section 125 refers to only Health Care 125.
- Remember that ”reductions” does not mean ”deductions.”
- ”Reduced Subject Withholdable” is the Subject Withholdable less the pretax reductions. This value is the final amount on which tax is calculated for income taxes.
- For limit taxes only, ”excess” is the amount of the subject over the tax limit.
- For limit taxes only, ”taxable” is the Reduced Subj Witholdable – Excess. This value is the final amount to which tax is applied for limit taxes.

  Note: Notice the limitations on which tax balances are valid for the type of tax, limit, or income. In the case of no tax rules (as in the case of a state with no income tax), the above tax balances are meaningless, and thus should all be zero.

Tax Balances Form

The Tax Balances Form displays a summary of an employee’s tax balances for all the various taxes.

Tax Balance API

This function provides a simple single function call method for obtaining the above tax balances. It calls the core balance user exit. For example, in the package PAY_US_TAX_BALS_PKG:

```
FUNCTION  us_tax_balance (p_tax_balance_category  in varchar2,
           p_tax_type     in varchar2,
           p_ee_or_er     in varchar2,
           p_time_type    in varchar2,
           p_asg_type     in varchar2,
           p_gre_id_context   in number,
           p_jd_context       in varchar2  DEFAULT NULL,
           p_assignment_action_id  in number    DEFAULT NULL,
           p_assignment_id      in number DEFAULT NULL,
           p_virtual_date    in date      DEFAULT NULL,
           p_payroll_action_id in number  DEFAULT NULL)
RETURN number;
```

The allowed tax_balance_category values follow:

- GROSS
- SUBJ_WHABLE
- SUBJ_NWHABLE
- SUBJECT
- EXEMPT
- 401_REDNS
- 125_REDNS
- DEP_CARE_REDNS
- REDUCED_SUBJ_WHABLE
- TAXABLE
- EXCESS
- WITHHELD (may be used for any, and will translate to LIABILITY or ADVANCED as necessary)
- ADVANCED (only used for EIC)
- LIABILITY (only for ER taxes)

These values are in the lookup type US_TAX_BALANCE_CATEGORY.

The allowed tax types follow:

- FIT

- FICA*
- FUTA
- MEDICARE*
- EIC
- SIT
- SUI*
- SDI*
- CITY
- COUNTY
- SCHOOL
- HT

The asterisk (*) indicates that the type requires a value EE or ER, denoting either the employee or employer tax in the parameter p_ee_or_er. FUTA and HT are ER only, and the rest are EE only. The API returns an error if an inconsistent value is entered. These tax types are in the lookup type US_TAX_TYPE.

The allowed time types follow:

- RUN
- PTD
- MONTH
- QTD
- YTD

The allowed asg types follow:

- ASG (assignment level)
- PER (person level)
- GRE (entire GRE level).

For ASG and PER, the GRE is implicit, so these types return only those values that are associated with the GRE identified by p_gre_id_context. These values are permitted in the following combinations:

	RUN	PTD	MONTH	QTD	YTD
ASG	x	x	x	x	x
PER	x		x	x	x
GRE	x		x	x	x

The tax unit id is p_gre_id_context and is mandatory. p_jd_context should contain the *xx–xxx–xxxx* format for the Vertex geocode. It is ignored for the federal taxes.

For school district, the five–digit code may be appended, as follows: *xx–xxx–xxxx–xxxxx*, or it may be appended after the state code, as follows: xx–xxxxx

Two modes of access are allowed (as in the core balance user exit):

- Assignment Action Mode (for which only the assignment_action_id is needed)
- Date Mode (for which an assignment_id and a virtual_date are used)

All three parameters are provided for, and should be set NULL when not used. Assignment action mode is required for all RUN level inquiries. However, for an asg type of GRE, we can only use Date Mode, and without an assignment_id, so the virtual_date must be set. Also, for GRE, we must provide the payroll_action_id parameter for RUN level balances.

Database design

Instead of hardcoded balances and dimensions in the report definition, the user can define the set of tax balances and the set of dimensions to report on.

The tax report name is stored in hr_lookups (lookup_type = 'US_TAX_REPORT').

Defining a New Tax Balance Report

To define a new tax balance report, perform the following steps:

1. Add a new lookup code in hr_lookups for lookup_type = US_TAX_REPORT.

   The MEANING column is used as the report title. This lookup_type is the primary key for the dimension and balances tables such as

STATE_QTD. (At runtime, the user selects from the US_TAX_REPORTs, thereby identifying the set of balances and dimensions.)

2. For each dimension to be reported, insert a row in pay_us_tax_report_dimensions.

   Valid time dimension values are lookup_codes with US_TAX_BALANCE_DIMENSION lookup_type, such as report_code = STATE_QTD, dimension_code = QTD.

3. Select the subset of balances from pay_us_tax_balances. Insert a row for each tax_balance_id in pay_us_tax_balances.

   pay_us_tax_report_dimensions stores the dimensions to be used for the report. It includes the following lookup codes:

   - REPORT_CODE: report lookup code
   - DIMENSION_CODE: dimension lookup code

   pay_us_tax_types include the following:

   - TAX_TYPE_ID
   - EE_ER_CODE: indicates whether the tax is EE, ER or ”EE and ER”

     Do not use EE_ER_CODE when calling the tax balance api; it is used for informational purposes only.
   - LIMIT_TAX_FLAG: indicates whether or not the tax is a limit tax.
   - TAX_DOMAIN_CODE: indicates whether the tax is Federal, State or Locality
   - TAX_TYPE_CODE: indicates the tax type lookup code (US_TAX_TYPE)

   pay_us_tax_balances include the following:

   - TAX_BALANCE_ID
   - TAX_TYPE_ID: FK to pay_us_tax_types
   - BALANCE_CATEGORY_CODE: balance category lookup code (US_TAX_BALANCE_CATEGORY)
   - EE_OR_ER_CODE: indicates whether the tax is EE or ER for the category.
   - USER_REPORTING_NAME: used for reporting pay_us_tax_report_balances
   - REPORT_CODE: report lookup code

- TAX_BALANCE_ID: FK to pay_us_tax_balances
- BALANCE_PRINT_SEQUENCE: used for determining the relative order to print the balance for the report.

Other Forms

Statement of Earnings

The Statement of Earnings form provides a summary of an assignment's earnings, deductions, and taxes, as produced by the payroll run process. It can be reached from the Assignment Form, in which case it displays the results of the most recent run, or from the Assignment Actions Form, in which case it shows the results of the desired assignment action.

Assignment Run Results

For debugging purposes, you may need to view the raw run results, not in the processed format shown in the statement of earnings. The Assignment Run Results Form lists the actual run results and their run result values. For example, it shows in the *xx–xxx–xxxx* format the raw geocodes in the jurisdiction run result values.

Tax Implementation

This section provides an overview of how the tax system for Oracle Payroll operates. Subsequent sections detail the various components, including those used to calculate taxes, and those used to gather the results.

The Vertex Tax Calculation System

A third party tax calculation system, provided by Vertex Inc., has been chosen for use in Oracle Payroll. Vertex is the leading vendor for US tax calculation, providing a COBOL tax calculation function and a research / support infrastructure to ensure that taxes at user sites are always correctly calculated using up–to–date changes in tax rates and limits. Vertex will soon provide a new version of their calculation module, written in C. Oracle Payroll takes advantage of the many and varied calculation rules and parameters offered in the Vertex tax

calculation module. Oracle also includes a layer of additional functionality to address areas of extended requirements, as follows.

- Vertex does not hold earnings taxability rules, but does hold rules for pretax deductions. For the sake of consistency, Oracle Payroll takes all taxability rules into account (for both earnings and deductions) and merely passes the final gross (after pretax deductions) to be taxed upon into the Vertex subsystem.
- Vertex only calculates values for one work and one resident location at a time. Oracle Payroll therefore iterates calls to the Vertex subsystem for multiple work state situations.

The interface that Vertex provided is a single chunk of memory organized as a structure of records, and serves for both input and output values. The Oracle Payroll tax system gathers the information Vertex needs into this ”link area,” calls the Vertex COBOL code, and then extracts the results, converting them into the familiar run results. Generally, inputs are comprised of tax filing information (filing status and number of exemptions, for example) and the gross amounts. Vertex returns tax codes and the tax amounts.

Vertex Elements

Calculation of tax for an assignment is triggered off by a special element type called ”Vertex.” The Assignment Tax Rules Form creates Element entries of this element type, one for each jurisdiction in which the assignment has registered for tax, and at all levels (state, city, and county). The two most important input values for this element are Percentage and Jurisdiction (called JD for short).

The Jurisdiction input value is set to the state or locality jurisdiction code as appropriate. The jurisdiction code is determined from the location in question, whether it be the employee’s resident address, work location, or subsequent state and locality records entered in the form.

The Percentage input value is set equal to the Remainder Percent for the State and Time In Locality for the Locality. The State’s remainder percent is the time in state less the total of the time in localities for the particular state.

In other words, the Percentage input value contains the percentage of time that the assignment spends in the jurisdiction solely at that level. This percentage different from the percentages the user enters in the Tax Rules Form.

In the form, a user might enter 40 percent for California, and then 10percent for Mountain View. The Vertex entry created for Mountain

View will have 10 for its input value, but California will have an input value of 30 (the percentage of time spent purely at the state level; this percentage represents the remainder of time not spent in any specific city or county in California). This approach makes sense since the 10 percent in the Mountain View Vertex entry will produce California state tax results, and when the 30 percent is processed, the tax on the other 30 percent will be produced, accounting for the full 40 percent. As a consequence, all the percentages of all the Vertex entries will always sum to 100 percent.

The Jurisdiction input value is used to set up the US–specific "Jurisdiction" context in the payroll run (using the localization C hook). The Jurisdiction input value serves as the reference for all database items in the formula attached to the Vertex element. However, the processing required for setting up Vertex involves database items requiring several different jurisdiction contexts (the primary work location, the resident location, and the SUI state).

Vertex Formulas

In order to enable using multiple contexts when processing a Vertex element entry, a chain of elements is used; the Vertex element spawns a different element as an indirect result, passing it a new jurisdiction context. In the chain of elements appears as follows, each vertical arrow indicates an indirect result and a horizontal arrow indicates formula results without result rules.

(JD = primary work location)

|

VERTEX —> FIT / FSP / EIC inputs to Vertex (put into the link area)

|

V

VERTEX2 —> FICA / FUTA / Medicare inputs to Vertex

|

V

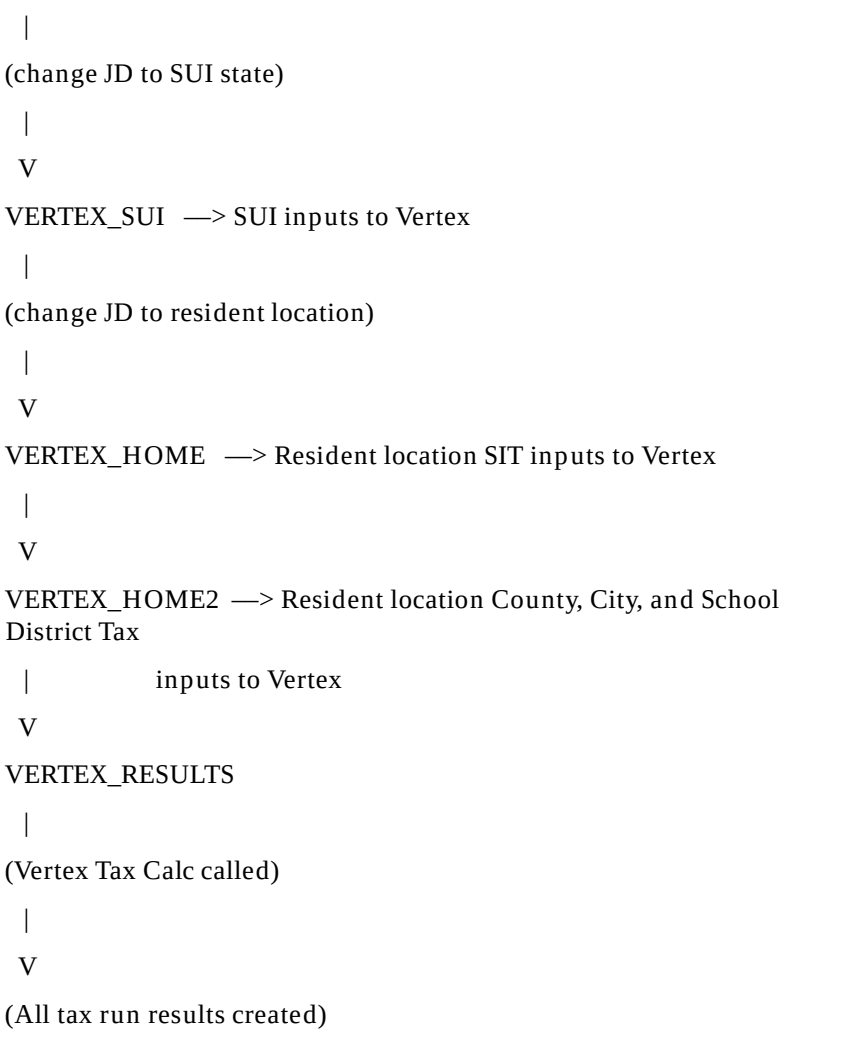
VERTEX_WORK —> Work location SIT and SDI inputs to Vertex

|

V

VERTEX_WORK2 —> Work location County, City, Head Tax and School District Tax

| inputs to Vertex



Each Vertex element is associated with a specific formula (with the same name as the element), which serves to set up the various inputs to the Vertex tax calculation. Additionally, some direct results are created to store some useful results for safekeeping. Multiple elements are used under the same jurisdiction, because the formulas are limited in size and had to be split up. Because database items are cached in memory for use across formulas, this splitting causes minimal performance impact.

The above diagram indicates that after each formula is run, its results get dumped into the Vertex link area. This is accomplished by a C hook that, when the element's name takes the form VERTEX%, scans the formula results to identify those meant to be passed to the Vertex calc.

The formula results are then passed into the link area (which is why they do not require proper formula result rules).

For VERTEX_RESULTS, the hook takes the additional step of calling the Vertex Cobol code, and after the calculation is complete, transfers the results out of the link area into the formula results of VERTEX_RESULTS. Formula result rules then take over to produce the final tax run results.

This processing is repeated once for every Vertex entry (each work jurisdiction) belonging to the assignment. A skip rule is used on the VERTEX element to avoid processing overhead when the gross is zero for the jurisdiction. Since multiple Vertex entries may be processed, but federal taxes must only be calculated once, the formulas check whether the jurisdiction on the Vertex entry is the same as the employee's primary work location. Since this can only occur once, federal taxes are calculated at that time, and skipped otherwise.

Processing Priority

An important feature of the tax elements, including the VERTEX% elements, is that all share the same processing priority (4250), and the VERTEX indirect results use a new formula result rule type ("order indirect") to ensure that the entire above chain is completely processed before another chain (headed by a different Vertex entry) is begun. This prohibits the processing of two different Vertex entries from interleaving, which if allowed, can cause subtle problems due to the nature of the dependencies between successive Vertex formula processing chains. (For example, in the above scenario, when the California Vertex entry is started, the Mountain View processing chain must have finished. Its SIT tax result can then be included in the SIT_YTD balance accessed in the California Vertex entry processing.)

Tax Elements

Tax Deduction Elements

Tax deduction elements are categorized as follows:

- The federal tax deduction elements are FIT, EIC, FUTA, Medicare_EE, Medicare_ER, SS_EE, SS_ER
- The state tax deduction elements are SIT_WK, SIT_RS, SDI_EE, SDI_ER, SUI_EE, SUI_ER
- The county tax deduction elements are County_SC_WK, County_SC_RS, County_WK, County_RS

- The city tax deduction elements are City_SC_WK, City_SC_RS, City_WK, City_RS, City_HT_WK

The county and city taxes (which are income taxes) have two input values each: Pay Value (which contains the tax amount) and Jurisdiction.

The SDI and SUI elements have three input values: Pay Value, TAXABLE (which contains the portion of the gross that was not over the limit), and Jurisdiction.

The SIT elements have three input values: Pay Value, Supp Tax (which contains the portion of the Pay Value tax amount that was derived purely from the supplemental earnings), and Jurisdiction.

EIC has only Pay Value. FIT has only Pay Value and Supp Tax. FUTA, Medicare, and FICA have Pay Value and TAXABLE.

Naturally, the jurisdiction value matches the level of the tax (such as a state geocode for a state–level tax and a city geocode for a city–level tax). However, for the school district taxes, the jurisdiction value used is *state code–school district code*.

The reason there are WK (work) and RS (resident) element pairs of some tax types is that the VERTEX_RESULTS formula outputs two results for each such tax, since both work and resident jurisdictions may tax the earnings. A formula cannot produce two results of the same element type. Thus, every non–federal income tax deduction element must be split into two versions, each identical (even in their balance feeds) except for name.

Subject Elements

The SUBJECT Elements (classification of Information, with category of Tax Balance) are special elements that store results the Tax Balance API uses. Because the earnings accumulations rules for non–federal taxes are complex, the tax balance values are not derived directly from the earnings and deductions results. Consequently, the run–level values of each non–derived non–federal tax balance are retained as run results. These results are direct results from the appropriate VERTEX formulas.

The SUBJECT Elements follow: City_SUBJECT_RS, City_SUBJECT_WK, County_SUBJECT_RS, County_SUBJECT_WK, SIT_SUBJECT_RS, and SIT_SUBJECT_WK, School_SUBJECT_RS, and School_SUBJECT_WK.

These SUBJECT elements have the following input values: Gross, Subj Whable, Subj NWhable, DC 401 Redns, S 125 Redns, Dep Care Redns.

The SDI_SUBJECT_EE, SDI_SUBJECT_ER, SUI_SUBJECT_EE, SUI_SUBJECT_ER elements have these input values: Gross, Subj

Whable, DC 401 Redns, S 125 Redns, Dep Care Redns, Jurisdiction (this reflects the fact that limit taxes don't have not–withholdable rules).

Additionally, the FSP_SUBJECT element has the input value Reduced Subj Whable. This input value is used only internally as an input for Vertex.

SUBJECT elements for the federal taxes are not necessary because the federal tax balances can easily be derived directly from the earnings and deductions results through the SUBJECT dimensions.

Run Result Suppression

Because of the potential of each assignment creating more than 40 run results (and over 100 run result values) for every run, and the performance degradation that would ensue, some means to prevent unnecessary run results was required. The core mechanism does not easily allow for the conditional creation of run results. The average assignment will only have 10 or so useful run results (since many of the possible taxes and situations catered for are not very common).

The solution was to add code to the localization C hook. This hook serves to set up the non–core (localization) contexts, such as Jurisdiction and Government Reporting Entity for use in formulas. It is called for each element as it is processed, and allows a check for whether the result is unnecessary. In this case a special flag is set to disable its creation on the database. This portion of C code is called the "run result suppressor"; the rules it follows to strip out useless run results follow:

1. If the element name is like VERTEX% then suppress it.

   (Such results are always useless; the element exists only to trigger processing, not to store information.)

2. If the element is a non–federal tax deduction and its value is zero, then suppress it.

   This suppression is appropriate since the balance mechanism treats the absence of run results as zero, and no code is dependent on the existence of tax run results.

3. If the element is a SUBJECT element, then check if the return code indicates that the corresponding tax does not exist, in which case suppress it.

   This suppression prevents the big SUBJECT results from being created for jurisdictions that do not even have tax, such as the state income tax for Texas, or the city tax for Mountain View.

Balances

Naturally, all the tax deduction and SUBJECT elements each have their own primary balances. There is one balance for each of the SUBJECT elements' input values (except Jurisdiction). The earnings and deductions elements also have their own primary balances.

Some of the important composite balances follow:

- Gross Earnings (includes all earnings: regular, supplemental, and imputed)
- Regular Earnings (includes all elements of classification Earnings)
- Supplemental Earnings (includes classifications of Supplemental Earnings and Imputed Earnings)
- Imputed Earnings (includes all Imputed Earnings)
- Pre Tax Deductions (includes all elements of classification Pre–Tax Deductions)
- Tax Deductions (includes all elements of the Tax Deductions classification)
- Deductions (includes Involuntary Deductions and Voluntary Deductions)
- Net (this is equivalent to all earnings (except imputed) + tax credits – all deductions)

The following balances are special in that they are fed by elements, not on the basis of their classification, but by their classification and category. This process is accomplished using a PL/SQL procedure called the "category feeder"that is called by the earnings and deductions template procedures to set up any necessary feeds. The package containing the feeder procedure is called pay_us_cgty_feeds_pkg.

- Commissions (fed by "Supplemental Earnings" of category "Commissions")
- Def Comp 401K (fed by "Pre–Tax Deductions" of category "Deferred Comp 401k")
- Dependent Care (fed by "Pre–Tax Deductions" of category "Dependent Care 125")
- Section 125 (fed by "Pre–Tax Deductions" of category "Health Care 125")

  Note: There is no true Section 125 Balance that includes both Health Care 125 and Dependent Care 125 (for the Production 3 release).

For all the above (and many other startup balances), the following dimensions combinations are used: _ASG_RUN, _ASG_PTD, _ASG_MONTH, _ASG_QTD, _ASG_YTD, _ASG_GRE_RUN, _ASG_GRE_PTD, _ASG_GRE_MONTH, _ASG_GRE_QTD, _ASG_GRE_YTD, _PER_RUN, _PER_MONTH, _PER_QTD, _PER_YTD, _PER_GRE_RUN, _PER_GRE_MONTH, _PER_GRE_QTD, and _PER_GRE_YTD, _PAYMENTS.

For balances accessed in the Tax Balance API (including the earnings and pre–tax deductions balances), these GRE–level dimensions are also used: _GRE_RUN, _GRE_MONTH, _GRE_QTD, and _GRE_YTD.

For the non–federal tax deductions, jurisdiction gets involved and then the JD versions of the above dimensions must be used. The JD dimensions include extra SQL to filter out those run results whose jurisdiction result value does not match the given context. Thus, SIT_WITHHELD_ASG_JD_YTD, with the JD contexts et to 05–000–0000 (which is California), will return only California state tax withheld.

Jurisdiction Level

In order that the JD dimensions may be used at different levels (state, county, and city), some method of knowing how strictly to match the run result and context JDs is needed. For example, it calculate ”all earnings earned within the state of California,” all earnings tagged like 07–234–2893 or 07–484–3497 would be included, thus matching the first two digits. But for ”all earnings earned within California, but not in any city or county,” only tags that exactly match ”07–000–0000” would be included.

Creating the jurisdiction level and adding it to the balance type as a column allows for this calculation. It contains a code that specifies how many characters of the jurisdiction code (starting from the left) must match to be included in the balance sum. The allowed values are 0 (federal), 2 (state), 6 (county), 11 (city). Putting this information on the balance type prevents the number of dimensions from multiplying by 4 (since separate, federal, state, county, and city level dimensions would be required). However, the side effect is a replication of some balances into four (to maintain the number of necessary defined balances).

For example, four versions of the Regular Earnings balance are exactly the same, except that they differ in jurisdiction level. These versions follow:

- Regular Earnings (jd level of 0)
- Regular Earnings State (jd level of 2)
- Regular Earnings County (jd level of 6)

- Regular Earnings City (jjd level of 11)

Those balances whose jurisdiction level is 0 are required to use the non–JD dimensions and those whose jd level is not 0 are required to use the JD dimensions. For example, the defined balance REGULAR_EARNINGS_ASG_RUN returns all regular earnings in the run, but REGULAR_EARNINGS_CITY_ASG_JD_RUN (with the JD context set to Houston) returns all those earnings that are tagged, and whose entire eleven character code matches Houston's. Regular Earnings is the only balance that is replicated in this way, because it has the unique property of including tagged and untagged elements. The State and County versions are unused, and the City version is used to get the tagged portion for Step 4 of the Earnings Accumulation Rules for non–federal taxes.

The balance dimension of _DEFAULT_ASG_RUN exists as a special case; it includes only those run results that are not tagged. This balance dimension is used only in conjunction with the Regular Earnings balance, in order to implement Step 3 of the Earnings Accumulation Rules for non–federal taxes.

The jurisdiction code placed on the school district results is *state code–school district code*. Since the school district code is five digits long, the jurisdiction level is 8. The only place such a code may appear is on a school district tax result; earnings may not be tagged in such a jurisdiction.

Tax Type and the Subject Dimensions

The Subject dimensions were created to allow filtering of run results according to taxability rules. These dimensions are instrumental for obtaining the subject withholdable and pretax reductions kinds of tax balances.

A design tradeoff similar to the jurisdiction design tradeoff was encountered; tax type is a new context introduced for the subject dimensions, but it would potentially force the creation of a whole set of dimensions for each tax type (of which there are ten). Instead, as for jurisdiction, it was decided to place the tax type context on the balance type, forcing a replication of balance types, though limited only to a small set.

Allowd tax type values are based on the lookup_codes from US_TAX_TYPE). These values follow: FIT, MEDICARE, FICA, FUTA, EIC, NW_FIT, SDI, SUI, SIT, and NW_SIT. (As mentioned before, city and county level taxes use the SIT tax type.)

Balances used with the subject dimensions are fed by elements for which taxability rules exist. These elements follow:

- Supplemental Earnings
- Def Comp 401K
- Section 125
- Dependent Care
- Commissions

The first four balances are used for composing tax balances. Ten versions of each balance appear, with names in the form *old name* for *tax type*: Supplemental Earnings for FIT, Supplemental Earnings for NW_FIT, and Supplemental Earnings for SDI, for example.

Commissions is used only internally, so only two versions exist. The jurisdiction level on the balances matches the tax type.

When combined with a subject dimension, these tax types indicate the tax to which the taxability rules refers. For example, when using Supplemental Earnings for SDI with _SUBJECT_TO_TAX_JD_ASG_GRE_RUN and the JD context set to California, we get all supplemental earnings which are subject to California SDI. For income taxes, the FIT or SIT tax type returns the subject and withholdable amount, and the NW_FIT or NW_SIT returns the subject and not the withholdable amount. (For deductions, this distinction is meaningless and NW_FIT / NW_SIT is not used; FIT / SIT tax types return the amount of pretax deduction that is not allowed to reduce gross.)

The following subject dimensions are currently used:

- _SUBJECT_TO_TAX_ASG_GRE_RUN

  This subject dimension is used with all the ”for FIT / FUTA / FICA / MEDICARE / EIC / NWFIT” balances.
- _SUBJECT_TO_TAX_ASG_GRE_PTD
- _SUBJECT_TO_TAX_ASG_GRE_MONTH
- _SUBJECT_TO_TAX_ASG_GRE_QTD
- _SUBJECT_TO_TAX_ASG_GRE_YTD
- _SUBJECT_TO_TAX_PER_GRE_RUN
- _SUBJECT_TO_TAX_PER_GRE_MONTH
- _SUBJECT_TO_TAX_PER_GRE_QTD
- _SUBJECT_TO_TAX_PER_GRE_YTD
- _SUBJECT_TO_TAX_GRE_RUN

- _SUBJECT_TO_TAX_GRE_MONTH
- _SUBJECT_TO_TAX_GRE_QTD
- _SUBJECT_TO_TAX_GRE_YTD
- _SUBJECT_TO_TAX_JD_ASG_GRE_RUN

  This subject dimension is used with the ”for SIT / NWSIT / SDI / SUI” balances.

These merely extend the basic balance dimensions with the taxability rules filtering for each run result, its classification and category areretrieved from the parent element type, referenced into the taxability rulestable along with the tax type from the balance type. If a row is found inthe taxability rules table, then the result is included in the sum. Hencethe relationship between the meaning of the existence of a row in the taxability rules table and the value returned by the subject dimensions.

The reason only one JD subject dimension exists is that only the ASG_GRE_RUN level dimension is needed.

Tax Balance API Implementation Details

Federal taxes are obtained directly from the earnings and deductions results. Given a balance dimension (*BD*), and a reference tax (*TAX*), the defined balances are as follows:

- Gross: GROSS_EARNINGS_*BD*
- Subj Whable: REGULAR_EARNINGS_*BD* + SUPPLEMENTAL_EARNINGS_FOR_*TAX*_SUBJECT_TO_TAX_*BD*
- Subj NWhable: SUPPLEMENTAL_EARNINGS_FOR_NW*TAX*_SUBJECT_TO_TAX_*BD*
- 401 Reductions: DEF_COMP_401K_*BD* – DEF_COMP_401K_FOR_*TAX*_SUBJECT_TO_TAX_*BD*
- 125 Reductions: SECTION_125_*BD* – SECTION_125_FOR_*TAX*_SUBJECT_TO_TAX_*BD*
- Dep Care Reductions: DEPENDENT_CARE_*BD* – DEPENDENT_CARE_FOR_*TAX*_SUBJECT_TO_TAX_*BD*
- Taxable: *TAX*_TAXABLE_*BD*

Other federal balances derived from the above balances follow:

- Subject Earnings = Subj Whable + Subj NWhable

- Exempt = Gross – Subject Earnings
- Reduced Subj Whable = Subj Whable – 401 Reductions – 125 Reductions – Dep Care Reductions
- Excess = Reduced Subj Whable – Taxable

For the nonfederal taxes, we use the balances associated with each of the subject element input values, which store the run–level amounts for the basic (non–derived) tax balances:

- Gross: *TAX*_GROSS_*BD*_JD
- Subj Whable: *TAX*_SUBJ_WHABLE_*BD*_JD
- Subj NWhable: *TAX*_SUBJ_NWHABLE_*BD*_JD
- 401 Reductions: *TAX*_401_REDNS_*BD*_JD
- 125 Reductions: *TAX*_125_REDNS_*BD*_JD
- Dep Care Reductions: *TAX*_DEP_CARE_REDNS_*BD*_JD
- Taxable: *TAX*_TAXABLE_*BD*_JD

Other nonfederal balances derived from the above nonfederal balances follow:

- Subject Earnings = Subj Whable + Subj NWhable
- Exempt = Gross – Subject Earnings
- Reduced Subj Whable = Subj Whable – 401 Reductions – 125 Reductions – Dep Care Reductions
- Excess = Reduced Subj Whable – Taxable

Employee Tax Rules Form

Form Processing

When a user enters the Employee Tax Rules Form for the first time, a substantial amount ofprocessing occurs behind the scenes. Briefly, Oracle Payroll performs the following processing:

1. Retrieves the employees work and resident states.
2. Defaults a Federal tax record for the employee assignment.
3. Defaults a Workers Compensation element entry for the employee assignment.
4. Defaults appropriate State tax record(s) for the employee assignment.

5. Defaults the VERTEX element entry for the state jurisdiction for the employee assignment.
6. Defaults Locality tax records for the employee assignment.
7. Defaults the VERTEX element entry for the local jurisdiction for the employee assignment.

Data Structure

The actual tax information entered in this form is stored on the Further Assignment Information Developer Descriptive Flex on the table PER_ASSIGNMENT_EXTRA_INFO. This table is not date tracked.

The AEI_INFORMATION_CATEGORY & INFORMATION_TYPE are set to FEDERAL, STATE, or LOCALITY, depending on the type of tax information being recorded.

Inserting the Federal Default Tax Rule

There is only one federal tax rule per employee assignment. The SUI state defaults to the employee's work or location state.

Filing status = select lookup_code
from hr_lookups
where lookup_type = US_FIT_FILING_STATUS
and upper(meaning) = SINGLE;

EIC Filing Status = select lookup_code
from hr_lookups
where lookup_type = US_EIC_FILING_STATUS
and upper(meaning) = NO EIC;

Relevant segments for federal rules are;

Segment	Name	Default
aei_information1	Withholding Allowances	0
aei_information2	Additional WA Amount	0
aei_information3	Allowances Reject Date	NULL
aei_information4	Filing Status Code	see above
aei_information5	FIT Additional Tax	0
aei_information6	FIT Exempt	N

aei_information7	Medicare Tax Exempt	N
aei_information8	SS Tax Exempt	N
aei_information9	FUTA Tax Exempt	N
aei_information10	Statutory Employee	N
aei_information11	Cumulative Flag	N
aei_information12	EIC Filing Status Code	see above
aei_information13	W2 Filed Year	NULL
aei_information14	FIT Override Rate	0
aei_information15	FIT Override Amount	0
aei_information18	SUI State Code	see above

Defaulting Workers Compensation Element Entries

One workers compensation element entry is created with a Jurisdiction input value equal to the SUI state.

Inserting the State Default Tax Records

If the work state is not the same as the resident state then one record per state is defaulted. The state code is set appropriate to the work or resident state. Time in state defaults to 100% for the work state. Remainder Percent is 0 for resident state and 100 for the work state. Relevant segments for state rules follow:

Segment	Name	Default
aei_information1	State Code	see above
aei_information2	Filing Status Code	1
aei_information3	State Non Resident Certificate	N
aei_information4	SIT Exempt	N
aei_information5	SDI Exempt	N
aei_information6	SUI Exempt	N
aei_information7	Enabled	Y
aei_information8	SIT Additional Tax	0
aei_information9	Secondary WA	0
aei_information10	Withholding Allowances	0
aei_information11	Additional WA Amount	0

aei_information12	SIT Optional Calc Ind	NULL
aei_information13	Time In State	see above
aei_information14	SIT Override Rate	0
aei_information15	SIT Override Amount	0
aei_information16	Remainder Percent	see above

Inserting Local Tax Records

The state defaults to the state within which the locality exists. The jurisdiction code for both the resident and work addresses are retrieved when the employee's address and location details are verified. If work and resident localities are different, then one record per locality is defaulted.

For the resident locality only, both a county record and a city record are defaulted. The school district code for these resident tax records is also defaulted when one exists for the particular county or city. The resident city's local tax record's time in locality always defaults to 0, unless the work and resident cities are the same. In such cases the city defaults to 100. Thus, the work city always defaults to 100. The resident county's local tax record's time in locality always defaults to 0.

Filing Status Code = select lookup_code

from hr_lookups

where lookup_type = US_LIT_FILING_STATUS

and upper(meaning) = SINGLE;

County School District = SELECT school_dst_code

FROM pay_us_county_school_dsts

WHERE state_code = *STATE_CODE*

AND county_code = *COUNTY_CODE*

ORDER BY TO_NUMBER(school_dst_code);

STATE_CODE and *COUNTY_CODE* are passed as parameters and are the appropriate parts of the 11 character jurisdiction code derived from the employee's resident and work locations as appropriate. The jurisdiction code is in the form 00–000–0000, where the first set of two zeroes would be the state code (see PAY_US_STATES), and the second set of three zeroes would be the county code.

City School District = SELECT school_dst_code

FROM pay_us_city_school_dsts

```
WHERE     state_code  = STATE_CODE
AND    county_code = COUNTY_CODE
AND    city_code = COUNTY_CODE
ORDER BY TO_NUMBER(school_dst_code);
```

COUNTY_CODE as above is derived from the jurisdiction code for the particular locality. The third set of four zeroes would be the county code.

Relevant segments for local rules follow:

Segment	Name	Default
aei_information1	State Code	see above
aei_information2	Jurisdiction Code	see above
aei_information3	Filing Status Code	see above
aei_information4	Withholding Allowances	0
aei_information5	Additional WA Rate	0
aei_information6	LIT Additional Tax	0
aei_information7	LIT Exempt	N
aei_information8	Enabled	Y
aei_information9	School District Code	as above
aei_information10	Time In Locality	as above
aei_information11	LIT Override Rate	0
aei_information12	LIT Override Amount	0

Defaulting VERTEX Element Entries

VERTEX entries are created for both the defaulted state and city tax records with a Jurisdiction input value equal to the state or city jurisdiction code as appropriate. The Percentage input value is set equal to the remainder percent (aei_information16) for the State and Time In Locality (aei_information10) for the Locality.

> **Note:** A Vertex element entry is not created for the defaulted resident county record, but is created for all subsequent county records defined in the form.

Geocode Tables

For the purpose of identifying taxing jurisdictions, Vertex Inc. has defined a numbering scheme called geocodes. When entering employee tax filing information and work and resident locations, the state name / county name / city name information entered by the user must be converted into geocode format to be understood by the Vertex calculation. Each filing jurisdiction (from the Tax Rules Form) is converted into geocode format and placed in the jurisdiction input value of a Vertex element entry. The geocodes derived from the work and resident addresses are obtained through a PL / SQL function called Addr_Val, since the address tables hold only city / county / state names (plus zip code).

The geocode format is *xx–xxx–xxxx,* where the first two digits are the state code, the next three digits are the county code, and the last four digits are the city code, with each code separated by hyphens. The county code refers to ”county within the state,” meaning that the code is meaningful only given the state code. Similarly, the city code identifies the ”city within the state”.

All zeros for a code means that the particular level is ignored. Thus, 44–201–0000 refers to a whole county and no city, 44–000–0000 represents an entire state, and 00–000–0000 is the federal jurisdiction (anywhere in the United States). Note that 44–201–0000 refers to Texas–Harris County, but 18–201–0000 refers to Robertson County in Kentucky. 44–201–1440 represents Texas–Harris County–Houston, but 44–157–1440 means Texas–Fort Bend County–Houston. This approach enables us to identify cities that span multiple counties.

Since counties cannot cross state borders, it is sufficient to identify them by a unique number within a state. Identifying cities is a different matter. Cities can cross county and state boundaries, so the proper scheme is to use a universally unique code. Vertex does not use this approach, so we find that 04–091–0590 and 44–037–2950 represent two parts of the city of Texarkana, which lies partly in Arkansas, and partly in Texas. by looking at the geocodes, there is no way to tell that they refer to the same actual city.

This is a limitation of the Vertex geocode scheme and means that Oracle Payroll cannot calculate the ”city tax in all of Texarkana.” The Texas and Arkansas portions must be calculated separately.

A similar problem concerns the format of the geocode and the simplistic matching scheme of the JD dimensions. For city–level balances, the jurisdiction level is 11, and so the entire geocode must match in order to be included in the balance sum. This approach has the side effect of only including the part of the city in the county that appears in the JD

context. For example, CITY_WITHHELD_ASG_JD_RUN with the JD context set to 44–157–1440 only returns the portion of the Houston city tax based on earnings in Fort Bend County, not in Harris or Montgomery County. To get the full "Houston city tax withheld" value, the balance must be evaluated for all three counties.

A somewhat separate set of codes are the school district codes, used to identify which school district is owed tax. The school district code is a five digit number, with one set of codes for each state, chosen by the state.

An employee registers for school district tax in his resident city. Earnings are never tagged against school districts, so the code only appears on jurisdiction _result_ values for school district tax, the school district SUBJECT results, and nowhere else. The format used is *xx–xxxxx*, where the first two digits are the state code, and the next five digits are the school district code. This approach avoids the potential problems of school districts crossing city borders. Thus, when accessing school district level balances, this format must be used for the JD context.

Table Structure

The geocode data is held in the following tables:

- PAY_US_STATES contains all the US states and territories.
- PAY_US_COUNTIES contains all the counties for each state.
- PAY_US_CITY_GEOCODES contains all valid geocodes for cities.
- PAY_US_CITY_NAMES contains the names for each city geocode.

  Each geocode may have multiple names (aliases), but the official one is flagged such that PRIMARY_FLAG = Y.
- PAY_US_ZIP_CODES contains valid zip code ranges for each city geocode.
- PAY_US_CITY_SCHOOL_DSTS contains school district names / codes for each city (only those districts that have a tax).
- PAY_US_COUNTY_SCHOOL_DSTS contains school district names / codes for those not in cities.

The data in these tables is sourced from Vertex, and does not contain cities with less than 250 residents.

APIs

The tables that contain addresses (PER_ADDRESSES and HR_LOCATIONS) are referenced against this data to ensure that proper geocode translations exist. These tables themselves only hold the state, county, city names and the zip code. A function Addr_Val translates the information on the address to the internally used geocode representation. In addition to the validation built into the forms, a SQL script is available to check all addresses at once.

Address Validation

A SQL script (pyvaladr.sql) has been provided to validate US addresses. Customers who have been using Oracle Human Resources and are now implementing Oracle Payroll can use this script to ensure that the addresses have already been entered are valid. In this case, valid means that the state, county, city, and zip code are all consistent with the data in the geocode tables. This script lists all invalid addresses in an output file (pyvaladr.lst).

For PER_ADDRESSES, it shows the full name, employee number, social security number and address of those people whose address is invalid. For HR_LOCATIONS, it shows the location code and address of the location whose address is not valid.

Addr_Val

This function takes the state abbreviation, county name, city name, and zip code from an address and returns the corresponding geocode. Addr_Val is in the HR_US_FF_UDFS package. This function is available for use in formulas as a User–Defined Function (UDF) called get_geocode. It appears as follows:

```
FUNCTION addr_val (     p_state_abbrev  IN VARCHAR2 DEFAULT NULL,
                p_county_name   IN VARCHAR2 DEFAULT NULL,
                p_city_name     IN VARCHAR2 DEFAULT NULL,
                p_zip_code      IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

If no valid geocode is found, then 00–000–0000 is returned.

Expansion

In the near future, Oracle Payroll will allow employees to have work and resident locations for cities that Vertex does not support. In order to do so, a special non–Vertex geocode *xx–xxx*–UNKN will be used. No city

with such a geocode will have the PRIMARY_FLAG set to Y, since they will not be aliases to a single true city with that geocode.

Users will be able to add new cities to the geocode tables, which will be marked with the special geocode. Addresses can then include the new cities. The tax subsystem, of course, will not support these new cities; tax rules will not be permitted to be entered against the new cities. Instead the user will need to enter any relevant percentages and filing status against the containing county. Addr_Val (from above) assists by converting the UNKN city code to 0000, resulting in a county level geocode. This measure prevents any problems stemming from the fact that different ”new cities” might have the same geocode. If multiple Vertex entries with the same geocode were created, the same tax could potentially be calculated more than once.

PayMIX

This essay covers the following topics:

- Overview
- PayMIX architecture
- PayMIX processes
- User customization
- Link and performance considerations
- Populating from an external source

Overview

PayMIX is an Oracle Payroll feature for US users. It allows you to enter large volumes of earnings–related data, such as timecard data, in each payroll period. You can enter this information manually using the online windows or you can load batches from an external source directly into the PayMIX interface tables.

The PayMIX batch loading process loads the data from these interface tables and ensures the data entered is accurate and maintains the integrity of the Payroll system.

Typical types of data you might use PayMIX to load are:

- Hours worked with cost data
- Overrides or adjustments to earnings or deductions
- One time earnings or deductions

Using PayMIX, you set up your input data in batches. You can define your own control totals for each batch and use these to reduce the possibility of input error. Once these batches have been balanced, they should be kept in balance until they are released to the payroll calculation process.

PayMIX Architecture

This section summarizes the individual components of PayMIX.

PAY_PDT_BATCH_HEADERS

This table stores the header record for a batch. A stored procedure is available to insert and update records in this table. It has an associated view PAY_PDT_BATCH_HEADERS_V, for forms, view resolves all foreign keys and the lookup codes.

Sequence: PAY_PDT_BATCH_HEADERS_S

PAY_PDT_BATCH_LINES

This table does not have any form view associated with it. This has minimum validation for data entry in order to allow ”heads down” data entry. Data is validated by the batch validation process.

Sequence: PAY_PDT_BATCH_LINES_S

PAY_PDT_BATCH_CHECKS

This table stores running totals for each batch. Each batch type has different types of running totals

Lookup Type	Code	Meaning
PAY_PDT_ACTION	DEC	Decrease
	INC	Increase
	REP	Replace
PAY_PDT_BATCH_STATES	E	Error
	H	Hold
	R	Ready
	T	Transferred
	TW	Transferred with warnings
	V	Validated
	W	Warning
PAY_PDT_BATCH_TYPES	D	Deductions
	E	Earnings
	T	Time
PAY_PDT_DED_CKSUM	LINES	Line count

Lookup Type	Code	Meaning	
		TOTAL_DOLLARS	Total amount
	PAY_PDT_EARN_CKSUM	LINES	Line count
		TOTAL_DOLLARS	Total amount
	PAY_PDT_TIME_CKSUM	H	Hours
		LINES	Line count
		TOTAL_DOLLARS	Total amount
	PAY_PDT_VDPO	TO	Tax only
		TP	Tax and pretax
	PAY_PDT_PROCESS_MODE	T	Transfer
		V	Validate

Forms

PayMIX has two forms:

- PAYBTENT is the PayMIX entry form.

  Use this form to setup and maintain batches and batch lines.

- PAYBTSUM is the batch summary form.

  Use this form to review all batches and their current status. Batches with a status of Error are displayed first.. You can call PAYBTENT from this form whenever you want to view the details for a specific batch.

Packages

There are four Pl/sql packages associated with PayMIX, as follows:

- pycskfli allows users to enter costing validation, as PayMIX itself does not perform such validation.
- pypdtchk contains server side validations for Batch Header Block in PAYBTENT.

- pypdtent contains Insert / Update / Delete procedures for PAY_PDT_BATCH_HEADERS.
- pypdttrx contains all the transfer and validation code.
- pypdtval contains validation used by the PayMIX entry form.
- pypdtuvd contains a procedure that users can use to create customized validation.

Predefined Rules and Edits

PayMIX is delivered with following predefined edits at batch level:

- Line Count

  Users can enter the number of lines they are planning to enter for a given batch. System calculates actual lines entered and matches the total against expected number of lines.

- Amount

  Total of the amount column in a batch. This edit is available for all batch types.

- Hours

  Total of the Hours column in a batch. This edit is available only for a batch type of ”TIME.”

PayMIX is delivered with following predefined edits for each line item:

- Ensure that an employee exists and is valid for the transaction being processed
- Ensure that a terminated employee is not paid for a period after his / her termination date
- Ensure that an employee is eligible for the element
- Ensure that if an override is entered, then the element entry to be overridden exists
- Ensure that transactions do not incur excessive pay
- Ensure that total regular hours reported as worked do not exceed an employees regular weekly scheduled hours
- Check for a change in base rate for hourly time card workers to ensure that hours are paid at the correct rate
- If the hours reported are for paid time off the accrued hours for the appropriate PTO category are checked to ensure that there are sufficient hours accrued to cover the hours reported

You can customize edits and predefined rules. You can extend the lookup provided for batch totals edits.

Mapping to Predefined Rules for Earnings and Deductions

Earnings and Deductions Input Value Mapping

PayMIX maps to earnings and deductions using the following table. PayMIX entry is allowed for the earnings and deductions where the MIX Entry Allowed flag is set to Y.

PayMIX Column Name	Input Value Name	Time	Earnings	Deductions
Earning Name	N / A (Element Name)			
Amount	Amount *or* Pay Value if Amount Input value does not exist.	x	x	x
Cost Center	N / A (KF Segment)			
Action	INC – Additional Amount (+)		x	x
	DEC – Additional Amount (–)			
	REP – Replacement Amount			
Location	Jurisdiction Code	x	x	
Date Earned	Date Earned		x	
Separate Check	Separate Check	x	x	
Tax Separately	Tax Separately	x	x	
Hour Type	N / A (Element Name)			
Hours	Hours	x		
Labor Distribution Code	N / A (KF Segment)			
Rate Code	Rate Code	x		
Hourly Rate	Rate	x		
Shift	Shift	x		
Multiple	Multiple	x		
From Date	Effective Start Date	x		
To Date	Date Earned	x		

PayMIX Column Name	Input Value Name	Time	Earn-ings	Deductions
Deduction Name	N/A (Element Name)			
Asc Balance Over-ride	Towards Owed			x
Tax Rule	Deduction Processing	x	x	

Relationship to Special Inputs Element

Whenever an earnings and deduction is created using the template screens, two extra elements are created for entering PayMIX entries. PayMIX maps to the Special Inputs element, which is a nonrecurring element created with a recurring element in order to allow entry through PayMIX. The following table explains the mapping rules.

Circumstance	Increase/Decrease	Replace
Recurring Earn/ Dedn Entry Exists	Create Special Inputs – Addl Amt Entry	Create Special Inputs – Replace Amt Entry
Recurring Earn/ Dedn Entry does not Exist	Reject – no recurring entry to adjust	Reject, no recurring entry to replace
Nonrecurring Earn	Rejject – no recurring entry to adjust	Create nonrecurring element entry

PayMIX Processes

PayMIX has one SRS program to validate and transfer batches. The name of this PL/SQL stored procedure is PAY_US_PDT_PROCESS.RUN_BATCH. You can specify multiple batches as an input to this procedure; the commit unit is one batch. When a single line item is in error, the status of the batch is set to Error. The same stored procedure is used to validate and transfer. A batch is validated every time before a transfer irrespective of its current status.

Effect of Various PayMIX Inputs on the Payroll Process

The following PayMix inputs affect the payroll run process. If the Date Earned falls in previous payroll periods, then an entry is created in that payroll period. However, you cannot see this entry from the Element Entry window unless the effective date is set to fall within that period.

- Date Earned

- Separate Check
- Tax Separately

Example

The following entry will be processed by a payroll run process which processes all prior period unprocessed element entries.

- Current pay period is JUL–15–96 to JUL–31–96
- A PayMIX entry is made for an Earning with Date Earned set to 01–JUN–95
- An element entry will be created for the 01–JUN–96 to 15–JUN–96 pay period

User Customization

Batch Level Edits

Batch level edits are user extensible. Users can define new batch edit types using the Quickcodes window. Whenever a new edit type is defined, you need to change the stored procedure to perform appropriate processing.

Steps to define a new batch edit follow:

1. Add a new Quickcode value for the appropriate batch type edit.

   PAY_PDT_DED_CKSUM for a Deductions entry

   PAY_PDT_EARN_CKSUM for an earnings entry

   PAY_PDT_TIME_CKSUM for a Time entry

2. Create a new procedure for a new batch edit.
3. Modify the PAY_PDT_BATCH_VAL_PKG.DO_EDITS procedure to call the procedure created in last step.

Steps to define a new line edit follow:

1. Create a procedure for a new line edit.

   This procedure could be for a new validation you want to perform on each PayMIX line or a new business rule you need to apply before transferring data into Oracle Payroll tables.

2. Modify PAY_PDT_BATCH_VAL_PKG(VALIDATE_LINE for a new line edit.

3. Modify PAY_PDT_BATCH_VAL_PKG(VALIDATE_BUSINESS_RULE for a new business rule.

The suggested approach is to keep all user customization in a separate stored procedure. Any upgrade or enhancement to a base PayMIX function will change the base package to change, you must apply such changes after each upgrade to PayMIX.

Costing Keyflex Validation

In order to allow PayMix to process more costing information, the PayMix window now pops up from the cost allocation key flexfield instead of the cost center code and labor distribution code. The cost allocation key flexfield is user–defined, and allows up to 30 segments. PayMix allows the user to default any or all of the defined segments, and does not enforce the entry of ”required” segments. All costing information is optional; it is not required and is allowed to be null.

PayMix does not validate costing information, but it does provide for user–defined validation of the cost allocation key flexfield.

Skeleton Flexfield Validation Packages

To maintain the costing key flexfield through PayMix, use the skeleton flexfield validation package. Your implementation team can modify this skeleton code to carry out your flexfield validation using PL/SQL. The skeleton flexfield validation package for PayMix is in file pycskfli.pkb in $PAY_TOP/admin/sql.

The supplied validation will raise a PL/SQL exception if any of the flexfield segments are set. Therefore, if you wish to transfer costing information using PayMix, you must implement your own validation in place of the supplied validation.

Only the PayMix transfer batch process calls this package to validate costing flexfield data. Forms will still use the definitions in the Oracle Application Object Library Forms. Hence, it is important for you to ensure that any changes to the definitions and flexfield validation packages are synchronized.

Modifying the Flexfield Validation Package

Make a copy of the skeleton file. A procedure called kf should be edited.

The PayMix transfer batch process calls the validation package just before it transfers the batch. Every segment for the costing key flexfield is passed to the kf procedure in a single PL/SQL record structure. The components of the record structure cannot be modified; the values can only be read.

It is best to implement validation for each flexfield segment in a separate procedure. Then modify the kf procedure to call the relevant validation procedure for each segment. For example, suppose segment1 of the costing key flexfield contains the cost center. If the valid cost centers are 101, 102, 105, and 107, the validation for this particular segment would look like this:

```
Write a validation procedure to check that the segment value is
valid. Raise a PL/SQL exception if it is not.
 procedure val_seqment1
   (p_rec in pay_csk_shd.g_rec_type
   ) is
   begin
      if p_rec.segment1 NOT IN ('101', '102', '105', '107') then
   - -
   - - The cost center is invalid.
   - -
   dbms_standard.raise_application_error
      (num => -20999
      ,msg => 'Cost Center is invalid'
      );
   end if;
   - -
end val_segment1;
```

Then modify the kf procedure to call the validation procedure which corresponds to segment 1:

```
procedure kf
   (p_rec in pay_csk_shd.g_rec_type
    ) is
- -
begin
  - -
  - - Check if segment1 has a value, then call the relevant
  - - validation procedure.
  - -
  if p_rec.segment1 is not null then
    - -
    val_segment1(p_rec => p_seq);
    - -
  end if;

  -- ...repeat for each defined flexfield segment.
```

```
  if p_rec.segment2 is not null then
    - -
    - - Call validation procedure for segment2
    - -
    val_segment2(p_rec => p_rec);
    - -
    - - and so on
  end if;
end kf;
```

It is your responsibility to ensure that:

- The supplied flexfield validation package body creation scripts are not edited.

  If you want to customize the supplied flexfield validation packages, copy the scripts and modify the copies. After an upgrade, you should check that your customizations are consistent with the new packages supplied with the upgrade. If so, you can rerun your customized flexfield package body creation scripts.

- No other packages, including the supplied flexfield validation package headers, are modified.
- Any flexfield attributes or segments which have no corresponding definition are checked to ensure that they contain a null value.
- Any code changes made to these files compile without errors and have been fully tested.
- If the Forms flexfield definitions are altered, the corresponding PL/SQL code change is made to your copies of the Flexfield validation package creation scripts.

User–defined Validation in PayMix

You can supplement PayMix–supplied validation with user–defined validation by adding PL/SQL code to the procedure validate_business_rules in the file pypdtuvd.pkb in $PAY_TOP/admin/sql. The procedure includes several useful arguments and contains sample code to raise an error for duplicate timecard entries. (This code is commented out.)

It is your responsibility to ensure that:

- The supplied flexfield validation package body creation scripts are not edited.

If you want to customize the supplied flexfield validation packages, copy the scripts and modify the copies. After an upgrade, you should check that your customizations are consistent with the new packages supplied with the upgrade. If so, you can rerun your customized flexfield package body creation scripts.

- No other packages, including the supplied flexfield validation package headers, are modified.
- Any code changes made to these files compile without errors and have been fully tested.

Link & Performance Considerations

Every earning and deduction defined using a window has two associated elements:

- Special Inputs (nonrecurring)
- Special Features (nonrecurring)

Only Mix uses the Special Inputs element.

See: Template Element Technical Reference Manual Essay

When you define an earning or deduction with the MIX Entry Allowed flag set to Yes, you must link the Special Inputs element just as the parent element is linked. Do not link the Special Features element; this element is used for indirect results from the formula to adjust the balance.

Each of the data entry blocks is designed to have minimum validation for data entry, in order to allow ”heads down” data entry. This allows a user to enter a whole batch of data without continually stopping to correct individual field values. LOVs are assigned to fields wherever appropriate, but designed not to force users to select an existing value. This design approach provides the following advantages to PayMIX:

- Low Network traffic
- Efficient remote data entry
- Modular user customization

Populating from an External Source

Data can be inserted into PayMIX tables directly. A batch is the commit unit for the PayMIX validation and transfer process. A large batch may cause the validation / transfer process to fail because not enough rollback segments exist. Following is the step–by–step process to transfer data from an external source into a PayMIX table:

1. Create an ASCII format data file from the external data source.
2. Use SQL*Loader to transfer data from the ASCII file to a temporary Oracle table. The temporary table should look similar to PAY_PDT_BATCH_LINES.
3. Create a batch header in PAY_PDT_BATCH_HEADERS for each 100 to 1000 lines from the temporary table.

   You must break your transaction into small batches, but you can use the same reference number for all the related batches, in order to process together all the related batches from one transaction during the transfer / Validate process. You can use following sample procedure call to create batch header.

```
Declare
     P_batch_id   Number         := NULL;
     P_rowid      Varchar2(30)   := NULL;
     P_payroll_id Number         := NULL;
---
     Cursor c1 is
     Select payroll_id
     from pay_payrolls_f
     where payroll_name = '** PAYROLL NAME **'
  and sysdate between effective_start_date
                  and effective_end_date;
 --
 begin
     OPEN   c1;
     FETCH  c1 INTO P_payroll_id;
     CLOSE  c1;
 --
 PAY_PDT_BATCH_HEADER_PKG Insert_Row(
   X_Rowid      => P_rowid,
   X_Batch_Id   => P_batch_id,
   X_Batch_Type => ['D'for Deduction] ['E'for Earning]
                   ['T' for Time] ,
   X_Batch_Status => 'R', -- Ready to process
   X_Payroll_Id                    => P_payroll_id,
         X_Business_Group_Id => [BG ID for Business],
         X_Reference_Num => [Ref No. Can be same for
                             a group of transactions],
```

```
                    X_Last_Update_Date => SYSDATE,
                    X_Last_Updated_By  => [A valid Apps User ID],
                    X_Last_Update_Login => [A valid Apps UserID],
                    X_Created_By       => [A valid Apps User ID],
                    X_Creation_Date    => SYSDATE);
          end;
```

4. Loop through temporary file and use the BATCH_ID from the previous step to create the desired number of batch lines in PAY_PDT_BATCH_LINES.

   No stored procedure is available to maintain this table. You need to insert a record directly into the table.

5. Insert a record for each batch edit type in PAY_PDT_BATCH_CHECKS.

   You should maintain a running total for each batch check type while inserting records in the PAY_PDT_BATCH_LINES table.

6. Follow the last three steps until all the lines from the temporary file are transferred to PayMIX tables.

7. Run the PayMIX Validation process for all the batches. If any line is in error, correct it using the PayMIX window and revalidate the Batch.

8. Run the PayMIX Transfer process to transfer all the validated batches to payroll tables.

9. Truncate the Temporary table.

The current release of PayMIX does not include a purge utility to delete records from PayMIX tables. You can design a purge utility to fit your own requirements. Following is an example of a SQL script to delete all the Header records and lines for all the transferred batches:

```
delete from pay_pdt_batch_lines
where batch_id in ( select batch_id from pay_pdt_batch_headers
                    where batch_status = 'T');
--
delete from pay_pdt_batch_headers
where batch_status = 'T';
--
Commit;
--
```

You can perform some edit functions to the data in your temporary table (total amount or total hours in a given transaction batch, for example). Also, you can check for business rules before transferring data into PayMIX tables (total number of hours for a person in a pay period

should not be greater than 100, or any earnings amount should not be greater then $5000.00, for example). You can issue warning messages and delete or modify the line item, if required.

APPENDIX

B

Technical Essay on Payroll Processes

This technical essay describes the payroll processes run as part of the regular pay cycle. It also identifies which payroll action parameters control the Oracle Payroll batch processes, and explains how assignment level interlock rules prevent payment results from being changed. Specifically, this essay discusses the following topics:

- Overview
- Payroll Run Process
- Pre–Payments Process
- Payment Processes
- Magnetic Tape Process
- Cheque Writer / Check Writer Process
- Cash Process
- Costing Process
- Transfer to the General Ledger Process
- Assignment Level Interlocks
- Payroll Action Parameters

Overview

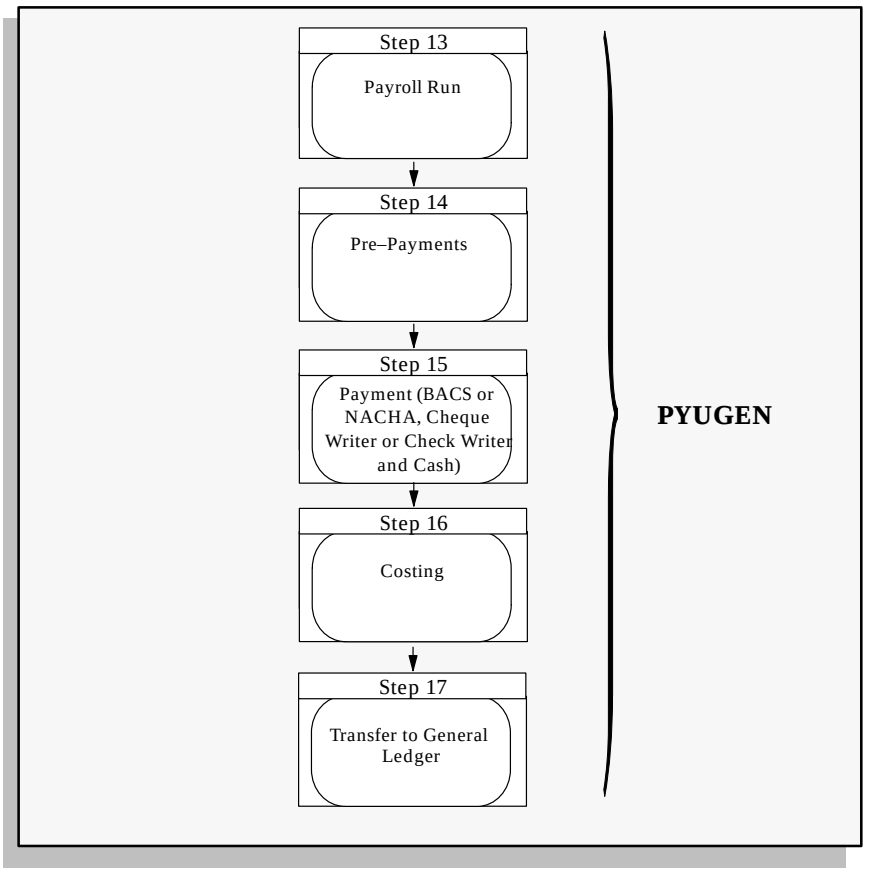
Oracle Payroll provides you with the flexibility you require to run your regular pay cycle in the best way to meet your business needs. To do this, we provide you with a modular batch process called PYUGEN.

PYUGEN

PYUGEN is a generic process that performs many actions, depending upon which parameter sets and defaults the Oracle Payroll system administrator registers it with.

The following figure illustrates the payroll processes executed by PYUGEN, and the typical sequence in which they are performed. Each individual process performs the different actions required to calculate and generate your employees pay.

Figure B – 1
Pay Cycle Sequence



The parameter identifies the specific payroll process to execute. These are predefined in Oracle Payroll; the value is not visible to the user.

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 may 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 B – 69

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 employee's 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 (BACS or NACHA)
- 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 B – 6

Pre–Payments Process: page B – 20

Payment Processes: page B – 26

- The Magnetic Tape Process: page B – 27
- The Cheque Writer / Check Writer Process: page B – 49
- The Cash Process: page B – 58

Costing Process: page B – 59

Transfer to General Ledger Process: page B – 63

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

See the Oracle Payroll User's Guide for more information about these supporting processes.

Assignment Level Interlocks

This sequence in which the PYUGEN calculates payment is critical to the success of processing. This is because 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 (that prepares the payments according to the payment methods) uses the results of the Payroll Run process (that 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.

See: Assignment Level Interlocks: page B – 64

Payroll Run Process

The Payroll Run process calculates the gross to net payment for your employees.

Payroll Run 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 identified as the *run result values*. These individual results are accumulated into balances that summarise gross to net, and in particular the payment balances. Payment balances are taken forward by the next process in the regular pay cycle, Pre–Payments.

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. This is set up according to the assignment set and element set specified in the batch submission. The default is All.

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 date–effective, the Payroll Run process also identifies changed DateTracked information and actions it accordingly. For example, an update of a date effective entity may require an actual insert into the table.

Main Entities

The following list indicates the main entities for processing:

Key: S = Select, U = Update, D = Delete, I = Insert.

Entity Name	Date Effective?	Processing
Payroll Action	No	S
Assignment Action	No	S, U
Element Entry	Yes	S, U
Element Entry Value	Yes	S, U
Person Latest Balance	No	S, U, D
Assignment Latest Balance	No	S, U, D

Entity Name	Date Effective?	Processing
Balance Context	No	S, U, D
Run Result	No	S, U, D, I
Run Result Value	No	S, U, D, I

Process Each Assignment

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. By default, recurring entries are always fetched and processed when you submit a payroll run. You must use Element Skip rules, or Element Sets to limit the inclusion of recurring entries.

If Oracle Payroll is installed outside the United States and uses 'additional' entries of a recurring element in the pay period, Payroll Run treats them as nonrecurring entries for processing purposes.

Processing Priority

The sequence of processing entries for each assignment is determined by the processing priority of the element type, 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 US legislations, 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.

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, for example, US legislations. The value 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

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.

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). Additionally, 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 or element types. 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

In processing entries, Payroll Run has to cope with many different conditions. For instance, a Substitute Override prevents the processing of any other entries of the same element type (it literally "overrides" them). Another example is that if a skip rule shows that an element

entry must be skipped, the target of the skip must not be processed, and so on.

Internally, Payroll Run uses the concept of a ”State Machine” to control this. At first, the ’state’ is set to indicate that it should process. Then, depending on the entry type and whether a skip rule has fired, a different mode is set. This then affects the processing of this entry and (possibly) subsequent entries. Taking an example from above, the Substitute Override sets a mode to say that it is processing an Override. When subsequent entries are encountered, they have a mode of Overriding set and are not processed. This remains in force until a change of element type.

Balances and Latest Balances

Payroll Run needs to be able to access and maintain balances and latest balances. For more information about latest balances, see the introduction to balances in the TRM essay, *Loading Latest Balances*.

Any existing assignment– 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 itself is a union, in this case because the two types of balances are held in separate tables.

Expiry Checking of Latest Balances

All loaded balances must be expiry checked. If they have expired, they are set to zero. Information for this comes from the dimension associated with a particular balances. The expiry step is entirely separate from the loading step, due to the need to deal with balance context values.

Expiry Checking and Performance

To process expiry checking, Expiry Checking code is called 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 approach used is:

- 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.

Creation and Maintenance 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.

The newly created or updated balances must be written to the tables.

Creation of New In Memory Balances

There are three places in the code where *in memory balances* are created, depending on what dimension type the balance has; one place is for types A, P and F, and two places are for type R.

> **Note:** In memory, balances are only created for the A, P, R and F types.

In memory balances are created for:

- A, P and F.

  This follows the execution of a Fast Formula. If it has just accessed a defined balance with one of these dimension types, that is not already held as an in memory balance, an in memory balance with the value as accessed by the formula is created.
- R:
  - Before the execution of a Fast Formula.

If it accesses a defined balance with the 'Run level' balance dimension type, it creates an in memory balance with a value of zero (which is what a run level balance must be, by definition).

– Before balance feeding time.

If the balance type it is attempting to feed has defined balances with run level dimension types, it creates in memory balances as appropriate, with zero value. Also see the section on maintenance of balances.

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:

*For each processed run result, look at the balance feeds (which identify the balance types that are potentially fed by each run result value). Then scan the in memory balances to see if there are any potential feeds. If so, perform 'feed checking'. Feed checking (as expressed by the feed checking type on the appropriate balance dimension) is performed. If this shows that it should feed, then: balance value = balance value + (result value * scale).*

There are a number of possible feed checking types:

- NULL

  The result always feeds 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

  Feeding occurs if there is a match between the in memory balance context values and the contexts held in the UDCA.

- J

  This is a special case for United States localisation.

- S

  This is a special case for United States localisation.

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 localisation. 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.

Formula Processing

Payroll Run calls Fast Formula to enable it to perform its complex calculations.

> **Note:** Even if a Formula has been defined against an element type via a Status Processing Rule, it does not fire if the Pay Value is not null.

Introduction to the Fast Formula Interface

The interface used by Payroll Run to access Fast Formula is made up of two sections, which are:

- the common part of the interface (such as those meant to be 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, that allows access to Formula's database item cache. Execution of Fast Formula 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 may access any of the defined balances held as latest balances, it writes the current value of the balance to the Fast Formula database item cache. It is this mechanism that allows us to benefit from the existence of in memory balances.
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 Fast Formula has told us that it returns are directed to a buffer area.

Execute the formula

The third party post formula hook is called. This allows special legislative dependent functions to manipulate the formula results before they are processed by Payroll Run. For instance, it allows 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 summarised 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. Following this, 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 type, 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 to 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 (the latter 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 way:

- 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. In addition, the special defaulting for ”Benefit” elements is processed as required.

At the same time, 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 reasons that may trigger it. If it is about to execute a Fast Formula that accesses a database item not held in memory. This is required because 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 type 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.

As it happens, the cache also introduces further advantages. These are:

- the SQL statements that access the data for the entity are isolated in one place which is good practice.
- the actual coding itself is easier to understand and maintain.

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 proportion 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 (U.K. 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 group has a default payment method, and this is used by Pre–Payments if 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. See Prenotification (Prenoting): page B – 22

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, providing 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 (U.K. 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 setup 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 QuickCode Values widow and query the QuickCode 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 organisation payment method.

Prenotification (Prenoting)

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. When an employee has changed bank or bank details, a payment value of zero is made to the employees 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. It must complete for an assignment action before that assignment can be paid. This is different in R10 in that it does not necessarily exist only for this time period. A consolidation set is now just a tag which ties groups of actions together. For example, in R9 a consolidation set is typically used to enable prepayments to run on a group of payroll runs with one easy action. In fact it is more flexible because it can be used to prepay all assignment actions in the set which have not yet been prepaid, whether or not they are in this period.

A further benefit of this change to the nature of a consolidation set is that in routines such as BACS which may consist of data from many periods, it can be used to tie together payments from different periods and even different payrolls. For example, a consolidation set may be used to force BACS 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 organisations or individuals. For example, court orders 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 allows the same third party element type to have two entries that are paid to different parties for an assignment. An example, of this would be an employee paying a third party element of 'Child Support' to two different people.

Third party payments can only be made by magnetic tape or cheque, 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, and as a result, a payment method must always be specified. 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. The only exceptions are the third party payments, these are paid as per the method specified for them.

The overriding payment method can be either:

- Cash
- Cheque

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 runs assignment actions.

See: Assignment Level Interlocks: page B – 64

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 THREAD 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. It is the PL / SQL procedures that create the entries in the Pre–Payment table.

The threads process the assignment actions, by retrieving the third party details and paying the value in the appropriate manner as defined by the personal payment methods. Following this, the value for

the remuneration balance for this assignment is retrieved using the PL/SQL balance functions. This is then broken down into payments 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 then marks the payroll action status as Complete.

A value can be set to error the payroll action after a set number of consecutive assignment action errors, the action parameter MAX_ERRORS_ALLOWED sets this value. If MAX_ERRORS_ALLOWED is not found then the chunk size is used as default.

All the error messages are written to the PAY_MESSAGE_LINES table with a more detailed explanation in the log file. This method of handling errors enables Pre–Payments to continue processing if minor errors are encountered. For example, if Pre–Payments has thousands of assignments to process and a few are paid by cash but the currency details have not been loaded, the process creates an error for the assignments with cash payments (process unable to perform the cash breakdown). Most assignment actions complete, only the assignments with errors have to be rerun.

Payment Processes

Now that you have run the Pre–Payments process to prepare the results for payment (according to the payment methods), you are ready to produce payment for your employees.

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

- The Magnetic Tape process – MAGTAPE (BACS or NACHA)

  See: Magnetic Tape Process: page B – 27

- The Cheque process – CHEQUE

  See: Cheque Writer / Check Writer Process: page B – 49

- The Cash Payments process – CASH

  See: Cash Process: page B – 58

  Note: This is a U.K.–only process.

Recording Manual Payment

You can also record any manual payment you make to a specific employee. This has the effect of marking the prepayment as paid. The Payment process take the unpaid prepayment values allocated to each payment type and produce the required payment file.

> **Note:** Payments can be made externally using the External Payments form. If you make a payment external to the system, the Magnetic Tape process does not pay these.

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.

The Process

There are two types of magnetic tape file:

- Payments
- End of year tax reporting

The two types of magnetic file, payments and reporting, are created differently. The order of the entries in the magnetic file is critical. Because of this, the Magnetic Tape process cannot run with multiple threads (unlike the PrePayments or Cheque Writer processes that can).

The actual format of these tapes is legislation specific.

In release 8 and 9, localisation teams had to write a process for each area for each legislation. Now, with release 10, localisation teams are provided with a generic 'tape writer' process that enables them to specify exact tape formats required by their legislation.

The tape process is a simple 'C' harness which calls ORACLE7 stored procedures and "FASTFORMULA" formulas to produce the required tape file. The routine is generic, you can use it for any task which requires magnetic tape reporting. The actual structure and content of the tape is defined entirely by the stored procedure and a series of formulas.

Examples

Some examples that use the routine are:

- BACS
- NACHA
- W2
- P35 submissions (and equivalent in other countries)

The routine can open several files for output, one of which is the magnetic file requested. The file name is based on the concurrent request ID when the code is run for magnetic tape payments or is

passed as a start parameter for magnetic reports. Several other files can be produced by this process, these files could be used to audit the assignments that are being processed. The file name is in a similar format as the magnetic tape file.

Magnetic Tape Structure

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:

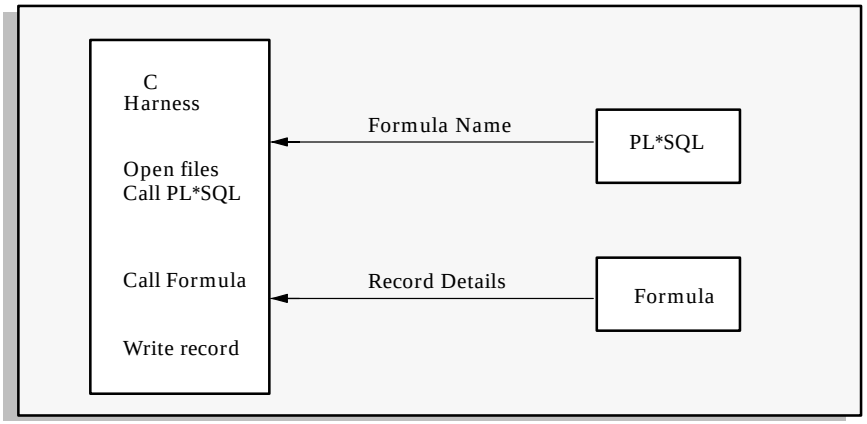
Record	Fields
Tape Header	Batch Id, Company Name, Batch Record Length, and so on
Employee	Employee Id, Salary, Age, Job, and so on
Tape Footer	No. of Records Processed, Salary Total, and so on

Magnetic Tape uses this strict layout. A piece of driving PL/SQL code returns the name of a formula, and the formula writes one type of record; that is the Tape Header to tape. This method of getting the formula and record name, then writing the record to tape is repeated until all the records are processed. A 'C' code harness performs the file handling (opening, closing and writing to files), and enables the PL/SQL and the formulas to interface.

C Harness and PL/SQL

The following figure illustrates the Magnetic Tape process:

- C harness for file handling and interfacing
- PL/SQL for sequencing records and formulas (to define the contents of the record)



Database Items

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 the variables used in the Magnetic Tape process. These parameters can be:

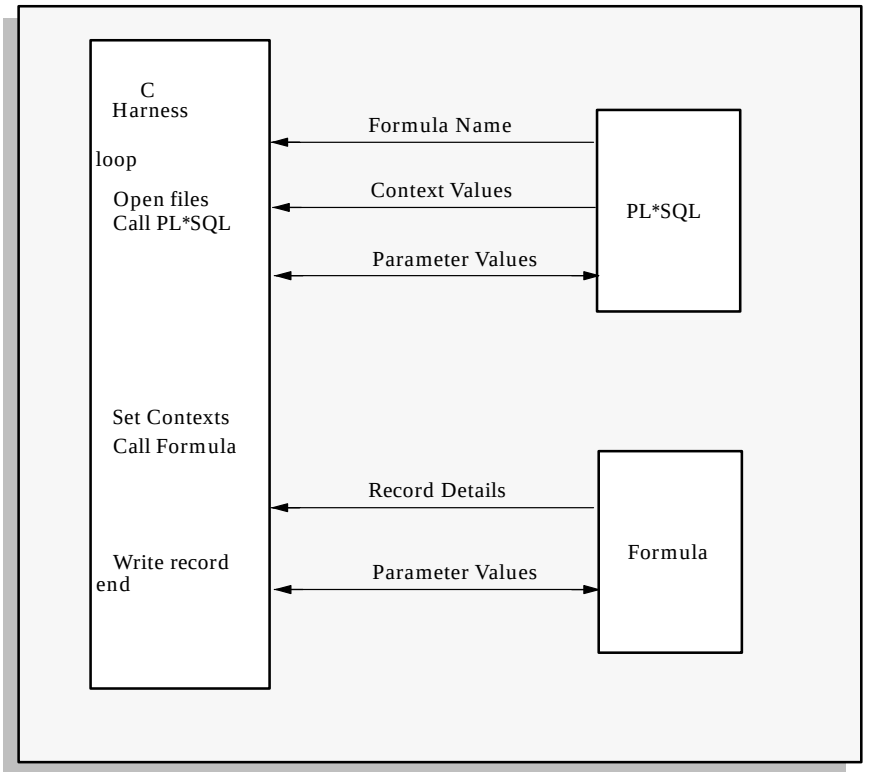
- Passed into the C process from the command line
- Created by the driving PL/SQL procedure
- Created by the formula

Only the driving PL/SQL procedure and the formula can update the values.

These parameters are used to store the variable data to be transferred between the formula and the PL/SQL. The running totals are passed to the formula in this way.

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



So, 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.

Magnetic Tape Formats

There are two things required to create the magnetic tape format:

- Formulas
- Driving PL/SQL procedure

Formulas

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

The prefix of the name of the return value determines which operation is performed on the return value. See the *Formula Interface* document for technical details.

The Driving PL/SQL Procedure

The driving procedure determines the sequence in which the records are processed. This driving procedure can be written from scratch by opening cursors processing a particular formula for each fetch of the cursor, or the generic PL/SQL can be used. See the *PL/SQL Interface* document for technical details.

Magnetic Tape Payments

The payroll assignment action creation code is the entry point to the Magnetic Tape Payments process. Employee magnetic tape payments such as BACS and NACHA are recorded on the core HR system 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 total NI bill to the DSS or 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**

A magnetic tape payroll action is initiated to pay any unpaid prepayments of the specified payment type. The consolidation set restricts so that it only pays those unpaid pre–payments which are for a prepayment action of the same consolidation set.

payment_type_id **Mandatory**

This parameter defines the driving PL/SQL procedure.

effective_date **Optional**

This parameter identifies the effective date for processing.

payroll_id **Optional**

This parameter processes assignments which are on the specified payroll on the effective date

start_date **Optional**

This parameter specifies how far back, date effectively 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**

This parameter creates assignment actions interlocking to unpaid prepayments for that payment.

legislative **Optional**

These parameters are free–format, available to all payroll actions. Localisation groups can 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 PL/SQL driving procedure used is that 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 created and not further restrict the assignments processed. If the PL/SQL was to further restrict which assignments were processed, then there may be magnetic tape assignment actions which couldn't ever be 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 prepayment 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.

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 a directory, such that:

```
if $APPLCSF is defined

 write to $APPLCSF/$APPLOUT

else

 write to $PAY_TOP/$APPLOUT

end if
```

The audit files are created in the same way, except that the file extension is different. The audit file extension is .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.

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 be 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 Formula Interface: page B – 41

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.

The magnetic tape filename is generated if it is not supplied to the process. The file 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, we define the generic tape writer process once, as an executable. We can then define any number of concurrent programs which 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 setup 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.

The PL/SQL Driving Procedure

The PL/SQL driving procedure determines the format of the magnetic tape file. The interface between the 'C' process and the stored procedure makes extensive use of PL/SQL tables. Pl./SQL tables are single column tables which 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 PL/SQL can be of two forms: it can be specifically for one type of magnetic tape file, or it can be used to drive off the magnetic tape batch 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:

1. data that is used in subsequent cursors
2. data that is to be used as context value data
3. 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 subsequent fast formula. The business_group_id column is being retrieved so that it can be used 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

Access to Data

Some cursors require access to data previously selected, this can be achieved in two ways:

1. If the column needed was selected as a context or an individual column (like business group in the previous example) then a function is provided to return 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)
   ```

2. Another way of using a value in subsequent cursors is to 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')
   ```

Different 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 it would be this value that would be used for the formula, it is at a lower level in the retrieval table than the previous DATE_EFFECTIVE, until the cursor is close. In which case 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 fast formula can access these values by selecting that parameter on the input line and if the formula returns a value for that parameter it overwrites the entry in the table. If the FastFormula returns a parameter that does not exist, the parameter is entered in the table.

Driving Structure

The driving structure for the package procedure is held in two database tables:

- PAY_MAGNETIC_BLOCKS
- PAY_MAGNETIC_RECORDS

SQL> desc PAY_MAGNETIC_BLOCKS

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)

SQL> desc PAY_MAGNETIC_RECORDS

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

In the following examples the tables have simplified:

Cursor/Block Table

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

Formula/Record Table

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

The block table has columns block_id (system generated), cursor name and blockname. However, no_of_select_values is the number of columns that the select statement specified by cursor_name retrieves and the type refers to the type of report that it represents. The main block column signifies the starting block to use, only one of these can be set to Y for a given report.

Types of Formula/Record

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 formula table has columns and a formula name. The block id refers to the block that this formula is part of and the seq refers to the sequence in the block. The next_block column signifies that after this formula has run the cursor defined by that block should be opened and that block's formula should be run until there are no more rows for that cursor.

The frequency column is used by the intermediate formula to specify the number of rows to be skipped before the formula is run. The O / F (overflow) column 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.

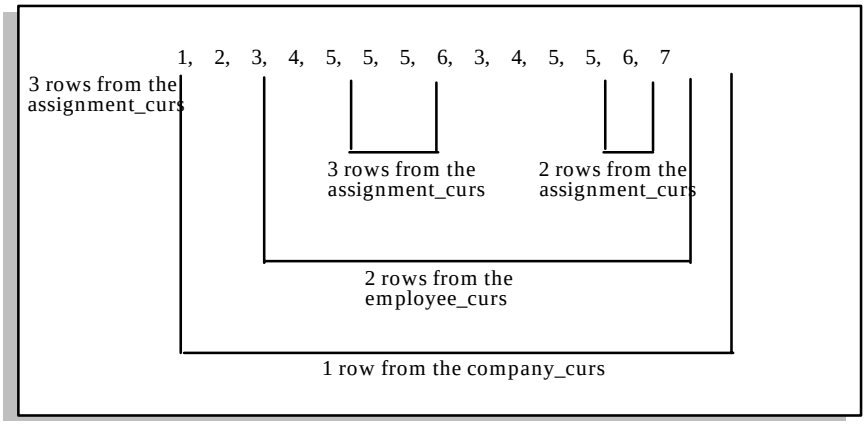
The exec.last column can apply to all the types of formula but most commonly the intermediate formulas. This column specifies that the formula can run one extra time after the last row has been retrieved from the cursor.

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:



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 records. Oracle suggests having a formula associated with (generating) each record type. A PL/SQL stored procedure provides the main control flow and determines the order in which the formula 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.

REPORT FILE	This writes the string result to the ”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>.

The transfer results follow a similar, but more stringent convention in that 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, this is achieved by embedding the file number in the formula return name, as in REPORT1_<result_name>, this writes to report/audit file 1.

Error Types

The errors can be of 3 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>.

Transfer Input/Return

The transfer input and return values are used simply to transfer values between the PL/SQL and the formulas.

Writing to the Tape File

If the formula returns a value with the name prefixed by WRITE_, the value is written to the magnetic tape ASCII file. The writes are performed in the order in which they are returned from the formula.

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 returned from formulas can be at three levels, as described in Error Types: page B – 42:

To initiate a payroll action level error the name prefix for the returning variable should be ERROR_PAY_. Similarly with the assignment level errors the prefix should be ERROR_ASS_. And finally the names for the warning messages is prefixed with ERROR_WARN_.

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 setup 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 formulas 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.

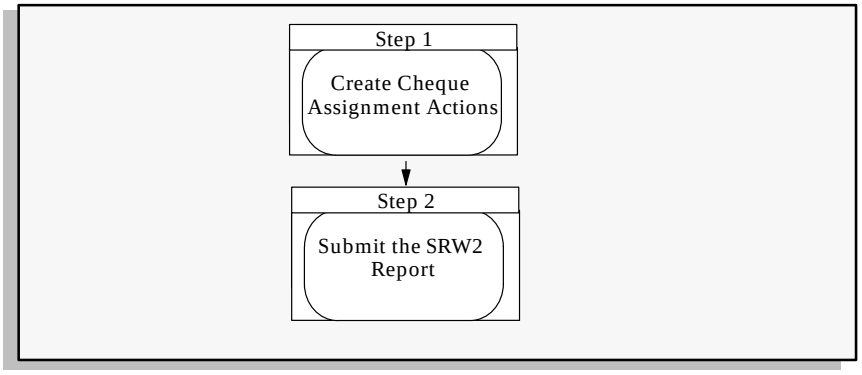
You run the Cheque Writer process to produce cheque payments for unpaid pre–payment actions. Before you run the process, you need to set up certain things, for example, the SRW2 report and the 'order by' option to sequence cheques, (if required).

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, they never have 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 interlocks back to previous actions.

   This happens in the same way as Pre–Payments and Magnetic Tape create assignment actions.

See: The Process: page B – 24 (Pre–Payments)

See: Magnetic Tape Payments: page B – 31

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 customise. 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 B – 56

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 are changed to 'Complete'. If the process fails, the status of the assignment actions are 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, in that, 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 organisation payment method. It would be used where different cheque styles are required by organisation 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 as part of the core product.

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 which 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 authorised, and has the cheque number preprinted on it. It is common in the U.K. for there to be a further cheque number box which is populated when the cheque is finally printed. It is this number which 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 reissue 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, if during a print run your printer jams on a single cheque and thinks 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 which has been created. The payroll action stores the start and end cheque numbers specified, and 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, then they must rollback the Cheque Writer payroll and assignment actions, and submit a separate batch request.

> **Note:** Marking a random spread of assignment actions within one Cheque Writer action for retry, is an unlikely operation. There is absolutely no guarantee (indeed, it is extremely unlikely) that the cheque numbers of the marked assignment actions are contiguous. Considering the issues of matching cheque numbers on the actions and on the preprinted cheque stationery, a print file of such non–contiguous cheques is unlikely to be useful.

SRW2 Report

The localisation teams 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 Quickcode 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 a directory, such that:

```
if $APPLCSF is defined

 write to $APPLCSF/$APPLOUT

else

 write to $PAY_TOP/$APPLOUT

end if
```

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 which 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. Organisation
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.

These changes should be made by the Localisation team, as part of the implementation process. 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 Quickcode Type is CHEQUE PROCEDURE. Place in the meaning field a meaningful description and in the description field the name of the option, for example, NEW ORDER BY.

Cash Process

The Cash process indicates 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 Centre
- 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 centres for the General Ledger
- accounts for cost centres and product lines within cost centres, for labour distribution purposes

Sample Payroll Results

Employee	***Work Structure***		***Earnings and Deductions***			
	Cost Centre	*Product Line*	*Salary*	*Wages*	*Overtime*	*Union Dues*
Employee 1	Production	H201 100%		1,000	400	20

Sample Payroll Results						
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 the sample run results previously displayed.

Example Costing Process Results							
Account Code	***Cost Centre***		***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 centre Production and the product lines H201 and H202.

Overhead Distribution for the Production Cost Centre

```
Total paid to Production Cost Centre as Wagesrun
result: $3,000.00
Total paid to Production Cost Centre as Overtime run
result: $1,000.00
Total for Earnings types specified for Distribution:
$4,000.00
Ratio for Wages distribution, Production Cost Centre
= 3000/4000 = .75
Wages overhead = Pension Charge 100 x .75 = 75.00
Ratio for Overtime distribution, Production Cost
Centre = 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 Centre and Production Line Totals

Account Code	*Cost Centre*	*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* (that accumulates costing results), you are ready to transfer the results to the General Ledger or other systems.

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 (that prepares the payments according to the payment methods) uses the results of the Payroll Run process (that 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 different payroll processes (such as Payroll Run and Costing) and action types (such as QuickPay and Purge), are classified as Sequenced, Unsequenced or Non Removable. 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 Initialisation	Sequenced	No
Purge	Sequenced	No
””	Non Removable	—
Pre–Payments	Unsequenced	Yes
QP PrePayments	Unsequenced	Yes
Ext / Manual Payments	Unsequenced	Yes

Processes and Action Types	Classification	Insert Interlock Rows?
Magnetic Tape Transfer	Unsequenced	Yes
Cheque Writer	Unsequenced	Yes
Cash	Unsequenced	Yes
Costing	Unsequenced	Yes
Transfer to GL	Unsequenced	Yes

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 B – 66.

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.

Non Removable Actions

This is used for 'Purge' actions. Actions can only be deleted by themselves.

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 rollback the monthly pay run for that employee if you have subsequently processed and paid them on the weekly payroll. You would have to rollback the payments process for the weekly assignment before you could rollback 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.

The existence of an interlock prevents subsequent insertion of other actions (and interlocks), as opposed to the order of when processing was performed.

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, validation is performed to ensure there are no sequenced assignment actions marked for retry existing in the past for any assignments that we are attempting to process.

The interlock rule is dependent on the legislation, and so is checked on an assignment or person basis.

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 after 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 assignment action is not inserted if there is an interlock for the assignment action currently being processed from another assignment action of an appropriate classification.

In other words, 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 summarises the rules for retry and rollback of payroll and assignment actions. For some processes, you cannot rollback actions only for an individual assignment, for example you cannot rollback an individual from the Magnetic Transfer process. This process actually produces the magnetic tape file so you must rollback the whole process, and then redo it.

Action Type Name	Payroll Retry	Action Rollback	Assignment Retry	Action Rollback
Payroll Run	Yes	Yes	Yes	Yes
QuickPay	Yes	Yes	Yes	No
Reversal	No	Yes	No	No
Balance Adjustment	No	Yes	No	No
Balance Initialisation	No	Yes	No	No
Purge	Yes	No	No	No
Pre–Payments	Yes	Yes	Yes	Yes
QP PrePayments	Yes	Yes	Yes	No
Ext / Manual Payment	No	Yes	No	No
Magnetic Tape Transfer	Yes	Yes	No	Yes
Cheque Writer	Yes	Yes	Yes	Yes
Cash	No	Yes	No	Yes
Costing	Yes	Yes	Yes	Yes
Transfer to GL	Yes	Yes	No	No

Rolling Back Sequenced Actions

You cannot rollback a sequenced action if there is a later sequenced action for the assignment, except for Balance Adjustments or Reversals. For example, you cannot rollback 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.

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(30)
```

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_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
THREADS	1 or more	1
TRACE	Y or N	N
USER_MESSAGING	Y or N	N

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 will determine 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 with 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.

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 rollback a specific process by deleting it from the *Payroll Process Results* or *Assignment Process Results* windows. When you rollback 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 will contain 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 QuickCode 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.

APPENDIX

C

Post Install Steps

This appendix describes any post install steps required for Oracle HRMS.

Post Install Steps for Oracle Human Resources

Warning: Be sure to check adpatch logs for errors. Although patch execution may appear to be 100% successful, errors may exist that require attention.

Attention: For more information about using adpatch, see AutoPatch Utility (adpatch), Release 11 in Chapter 4, Applying the Server Updates.

Step 1 Run the Generic HR Post Install Driver

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

$ cd $PER_TOP/admin/driver

$ adpatch

```
Apply the driver hr11gn.drv
```

Step 2 Run the Relevant Post Install Driver for the Legislation

UK and US users must install additional legislation specific startup data to enable them to make use of the soft–coded flexfield structures supplied with Oracle Human Resources.

If the legislation is UK:

$ cd $PER_TOP/admin/driver

$ adpatch

```
Apply the driver hr11gb.drv
```

If the legislation is US:

$ cd $PER_TOP/admin/driver

$ adpatch

```
Apply the driver hr11us.drv
```

Step 3 Load Schools and Colleges Seed Data for Career Management (Recommended)

Using the Schools and Colleges Attended form, you can maintain a record of people's attendance at schools, colleges, and universities. Complete this step if you want to load a set of US and / or UK universities and colleges as reference data.

Note: You can also enter schools, colleges, and universities manually using the Schools and Colleges form.

To load UK universities and colleges:

$ cd $PER_TOP/admin/driver

$ adpatch

```
Apply the driver hr11cmgb.drv
```

To load US schools, colleges, and universities:

$ cd $PER_TOP/admin/driver

$ adpatch

```
Apply the driver hr11cmus.drv
```

Post Install Steps for Oracle Payroll (US)

Warning: Be sure to check for adpatch for errors. Although patch execution may appear to be 100% successful, errors may exist that require attention.

Attention: For more information about using adpatch, see AutoPatch Utility (adpatch), Release 11 in Chapter 4, Applying the Server Updates.

Step 1 Run the Post Install Driver for US Legislation

US users must install additional legislation specific startup data to enable them to make use of the soft–coded flexfield structures supplied with Oracle Payroll (US).

To run the driver, type in the following commands:

$ cd $PER_TOP/admin/driver

$ adpatch

```
Apply the driver py11usi.drv
```

Step 2 Apply Additional Payroll Patches

For a US territory (under US legislation), you must apply the latest Vertex Cobol program update, which is available from Oracle Worldwide Customer Support, HR Applications Support Team.

Post Install Steps for Oracle Training Administration

There are no post install steps for Oracle Training Administration.

Glossary

A

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

Alternative Regions Parts of a window that appear in a stack so that only one is visible at any time. You click on the name of the region to pop up a list of the other regions in the stack. Select the name of a region to bring it to the top of the stack.

Applicant A candidate for employment in a Business Group.

Appraisal A 'superset' of recording opinions and setting and achieving objectives, plans and so on. See also: *Assessment*.

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.

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 on 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.

C

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.

Career Map A plan showing the expected routes by which employees can progress from one job to another within the Business Group.

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.

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 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.

Core Competencies 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.

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*

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*

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*

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.

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.

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

F

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

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.

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*

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

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.

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*

Leaver's Statement 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.

M

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.

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

O

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.

P

Pay Scale A set of progression points, which 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.

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

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.

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.

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*

Q

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

QuickCode 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 QuickCode Types, and you can add values to some predefined QuickCode Types.

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*

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*

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

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

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.

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*, *Report Security Group*

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.

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

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.

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, User Profile Options*

Special Information Types Categories of personal information, such as skills, that you define in the Personal Analysis key flexfield.

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*

T

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.

U

User Profile Options Features that allow system administrators and users to tailor Oracle HRMS to their exact requirements. See also: *Responsibility, Security Profile*

V

Work Choices Also known as Deployment 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, or at person level.

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

P

Q

R

S

T

V

W

Reader's Comment Form

Oracle HRMS US Implementation Guide
A58332–01

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.