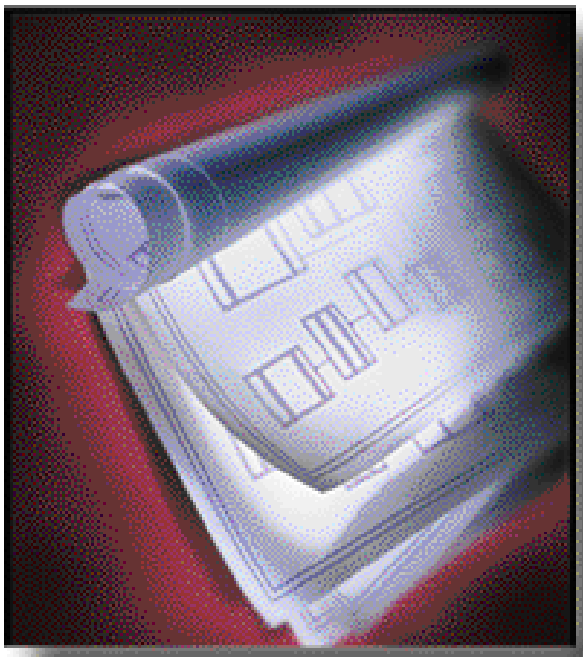
Oracle9i Designer Migration Guide

Contents

Oracle9i Designer Migration Guide

August 2002



Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication and disclosure of the Programs including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be licensee's responsibility to take all appropriate fail-safe, back up, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle8*i*, Oracle9*i*, PL/SQL and SQL*Plus are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.


Next


Contents

Oracle9i Designer Migration Guide

Contents (summary table)

Detailed contents

Part 1 Introduction, enhancements and new features

Part 2 Migrating the repository

Part 3 Generating and migrating existing database designs to Oracle9i Designer

Part 4 Migrating generated Forms applications to Oracle9i Designer

Part 5 Migrating generated WEB/PLSQL applications to Oracle9i Designer

Appendix A PVCS/VM administration query

Appendix B Quick reference information

Prev Next

Oracle9*i* Designer Migration Guide
Part 1. Introduction, enhancements and new features

Contents

Chapter 1 Introduction

This migration guide provides essential information for those who wish to migrate to Oracle9i Designer from Oracle Designer 1.3.2, 2.0 or 6.0. A previously published guide covered migration to Oracle Designer 6i. Migration from Oracle Designer 6i to Oracle9i Designer is unnecessary because certain 6i and 9i repositories are compatible, as follows:

- Designer/SCM 9i 9.0.2.0 has same repository level as 6i 4.2 (4.0.12.80.6)
- Designer/SCM 9i 9.0.2.1 has same repository level as 6i 4.3 (4.0.12.88.2)

This means that if you wish to transfer from 6i to 9i:

- from 6i 4.2 to 9.0.2.0 is a client change only, so no repository upgrade is required
- from 6i 4.2 to 9.0.2.1 requires a client change and repository upgrade
- from 6i 4.3 to 9.0.2.1 is a client change only, so no repository upgrade is required.

Note that you cannot transfer from 6i 4.3 to 9.0.2.0 because 9.0.2.0 is a lower repository version level than 6i 4.3.

Benefits and reasons for migrations

Oracle9i Designer offers the following recent enhancements:

- version control of repository objects
- workspace management (workareas)
- storage of text and binary files
- support for release management (via configurations)
- impact analysis tool (dependency analyzer)
- support for Oracle 9*i*, Oracle Reports 9*i* and Oracle Forms 9*i*
- support for repository management operations via an extended API
- support for repository management batch operations via a command line tool

Note: At the time of writing, Oracle9i Designer is certified with Oracle database 81700 or higher, and with Oracle9i patched to 9.0.1.2. See metalink.oracle.com for the latest situation.

Specifically, the option of version control of repository objects and related functionality (e.g. workspaces, configurations and storage of file system) offers the following non-tangible benefits:

- improvement of the functionality of the client generators (e.g. Oracle Forms 9*i*,) and server generators (e.g. support for Oracle9*i*).
- introduction of a configuration management solution that is fully supported with a standard Oracle tool set (instead of a solution with a third party source control tool (e.g. PVCS/VM, ClearCase)

- the availability of a single repository that captures structured elements and files during the complete lifecycle. A single repository fully supports model-based development and deployment and therefore improves the quality of the development and deployment process.

You could also identify the following tangible benefits as a result of adopting Oracle9i Designer:

- More efficient method to detect, analyze and solve issues via rich impact analysis tools (Dependency Manager)
- More efficient effort to build and verify multiple releases
- More efficient method to merge different releases (compare and merge facilities)
- More efficient method for building delta scripts via accuracy of the repository and richer supporting tools (DDL generator

Given this new functionality and benefits you may want to adopt Oracle9i Designer for one or more of the following reasons:

- Flexible and effective support of development of multiple releases simultaneously
- Effective prototyping tool – easy to create and store multiple alternatives via the version mechanism
- Synchronization between the logical and physical data model in Designer
- Synchronization between the database model implementation and the database definitions in Designer – high quality of meta data
- Usage of versions strings to effectively verify the database objects against the repository content
- Improvement of impact analysis capability - including the storage of column dependencies
- Improvement of functionality and performance of the creation/generation of database delta scripts
- Actualization of the current Designer tool stack to latest available technology.
- Cost effective access to resources for development and support (Oracle9i Designer/Developer 9i experts as opposed to Designer 1.3.2/Developer 4.5 experts)
- Provide a widely shared development and deployment environment
- Recognition of configuration management needs (multiple releases, release and build management)

The above mentioned benefits, offered by the new functionality, can be attained only by migrating your structured data (captured in application systems in previous releases of Oracle Designer) and/or files (captured in a third party source control tool or on the file system) to Oracle9i Designer.

Migration versus upgrade and migration scenarios

The term migration refers not only to the upgrade of the client software and server side software. It also covers the migration of the repository content, populating the repository with files and redesign of the migrated data.

Once your structured elements and files are in Oracle9i Designer (covered in the second part of the migration guide) you can apply five (potentially) different migration scenarios to your Database elements, Developer components (e.g. Forms, Reports, Menus and libraries) and WEB PL/SQL modules.

The migration scenarios listed below are described in detail (e.g. characteristics, steps, when to apply) in Part 3 (migration of database elements), Part 4 (migration of developer components) and Part 5 (migration of WEB PL/SQL modules). Note however that in these parts a specific scenario may be skipped because of its complexity and/or non-applicability.

Scenario 1. Migrate, Regenerate All, No Redesign

In this scenario, you regenerate your entire application from Oracle9i Designer, including all database elements (e.g. tables, views, PL/SQL definitions), Developer components (e.g. forms, libraries, menus, reports) and WEB PL/SQL definitions.

Scenario 2. Migrate, Regenerate All, With Redesign

In this scenario, you regenerate your entire application from Oracle9i Designer, including all database elements (e.g. tables, views, PL/SQL definitions), Developer components (e.g. forms, libraries, menus, reports) and WEB PL/SQL definitions. As you regenerate each database element and/or module, you make use of new features as appropriate.

Scenario 3. Migrate, Regenerate Incrementally

This is the most complex scenario. In this scenario, you migrate your application a little at a time, rather than all at once. You begin by upgrading your database elements in the context of an Oracle9i database (basically do they compile against an Oracle9i database?) then upgrading all of your forms, libraries, menus, libraries, etc. to Oracle9i Developer at the client. You will then make the changes required to run forms, menus, etc. and database elements generated from your previous release of Designer, alongside forms, menus, libraries, etc. and database elements generated from Oracle9i Designer.

Scenario 4. Migration only of developer components and database components

The first three scenarios all eventually require you to regenerate your application. Any post-generation modifications are lost. If you heavily modified your application post-generation you might consider upgrading the runtime environment to Forms 9*i*, upgrading your database elements to Oracle9i, and not upgrading both types to Oracle9i Designer. This implies that all future maintenance has to be done manually in Developer. You could however use Oracle Repository as your file source control tool by uploading all sources (including the generated WEB PL/SQL code) into Oracle Repository.

Scenario 5. Design Capture of database elements and developer components

If you heavily modified your application post-generation, and you would like to adopt (again) a 100% generation strategy, you may want to consider using the Design Capture features of Oracle9i Designer.

Usage and benefits of a pilot migration

The introduction and usage of a (migrated) Oracle9i Designer repository will have significant effects on the working method and writing method of the Oracle Designer repository participants. New configuration management roles will be introduced and/or the task set of existing configuration management roles will change significantly. Therefore you should consider organizing an Oracle9i Designer pilot project for the following reasons:

- Determination and verification of a migration strategy for migrating to the Oracle9i Designer repository and an investigation of the need for (additional) migration utilities
- Preparation of the participants for the new configuration management concepts

- Creation of a preliminary set of configuration management standards and guidelines
- Determination of the database and hardware requirements of a database instance that will capture your Oracle9i repository.

Chapter 2 What has changed in Designer?

This chapter describes enhancements found in Oracle9i Designer.

Storage of structured elements

A structured element is an element whose internal structure (secondary elements, references and properties) is fully known to and understood by the repository infrastructure. The main categories of Structured Element Types at this moment are the Oracle Designer element types (Entity, Business Function, Table Definition, etc.). In addition, Oracle Repository has a small set of core element types of its own, such as Folder and Configuration.

Each Element has properties. Properties can be optional or mandatory, system maintained or fixed or updatable, free text or bound by value domains. The properties come in three categories:

- singular values of data type Number, Date or Varchar2 with a fixed maximum length
- multi line text fields with no fixed length
- references to other objects

Most properties are specific to an element type. A small set of properties is generic: Name, Type, Version Label, Audit properties and IRID. The IRID replaces the ID known from earlier releases. The IRID identifies the Object, not just within its originating repository, but globally. The algorithm that derives the IRID (the SYS_GUID function in Oracle 8) returns a value that is guaranteed not to be produced again, in any repository. You can probably see the benefits this has for migrating data between repositories.

Storage of unstructured elements or files

All elements that have a structure that is unknown to the repository infrastructure is, somewhat strongly, labeled Unstructured Element. Note that this term does not claim such elements are in fact without internal structure, it merely states that the repository is unaware of that structure and therefore can only handle the element as a whole. Note that even though the Dependency Parsers know enough about the structure of files such as FMB, RDF, OLB, PL/SQL or Java to extract the dependency details, inside the repository these files are for the rest still managed as singular, unstructured objects.

The ability to store Unstructured Elements or Files in the repository is one of the major changes in the repository architecture. It is a change that will alter the role of the repository significantly. From now on **all** project deliverables and system elements can be stored and managed inside the repository! All Run Time Objects (executables), Sources, Meta-Sources (Templates, Designs, Models, Workshop Notes, User Interviews) and Supporting Objects (Test plans, Project Plan, Standards and Guidelines), whether they are structured or unstructured, can all be made part of the repository. And it is our strong recommendation that they all indeed *are* stored and managed in the repository.

Files are stored as C(haracter) or B(inary) LOB (Large Binary Object) in the table I$SDD_FILES. Files are created and updated through Uploading from any file system and can be extracted to any file system using the Download operation. Upload and Download can be performed on single files, groups of files or entire nested

directory structures, from the RON or the Command Line Tool (on Windows NT). You may *Open* a file by double clicking it in the RON. Based on the link defined in the Windows Explorer (View, Options, Registered File Types) and stored in the User Profile in the Windows Registry, the indicated tool is used to open (view, edit or run) the indicated file. This allows every user to use his or her favorite editor for opening files from the repository.

Files can be stored in compressed format. The compression and decompression is handled automatically by the repository on upload and download respectively. The compression algorithm is very fast and helps reduce storage space required for files to 2-10%. In the Repository File Registry is recorded which file types (by extension) should be stored as Text (CLOB), Binary format (BLOB) or in compressed format (always Binary). Note that Compare is available for Text Files and the binary files that are produced by Oracle Developer, such as FMB, PLL, OLB and RDF.

Workareas

A workarea is a user's view of the repository objects on which he or she is working. Within a workarea, repository data is further broken down into application systems and folders, which are known as containers.

In previous releases the systems modeling tools presented repository data in the context of a single application system. In Oracle9i Designer, repository data is presented in the context of a workarea, enabling multiple application systems and folders to be accessed from each tool.

From the familiar Navigator window, you now interact with the repository through a workarea, which is your own particular view of the repository contents. If you create a workarea, you are the owner of that workarea, though you can assign other users access rights to it. You must have access to at least one workarea before being able to do any work in the repository.

Other repository users may make changes to objects that are visible in your workarea. These changes are not immediately apparent to you in the Navigator window, so a workarea refresh mechanism is provided to bring the display up to date with the latest changes. You can preview the effect on the Navigator window display of refreshing a particular workarea, by listing the objects visible in the workarea that have been created, changed or deleted since the last refresh.

The contents of a workarea are initially defined automatically by the application of rules. However, you can manually override these rules and explicitly include any repository object in the workarea, or exclude any object that is already there.

Containers

You now create a repository object in the context of a container, a logical subdivision of the repository. Two types of container are supported in this release: application system (similar to that available in previous releases) and folder (similar to a directory in a disk-based file system). A repository object is created as a child object of its parent, or owning, container.

Version control

Overview

A completely new system of version control of individual repository objects replaces the existing application system level versioning available in previous releases. A checkout/checkin mechanism ensures that different users do not overwrite each other's changes. Graphical tools are available to display the version history of an object, to display object version details, and to display details of various version control events that have taken place for an object.

A branching mechanism permits the development of multiple versions of a product simultaneously using the same set of source files. Branches can be labeled for ease of identification.

Objects can be checked out with a lock, in which case no other user can perform a locked checkout of any versions of the object on that particular branch. Other users can still check out unlocked versions of the object on that branch, but they cannot check them back in on that branch until the original user has released the lock.

Objects can also be checked out unlocked, permitting other users to check out any version of that object without a lock. In this situation, the first user to check the object back in is permitted to do so, whereas subsequent checkins of the object must be merged with the latest version on that branch.

This release provides you with the ability to version and label your designs, for improved change control and multi user access.

Version control is the process of maintaining multiple versions of software development objects and is the most basic requirement for a software configuration management system.

Both repository and non-repository objects can be version-controlled in the common repository.

The repository manages all repository objects, as well as file system files and directories. This enables the tracking of changes to the organization of the source data, as well as tracking the changes to the contents of individual files.

Versions are organized into a version tree structure with branches. Branches have user-defined labels, typically chosen to indicate their role in the development process. All objects can be identified by a system-generated version number. Important versions can be assigned version labels by the user to indicate development milestones.

Version history

Each object in the repository is held as a set of snapshots (called object versions) of its state at certain points in time. The object versions are linked together by associations that record how each object version evolved from its predecessors. This web of associations, represented by the version tree, is called the version history of the object and can show succession, branching and merging.

The version history associations also have a number of version control attributes, including:

- a unique version identifier
- the identifier of the user who created the object
- the date and time the object was created
- the identifier of the user who changed the object
- the date and time the object was changed
- status of the object (for example checked out or checked in)
- a comment (a descriptive reason for checking in or checking out)

Checkin and checkout

Version control starts when an object is checked in. Objects that are checked in are write protected. The object must be checked out before it can be modified.

The checkin and checkout process is initiated from the Repository Object Navigator, or from the Version History Viewer utility or from the command line interface.

Only one version of an object can be included in a workarea and only one version of an object is visible through the workarea view. However, the complete history of an object and all the events that occurred to it can be viewed through tools provided by the repository.

Any file or folder in your file system, as well as repository objects, can be the subject of version control. You can copy files and folders into the repository and copy them back to the file system. You can also synchronize your file system with the repository at any time to ensure both are consistent.

When checking an object in or out you can provide notes to briefly describe the reason for the action.

Once the object is checked out, a copy of the object is moved into your workarea. Your workarea contains all the checked out objects you are working with. When the object is checked out, the workarea is refreshed so that the new object version appears in your version resolved view and can be edited by you.

You can undo a checkout but any changes you have made to the object are lost. The system administrator has the privilege to cancel a checkout.

Branching

When checking in an object, you can decide if the changes are to be included in the branch from where the object was checked out, or if the changes are to be represented on a new branch. Branch labels can provide some meaning to the development strand, for example, as a set of changes related to bug fixes.

Many of the checkin options are available by default. For example the default action could be to create a new branch, with a specified label. This default would apply to all objects checked in from your workarea. This could enable all the changes made on a given set of objects to be queried from the repository and used in release management or quality assurance procedures.

Locking, comparing and merging

During the checkout operation, the object can be locked so that no other users can check out a copy for editing. If you choose not to lock the object, there is the possibility that when the time comes to check the object back in, another user may have made changes in the interval. In these circumstances, and depending on who checks their version in again first, the compare and merge tools have to be used to ascertain how to check the object back in.

The repository currently supports comparing and merging of three object types:

- structured objects (standard repository objects)
- Oracle forms
- text files

However, it is possible that your organization has added support for additional types.

Version control tools

The following version control tools are provided:

- **Version History Viewer**. Illustrates the version history for a chosen repository object in diagrammatic form and allows you to perform version operations such as compare and merge.
- **Version Event Viewer**. Provides detailed information about the check in and check out version operations that have been performed for a chosen repository object. The information is provided in dialog box format.

The version control tools enable you to perform a number of tasks on both repository objects and file system objects. These include:

- checking in and checking out objects to create new object versions
- viewing the history of an object
- viewing the events that occurred to an object
- creating new version branches
- comparing object versions
- merging object versions
- mapping file systems to the repository
- viewing object version details

The Version History Viewer and the Version Event Viewer are available from the Repository Object Navigator and can also be used as standalone tools.

Configurations

A configuration is a collection of repository object versions that are related in some way that is significant to an application. Usually a configuration represents all the object versions for a checkpoint in development or component of an application, for example, all the object versions that make up a payroll application, or a screen form used in that application.

Any set of specific object versions can be recorded as a configuration, rather like striping or labeling in some configuration management systems. A specific object version that is included in a configuration is said to be a member of that configuration.

For example, when the development of individual objects reaches the stage where you can build a particular application, you need to specify exactly which version of each object is to be used for the build. The same applies when assembling a set of objects to be used for a test or included in a patch release - the configuration defines which versions of which objects are to be used.

Oracle9i Designer provides a configuration wizard (a predefined set of screen forms) that takes you through the creation process step by step.

The configuration wizard allows you to create a new configuration (or a new version of an existing configuration) containing, for example, all the latest object versions.

An object version that is included in a configuration is referred to as a member of that configuration. Members can be added or removed as and when required. However, only checked in objects can be included in a configuration. Each configuration is a repository object in itself, and can therefore be versioned.

To create a new configuration using the wizard, you start by entering the configuration name and description.

You can base the configuration on:

- a template file containing rules and configurations
- the contents of a workarea
- the latest versions of objects on a specific branch.

You can also create a custom configuration.

You then add, remove or change the version of members as required.

If you choose to base the configuration on a template file, workarea or latest versions on a branch, you may then refine it by adding or removing members or changing the versions of members. You can also add or remove selected rules and configurations.

When you create a new version of an existing configuration you have all the features that are available when creating a new configuration. In addition, you can simply refresh all existing members with the latest versions or refine the existing configuration as required.

All external references are displayed, and you can include externally referenced object versions into your configuration.

When you have completed your new configuration or new configuration version you can either save the configuration only, or save it and check it in to version control. You can add checkin notes and specify a checkin branch.

If you choose only to save the configuration you can edit it later without creating a new version.

Folder and file synchronization

When you upload a file, the system copies the file from the file system to the repository. Both copies can be opened and edited independently, and this could lead to differences between the two copies of the same file.

Some means is necessary of ensuring that the two copies are kept up to date with each other. Oracle Repository provides a synchronization mechanism that enables you to keep the repository copy up to date with the file system copy or vice versa.

When you perform synchronization, the system checks whether any changes have been made to the repository copy, the file system copy, or both.

The resulting actions are:

Repository copy	Repository copy	File system copy	System Action
Unchanged		Unchanged	None
Unchanged	Checked in	Changed	Displays dialog for user intervention
Unchanged	Checked out	Changed	Uploads copy from file system
Changed		Unchanged	Downloads copy to file system

Changed	Checked in	Changed	Displays dialog for user intervention
Changed	Checked out	Changed	Displays dialog for user intervention

Files where the file system copy has been changed while the repository copy remained checked in are known as "hijacked files". For these and other cases where a dialog is displayed for user intervention, you can choose to upload the file system copy, download the repository copy, check the repository copy in or out and resynchronize, or merge the two copies, resolving the differences one by one.

A few special cases are worth noting:

- If you change the repository copy and subsequently delete the file system copy before saving the changes, those changes are lost.
- If you delete the repository copy but not the file system copy before synchronization, the repository copy will reappear on synchronization. This is because in this case it is treated as a new file and uploaded accordingly. The same applies if you exclude the repository copy from your workarea.
- If you delete the repository copy but not the file system copy after the two have been synchronized, subsequent synchronization operations offer you the choice of deleting or retaining the file system copy. The same applies if you exclude the repository copy from your workarea.

System privileges

Repository owners and subordinate users need system privileges to use the repository. For example:

- a repository owner needs system privileges to install a new repository
- a subordinate user needs system privileges to create database objects.

You use the Check Requirements tool on the Repository Administration Utility to assign system privileges.

Note: Only the Oracle database administrator can assign system privileges.

Access rights

The functionality for repository user maintenance has been greatly extended. The repository owner can now specify whether a particular subordinate user can perform certain repository management operations, such as creation and deletion of workareas, or force delete or purge operations on repository objects. In addition, the repository owner can control a subordinate user's access to individual repository tools, such as the Repository Object Navigator or the Command Line Interface.

Repository policies

You can set repository-wide options that control certain aspects of:

- version control
- file naming (case sensitivity)
- object dependencies

Version control policies

A user with the Set Policy system privilege can impose restrictions on the following versioning features:

- branching
- version labeling
- locking

By setting policies you can ensure that specific development branches are not populated with new object versions at critical periods in development, for example during a build. It also ensures adherence to system-specified object version labels.

Automatic branching

When an automatic branching policy is set, all checked out objects will check in to the default checkin branch set for each workarea.

If a default checkin branch is not set for a workarea, object versions in that workarea will check in to the branch they were checked out from.

Automatic version labeling

When automatic version labeling is set, the user is unable to specify a label for an object version on checkin. The repository will automatically assign a label.

Strict locking

When strict locking policy is applied, users are forced to lock objects on checkout. Because two or more users cannot apply locking to object versions on the same branch, a strict locking policy prevents concurrent development of object versions on the same branch.

Without a strict locking policy in force, a user can choose whether or not to lock an object on checkout. In this situation, locking does not prevent another user from checking out an unlocked copy of the same version. It only prevents that user from checking in on the same branch until the locked version is checked in.

File naming policy

The case sensitive uniqueness policy ensures that case sensitivity is applied to the names of file system files and their containers when these are created in the repository. When the policy is in force, for example, you are able to create a file named "MyFile" and a different one named "myfile". When the policy is inactive, these two files would be treated as the same file, so creation of the second one would not be allowed.

This policy is of particular use for files that are uploaded from a UNIX server, where case sensitivity is a feature of file and directory names.

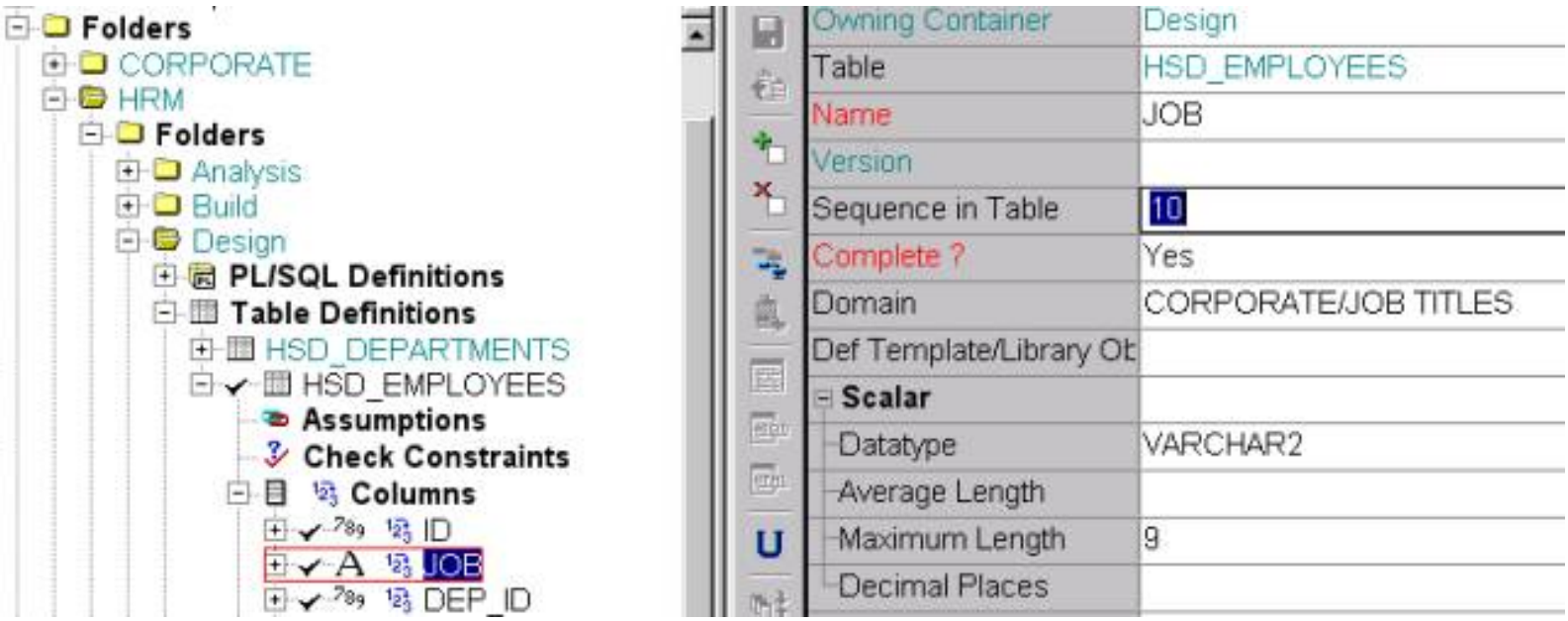
Dependency policies

Further options control whether object dependencies are copied by default when a repository object is either versioned or copied. In either case the default can be overridden at the time of the operation.

Short cuts versus shares

Shares as we know them in Oracle Designer 6.0 and before no longer exist. You are allowed to create references to elements anywhere in the repository, provided these elements are owned by folders that you have select privileges on. You do not need to create a share to the context folder. In the figure below Column Job in Table Definition HRM/DESIGN/HSD_EMPLOYEES has a reference to a domain JOB TITLES that is owned by Folder CORPORATE. This reference is created without sharing the domain to the HRM/DESIGN Folder. For those amongst you who suffered from dummy and skeleton application systems: they are a thing of the past! Moving an element from one folder to another will not create shares for referenced objects, as was the case for the transfer operation in Oracle Designer 6.0.

The following shows that no share is required in order to have a reference to a domain in a different folder:



A valid question now would be: does this mean we do not have shares at all anymore? The answer is: well, no, but we do have something very similar. And to overcome any ill feelings you may harbor against *shares*, this new mechanism is called *short cut*. It is indeed very much like the File System Short Cut you may know from MS Windows. And it offers some nifty features.

A *short cut* is a link between a Folder and an Element, with the ownership flag set to *false*. Short cuts are created by dragging and dropping the element to the Folder (or Edit, Copy; Edit, Paste Reference). Select privileges are required on the owning Folder and Update privileges on the target Folder. Links between a Folder and an Element with the ownership flag set to *true* indicate the Owning Folder link. Note that you cannot create a short cut for a Folder, only for individual Elements. Also note that a short cut applies to all versions of the element it is created for. Creating a short cut from Folder HRM/PUBLIC to Table Definition HSD_DEPARTMENTS means: linking every single version of HSD_DEPARTMENTS to HRM/PUBLIC.

In the context of a Folder, you will see all its owned Elements as well as its short cuts. That means that you can see and reference elements without having to search and navigate through the entire repository. Furthermore, if you click on the little hand icon or press *F12*, you will navigate directly to the source of the shortcut. New (compared to *shares* in Oracle Designer 6.0) is the ability to *edit-in-situ*: if you have update privileges on the Folder that owns the Element, you can directly update the Element short cut without having to navigate to the owning Folder. Another step forward is the ability to copy a short cut. That means: you can drag & drop a short cut to a Folder, thus creating a short cut to the same Element that the original short cut links to.

Chapter 3 A version enabled repository versus a non-versioned repository

You should realize that you can still use Oracle9i Designer as you always have before, i.e. use Oracle9i Designer as your repository for non-versioned structured elements only. In time you may consider the storage of files as well and finally enable the versioning of both element types. This chapter will provide you with an insight on how to use Oracle9i Designer, and it provides an overview of the relevant chapters of Part 2 "Migrating the repository" for each Oracle9i Designer mode.

Unversioned structured elements

If you do not want to version your elements - the reasoning on which to base your decision will be provided soon - you simply should not enable the version option in the Repository Administration Utility just after a fresh Oracle9i Designer installation. As long as the version option is not enabled you can store only "one version" of each element, just like in previous Oracle Designer releases. In fact the version properties of each element will not be populated, because you will never be able to check in an object. You will only commit your changes. Each participant will use the GLOBAL_SHARED_WORKAREA.

In addition you should instruct your participants not to store any file in the repository. Note that there is no system repository option available that can stop you from populating the repository with files.

You will typically use Oracle9i Designer in this specific mode in the following circumstances:

- The size of your project is quite small (e.g. less than 10 participants)
- You have not yet reached the test phase. You are still in the strategy, analysis, design or build phase. You do not need some kind of parallelism.
- The number of files to manage are limited (e.g. less than 30)
- You still want to use your third party source tool to manage your files

The following chapters of Part 2 "Migrating the repository" are relevant:

- Chapter 1 "Introduction"
- Chapter 3 "Migrate structured data"
- Chapter 7 "Migrate users and assign access privileges"
- Chapter 9 "A typical migration plan"

Obviously you may skip all sections in the above chapters that handle the versioning of data.

Unversioned structured elements with unversioned folder/file support

If you do not want to enter the versioned world, but you do want to store files in addition to structured elements, you also simply should not enable the version option in the Repository Administration Utility.

In addition you should provide your participants with guidelines on how to store files and folders in the repository. More specifically you should set up a folder hierarchy with appropriate access rights. See also Chapter 7 and Chapter 8 of Part 2.

You will typically use Oracle9i Designer in this specific mode - thus including the storage of unversioned files - in the following circumstances:

- The size of your project is quite small (e.g. less than 10 participants)
- You have not yet reached the test phase. You are still in the strategy, analysis, design or build phase. You do not need some kind of parallelism
- The number of files to manage are substantial (e.g. more than 30)
- You want to adopt a single repository approach and therefore you no longer want to use your third party source tool to manage your files

The following chapters of Part 2 "Migrating the repository" are relevant:

- Chapter 1 "Introduction"
- Chapter 3 "Migrate structured data"
- Chapter 4 "Migrate files from third party source control tools" or Chapter 5 "Migrate files that are stored on the file system"
- Chapter 7 "Migrate users and assign access privileges"
- Chapter 8 "Synchronize the file system and the database with the repository content" (specially the file system synchronization)
- Chapter 9 "A typical migration plan"

Obviously you may skip all sections of the above chapters concerned with the versioning of data.

Versioned structured elements with versioned folder/file support

If you would like to enter the versioned world, you simply enable the version option in the Repository Administration Utility just after a fresh installation and you provide at least some users with the container access right "version". See also Part 2, Chapter 7 "Migrate users and assign access rights".

You are likely to want to enter the versioned world for at least the following reasons:

- Your application is already in production, you have to maintain it and at the same time you would like to add new functionality and/or change existing functionality
- You want to improve the quality of your deployment process by using a variety of impact analysis tools
- You have to develop and support multiple releases for multiple customers
- You have to develop your application in multiple languages
- You have to support some other kind of parallelism
- Developers are losing work because they are frequently overwriting each other's sources. In other words there is a real need for workspace procedures.
- You want to keep intermediate copies or versions of specific elements in the context of prototyping.

Your migration efforts are more substantial. The following chapters of Part 2 "Migrating the repository" are relevant:

- Chapter 1 "Introduction"
- Chapter 3 "Migrate structured data"
- Chapter 4 "Migrate files from third party source control tools" or Chapter 5 "Migrate files that are stored on

the file system"

- Chapter 6 "Reorganize a migrated Oracle9i Designer repository"
- Chapter 7 "Migrate users and assign access privileges"
- Chapter 8 "Synchronize the file system and the database with the repository content"
- Chapter 9 "A typical migration plan"

Prev Next

Contents

Oracle9i Designer Migration Guide
Part 2. Migrating the repository


Contents

Chapter 1 Introduction

This second part provides different scenarios and related migration tools for the migration of structured data from previous Oracle Designer releases, and the migration of files either from a source control system (e.g. PVCS/VM, ClearCase) or the migration of files/folders directly from the file system. The different migration scenarios share the following common structure:

1. Preparation
2. Migration of structured data
3. Migration of files
4. Reorganization steps
5. Migration of users and access privileges
6. Synchronizing the file system and database

The *preparation* step basically involves the clean-up of your "original" elements and a fresh Oracle9i Designer installation. Preparation should be followed by *migrating your structured elements* via the migration wizard. You can then easily enter the versioned world by just checking in structured elements. By organizing the *files migration* after the migration of structured elements you are able to integrate your files with the structured elements - both types of elements will share the same root-container. The migration of both object types should be succeeded by *reorganization* steps that involve a simple implementation of a promotion model via a combination of workareas and configurations. The reorganized repository should then be made accessible to subordinate users. Guidelines were given for creating *subordinate users* and the distribution of the appropriate access rights on the previously mentioned objects. Finally a significant part of your repository content will ultimately populate your databases and file systems. You can implement the *file system synchronization* within each workarea via the folder mapping option and the *database synchronization* within each workarea via the DDL generator.

This second part is concluded with an example of a typical migration plan.

Chapter 2 Prerequisites and installation notes

You must precede the migration of structured data and files by an installation of Oracle9i Designer on the client side and a fresh repository installation on the server side in an Oracle8i or Oracle9i database instance.

The client and repository software is installed during installation of Oracle9iDS. The Oracle9iDS Installation Guide contains recommendations for hardware on the client and server sides. Once installed, you have to configure the repository in accordance with the Oracle9i SCM Repository Installation Guide.

This chapter contains guidelines for how to prepare a fresh Oracle9i Designer installation for migration.

Consult Chapter 3 in Part 1 for guidelines about entering or staying away from the versioned world. If you want to enter the versioned world you can enable element level versioning via the "Enable Version Support" option in the Repository Administration Utility directly after the fresh repository installation. Note that you cannot disable the versioning support option once you have enabled versioning, other than by creating a fresh Oracle9i Designer repository.

Verify the client side and server side installation

The verification of a fresh Oracle9i Designer installation involves both the client side and server side.

Verification of the client side

All software is now installed by the Oracle9iDS installation process. There is no longer a custom installation option.

Verification of the server side

There are two options available to verify the success of the Oracle9i Designer repository installation:

- Verification of the installation log files
- The "View objects" option in the Oracle9i Designer RAU

Verification of the installation log files

Verify the content of a fresh Oracle9i Designer installation log file, written to the ORACLE_HOME\repadm61\logs directory after the installation.

"View objects" in the RAU

While connected as the Oracle9i Designer repository owner, open the Oracle9i Designer Repository Administration Utility and launch the View Objects option to verify the server side installation. Sort the overview on the status property. A successful Oracle9i Designer installation may not display the following values for the status property of all objects:

- Disabled
- Missing
- Invalid

Obviously you could directly check if any of database objects of the Oracle9i Designer repository owner is "invalid" via the status property of the view "user_objects".

Prepare the Oracle9i Designer repository for migration

You can "run" the Oracle9i Designer repository either in a 'non-versioned' mode or version enabled. Multiple scenarios and criteria for versioning versus non-versioning are described in Chapter 3 of Part I: Version enabling versus non-versioning.

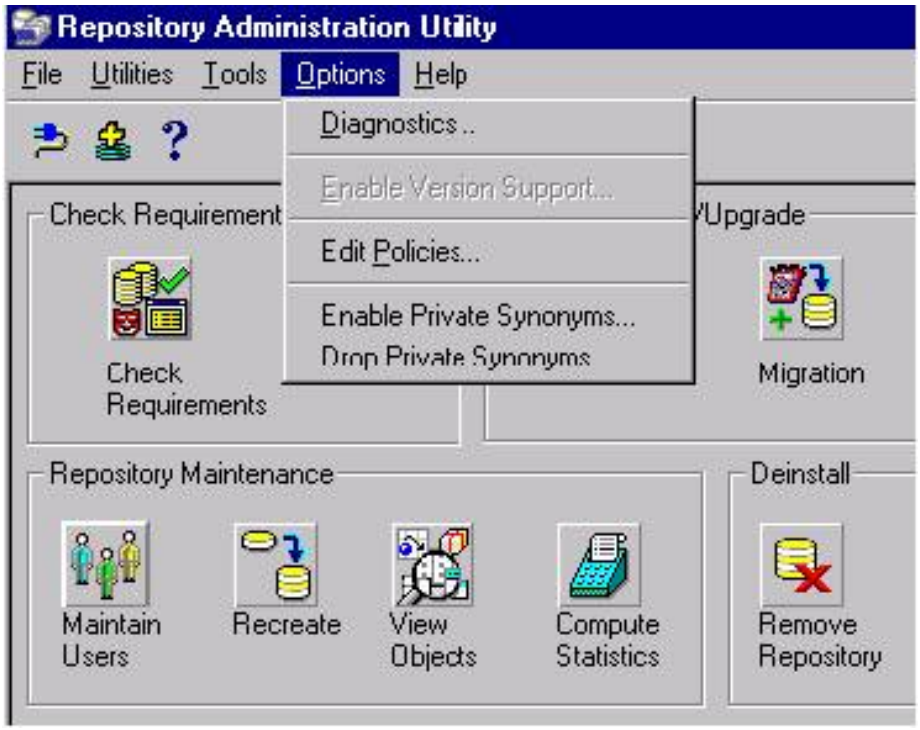
Skip the following activity if you do not want to enter the versioned world.

You can enable versioning via the Oracle9i Designer Repository Administration Utility (connected as the Oracle9i Designer repository owner). Subsequently enable the versioning option from the options menu if you would like to enter the versioned world.

Note that you cannot disable the version option once you have enabled versioning, other than by re-creating a fresh Oracle9i Designer repository.

Public or private synonyms

You can "run" the Oracle9i Designer repository either with public or with private synonyms. The enabling or disabling of private synonyms is available at the Repository Administration Utility, as depicted below.



The use of public synonyms is more productive - no more reconciles per user- and will occupy less space in the system tablespace. You do not need to create more than 1500 private synonyms per user. Note however that the use of public synonyms allows you to create only one Oracle9i Designer repository instance per database instance.

Reevaluate the file registry settings

Skip this section if you are not planning to load files in the repository.

Files in the repository can be stored as:

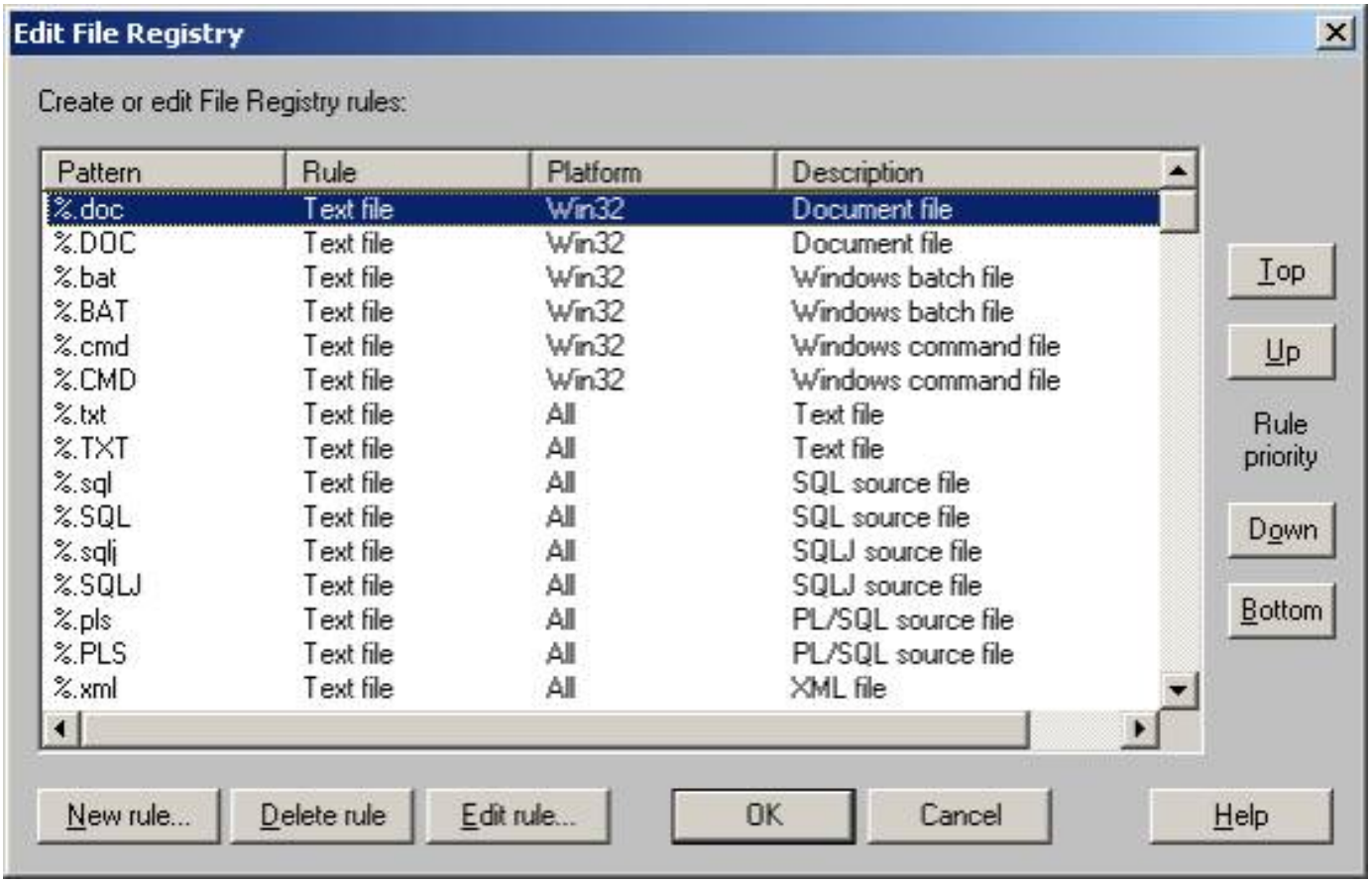
CLOB - Character Large Objects - Text file

BLOB - Binary Large Objects - Binary file

BLOB - Compressed

BLOB - Uncompressed

A file will be stored as one of the above methods based on its extension. These extensions and the subsequent storage method are stored persistently in a repository table - i$sdd_file_registry - that can be manuplated via the utility menu option Edit File Registry. Through this menu option new file extensions can be added and/or existing entries can be altered as illustrated below:



You should reevalute these file registry settings before the bulk loading of files from your third party source control tool or file system to enforce a cost effective and efficient storage of files. You cannot alter the storage method for a specific file and its subsequent versions once the file is uploaded in the repository. Any changes will only have an effect on new files.

Chapter 3 Migrate structured data

The fresh Oracle9i Designer installation should be followed by a migration of your structured elements. The following subjects for the migration of structured data are discussed:

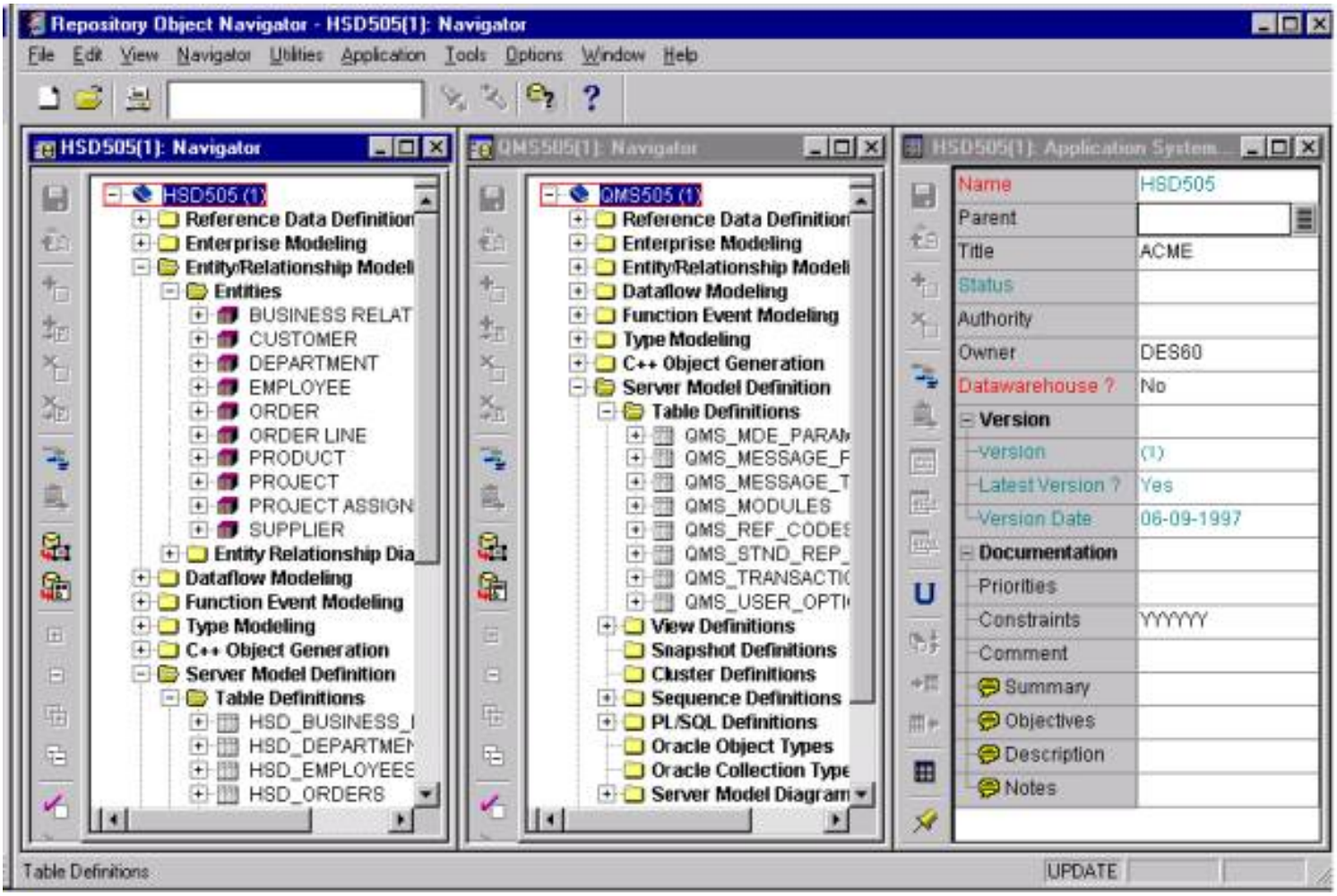
- An overview of the example of Headstart structured data
- An overview of different migration strategies for structured data
- Preparation steps
- Other hints and tips based on migration best practices of structured data
- Migration to an Oracle9i Designer repository (if applicable)
- Migration steps starting from an Oracle Designer 6.0 repository
- Post migration steps for structured data
- Verification of the migrated structured data

A great part of the activities of migrating structured data is also applicable if you do not want to enter the

versioned world. Any sections that are not applicable in a non-versioned repository will be announced in italic.

Example of Headstart structured data

The different migration scenerarios will be clarified using an abstract of 'Headstart 5.0.5' structured components and files. It is assumed that the structured Headstart 5.0.5 components are captured in two (Oracle Designer 6.0) application systems: HSD and QMS respectively, as illustrated below:



Migration strategies for structured data

There are two strategies for migrating the repository content to Oracle9i Designer:

- Migrate an entire repository content in a single operation
- Migrate a sub-selection of application systems within a specific Oracle Designer 6.0 repository

The first strategy loads and migrates all application systems into a fresh Oracle9i Designer repository.

The second strategy allows you to load and migrate one or more interrelated application systems. Non-selected interrelated application systems will stay behind in an Oracle Designer 6.0 repository. These application systems are candidates for migration at a later time.

Note that the Oracle9i Designer migration method has changed significantly. An Oracle Designer 6.0 repository will not be upgraded to an Oracle9i Designer repository during the migration.

The Oracle9i Designer repository content (or part of it) will be copied into a freshly installed (or already populated) Oracle9i Designer repository. Note also that the migration is only supported starting from an Oracle Designer 6.0 or Oracle Designer 2.1.2 repository - although it is highly recommended to update your Oracle Designer 2.1.2 to the latest patch level (patch 7 of an Oracle Designer 6.0 repository) before launching the migration wizard.

Migrate an entire repository content

There are several migration scenarios available to bring your structured objects forward to Oracle9i Designer. This section will discuss a 'Big Bang' migration or the migration of an entire repository.

The 'Big Bang' migration approach is applicable in the following circumstances:

- The structured elements in the application systems are closely interrelated. You cannot isolate a subset of application systems because of its (many) interdependencies (also known as shares). In addition there are strong relationships between the files captured in the third party source control tool and the structured data in Oracle Designer.
- The redesign activities - directly started after the migration - are applicable to all application systems simultaneously.

Migrate a sub-selection of application systems

You may want to adopt a staged migration scenario in the following circumstances:

- The Oracle9i Designer migration concerns a pilot or a test migration
- It may not be necessary to migrate all your application systems and all files in case of a pilot or an evaluation of your migration scenario.
- A coherent set of application systems will be migrated and subsequently redesigned in the Oracle9i Designer environment. At the same time other application systems within previous Oracle Designer repositories will or must stay behind.

  This migration approach may not need to be complex if you choose mutual independent sets of application systems for each migration. You can easily add new sets of application systems to your Oracle9i Designer repository as long as they are mutually exclusive. However if there is any overlap between your application systems your migration scenario will look like the following option. You should try to avoid staged migration of highly interdependent application systems.
- Support and therefore changes are needed on the "original" environment during the migration period of the structured elements and files. These interim changes must also be brought forward to Oracle9i Designer.

  This situation requires a second migration cycle - though on a smaller scale - specifically if you have to migrate a substantial number of applications. In this second migration cycle you have bring forward both changed structured elements and changed files from your "original" repositories to Oracle9i Designer. The migration for both types will be based upon an isolation of the changes in the "original" repositories and an isolation of the corresponding elements in Oracle9i Designer.

The necessary migration steps for the staged approach will be discussed later in this chapter. See "Post migration steps for structured elements within a staged migration approach".

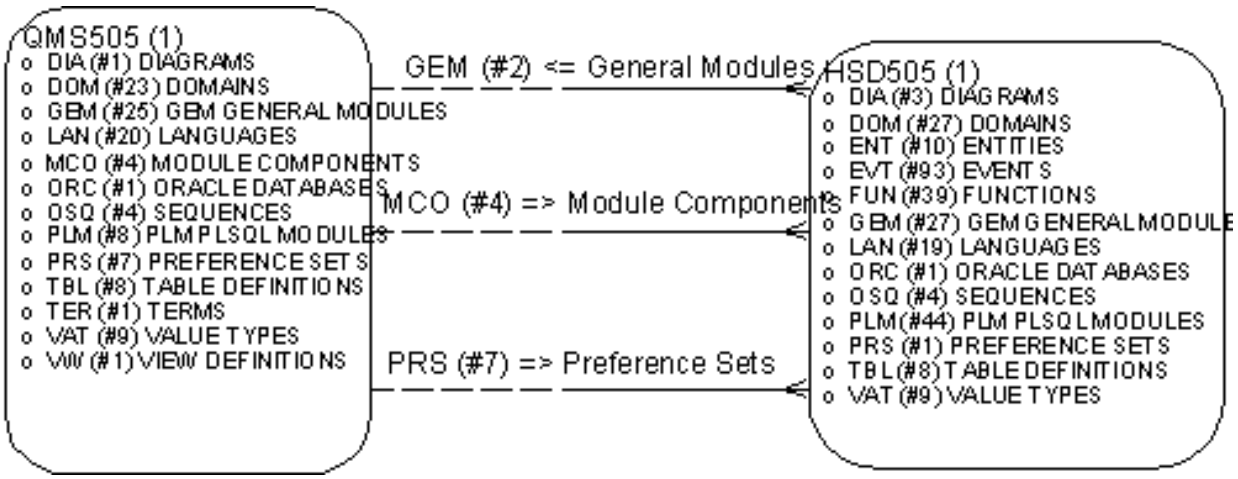
Preparing (cleaning up) your 'old' repository content

A clean-up of your old Oracle Designer repositories should precede the actual migration. It will not only save time during the migration but it will also limit future repository maintenance effort.

The following clean-up checklist is applicable to structured data:

- Remove obsolete users.
- Remove unused application systems. Candidate unused application systems are skeleton applications with no shares to 'actual' application systems or 'old' versioned application systems.
- Remove obsolete objects and unshare unused objects.
- Consider the removal of User Defined Sets. Although User Defined Sets are still available in Oracle9i Designer they will - by definition - not capture specific object versions. The object versions visible in a User Defined Set may change over time. You should use configurations instead if you want to capture specific object versions persistently like a change request.
- Reconnect your application systems that receive shares from 'obsolete' application systems to actual application systems. For example if HSD505 receives shares (e.g. preference sets) from QMS342, you could reconnect to for example QMS505 or higher by archiving HSD505 only and immediately restore HSD505 (as HSD505_copy) with the reconnect option to QMS505. Note that archiving the single application system HSD505 will result in a skeleton share from QMS342 and that the reconnect option - while restoring - allows you to reconnect the skeleton application to other application systems.
- Remove unused diagrams (e.g. old versions) and diagrams that can be fully derived from the repository content (e.g. Module diagrams).
- Consolidate all other diagrams. The visual diagram information is stored separately from the structured elements and their properties and may therefore not always be in sync. The consolidate option synchronizes the diagram information with the information of the structured elements and their properties. To consolidate a diagram in Release 2.x./6.0, open it and choose Edit > Consolidate > All.
- Remove any parent/child relationships between application systems. There is no such thing as a parent/child relationship in the context of Oracle9i Designer.

The application interdependencies - implemented by one or more shares between application systems - can be very complex. It may not be very obvious how application systems are mutually dependent. The Quickscan Analyzer utility of the Oracle Echo tool set is able to visualise the content and mutual dependencies between application systems via an Entity Relationship Diagram. The tool depicts a repository content (or a sub-selection of application systems) into a specific Quick Scan application system, an application system into an entity, object types instances into attributes and shares into relationships. An example of a quickscan ERD is given below:



Other hints and tips based on migration best practices

Below you will find a comprehensive overview of migration best practices of structured data based on Oracle Designer migration - abstracted from Metalink between July 2000 and February 2001:

- Apply the appropriate most recent patch levels to the target and destination repositories. You can find these patches for Oracle Designer and Oracle Repository releases on Metalink.
- Apply the same database character sets in the target and destination database that host the repositories.
- Preferably start the migration as the repository owner. Although the migration is supported as a subordinate user, you need specific database access rights to successfully end the migration of one or more application systems (see also the Oracle9i SCM Repository Installation Guide). In addition there is the issue of application system ownership. The migration user will automatically become the application system owner. He or she then needs to transfer that ownership to another - administrative - user. Your user management and subsequent access rights can become extremely labor intensive if you introduce multiple application system owners.
- The migration should not be started from multiple workstations simultaneously. Once you start the migration wizard on one workstation do not attempt to start another migration session on another workstation until the first session is successfully completed.
- Verify that you have the same tnsnames.ora running on your source and destination database node. Your tnsnames.ora on your destination database node that hosts your Oracle9i Designer repository should be equal to your tnsnames.ora on your Oracle9i Designer workstation. The wizard itself will create a database link from the source Oracle Designer 6.0 repository to the target Oracle9i Designer repository. Note that the process that creates this database link is executed on the database server and not on your workstation - unless your source and destination database is running locally.
- No manipulation of data should take place during the migration in the target and source repositories.
- If you are still using Designer 2.1.x, migrate your repository first to Oracle Designer 6.0 and apply the latest Oracle Designer 6.0 patch level (Patch 7 was available in February 2002). There are several issues with migrating from a Designer 2.1.x repository that were solved once moved forward to Designer 6.0.
- Pilot a large volume set of application systems first. If you are planning to migrate a large volume of application systems in an existing populated repository you should pilot this type of migration in a shadow environment and examine the logging of the migration carefully if it does not complete successfully. As an alternative to a shadow environment you could also choose to restore a pre-migration database backup if the migration fails.
- If you migrate user-defined report data from an Oracle Designer 6.0 repository, note that user-defined reports, groups and parameters can be accessed only by a username that is identical to that of the user who created them.

Note that specific migration best practices for database elements, Oracle Developer components (e.g. Forms, Reports, Libraries and Menus) and WEB PL/SQL components are handled in parts 3, 4 and 5 of this migration guide respectively.

Problems during migration

End of file on communication channel

Occasionally, the database connection is dropped during migration from a source repository on Oracle 8.0.4 or 8.0.5. The Oracle Error Number is 3113 and the message is 'End of file on communication channel'. This appears

to be RDBMS related. The following suggestions may help to overcome the problem:

(a) Change the INIT.ORA parameter file for the source and target repository servers to increase the number of open cursors (OPEN_CURSORS) to 3000

(b) Upgrade the source repository to Oracle server 8.0.5.2.1.

Should you still experience this problem, please contact Oracle Support.

Unable to allocate an extent of %s blocks from tablespace %s

During migration from 2.x or 6.0 to 9i the following error may be reported:

> ORA-3232 unable to allocate an extent of %s blocks from tablespace %s

This is because a join of two tables required a sort operation that requested a temporary table segment. The size of the segment requested was larger than the next extent of the temporary tablespace from which it was requested.

There are two possible solutions. The next extent of the temporary tablespace cited in the error can be increased, or the request size can be reduced to fit within the next extent.

The preferred solution is to reduce the request size. To do this, examine the database INIT.ORA file (or query SYS.V$PARAMETER) to determine your database block size. Usually this will be a multiple of 2K bytes, such as 4096 or 8192. Then issue the SQL command:

```
select next_extent/db_block_size from user_tablespaces where tablespace_name =
'tablespace_name';
```

where db_block_size is the appropriate value and tablespace_name is the name given in the original error message.

Take the value calculated and set the INIT.ORA parameter hash_multiblock_io_count to an equal or lesser value. Then restart the database and restart the migration.

Migration steps from an Oracle Designer 1.3.2 repository to Oracle Designer 6.0 repository

You may skip this stage if you have already migrated to Oracle Designer 6.0.

The migration method for structured data is supported only if starting from an Oracle Designer 6.0 repository. You have to apply the following migration steps if your current release of Oracle Designer is 1.3.2.

Load the user extensibility (if applicable)

Migrate via the Repository Administration Utility Upgrade tool or migrate via application system extracts.

Unload Oracle Designer 1.3.2 user extensibility (optional)

Obviously you can skip this activity if you have not defined any user extensibility or if you do not want to bring forward the user extensibility into the Oracle9i Designer repository.

You first have to unload the Oracle Designer 1.3.2 user extensibility in order to keep your user extensibility. The unload option is available from the Oracle Designer 1.3.2 Repository Administration Utility.

Load Oracle Designer 1.3.2 user extensibility into Oracle Designer 6.0 (optional)

If you want to bring forward the user extensibility in the Oracle9i Designer repository you have to load the user extensibility in a fresh Oracle Designer 6.0 repository before loading any application system. You have to load the Oracle Designer 1.3.2 user extensibility into an Oracle Designer 6.0 repository only if you apply the migration strategy via extracted application systems.

Migration via the Repository Administration Utility Upgrade tool

If you have chosen the "migrate entire repository content" migration strategy (see above), install the Oracle Designer 6.0 client software and launch the Upgrade utility from the Oracle Designer 6.0 Repository Administration Utility to bring your entire repository content forward to Oracle Designer 6.0.

Migration via extracted application systems

Use the application restore option (available in the Repository Object Navigator) if you have chosen the sub-selection migration strategy (see above).

Note that you have to select a coherent set of application systems in order to circumvent skeleton application systems. Note also that it requires additional migration effort if any changes are applied to the selected application systems in the Oracle Designer 1.3.2 repository after the migration has taken place. You may use the freeze application system option (as a rather rigid method) to prohibit any changes in the Oracle Designer 1.3.2 repository just after the migration.

You should verify the migration logging in the repadm60\log directory before moving on to the Oracle9i Designer migration.

Migration steps from an Oracle Designer 6.0 repository to Oracle9i Designer repository

Apply the following migration steps if your current release of Oracle Designer is 6.0:

- Load the User Extensibility
- Launch the migration wizard
- Check in structured elements

Unload Oracle Designer 6.0 user extensibility (optional)

Obviously you can skip this activity if you have not defined any user extensibility or if you do not want to bring forward the user extensibility into the Oracle9i Designer repository

In order to keep your user extensibility you first have to unload the Oracle Designer 6.0 user extensibility. The unload option is available from the Oracle Designer 6.0 RAU utility.

Load Oracle Designer 6.0 user extensibility into Oracle9i Designer (optional)

If you want to bring forward the user extensibility into the Oracle9i Designer repository you first have to load the user extensibility in a fresh Oracle9i Designer repository before migrating any application system. You always have to load the Oracle Designer 6.0 user extensibility into a fresh Oracle9i Designer repository for both migration strategies.

Launch the migration wizard

Read the migration section of the Oracle9i SCM Repository Installation Guide before launching the migration wizard.

The Oracle9i Designer migration method for structured elements has changed significantly. An Oracle Designer 6.0 repository will not be upgraded to an Oracle9i Designer repository during the migration. The Oracle Designer 6.0 repository content (or part of it) will be copied into a fresh installed (or already populated) Oracle9i Designer repository.

It should be noted that the migration wizard will actually clean up your Oracle Designer 6.0 repository. The migration wizard will for example correct invalid references, populate missing properties, update invalid properties or force delete duplicate values. This cleanup is optimized for Oracle9i Designer. You should therefore not reuse your Designer 6.0 repository for these application systems that are migrated to Oracle9i Designer. You should use a target shadow Oracle Designer 6.0 repository if you are still planning to use Oracle Designer 6.0.

The migration wizard allows you to connect to an Oracle Designer 6.0 repository and subsequently select one or more application systems. Apply the specific migration strategy as discussed before.

The wizard will verify if there are any incoming shares from other application systems that were not selected. These application systems will be added to the selection in order to circumvent skeleton application systems. A skeleton application system only captures the objects that are shared to one or more other application systems. It does not contain all objects. The selection of a coherent set of application systems is extremely important. Note that coherency is not only about receiving shares but also about outgoing shares. For example if you migrate the single application QMS505 (1) first and secondly the application HSD505(1) you would receive QMS505(1) again because of the incoming shares of QMS505 to HSD505. See the example of a quickscan ERD given in the section "Preparing (cleaning up) your 'old' repository content", above. You should however not re-select an application system that has already been (implicitly) migrated. In this example you should migrate HSD505 and QMS505 as a set.

The wizard will populate a workarea called GLOBAL_SHARED_WORKAREA with the migrated data if your repository is not version enabled.

The wizard will populate the migrated data in a new system generated workarea if your repository is (already) version enabled. The name of that workarea will be based upon the connect string (e.g. WA_des60_ds6). You should verify the Workarea rule of this system generated workarea. You should change this rule to your own specifications if it contains the following rule INCLUDE_FOLDER(SYSTEM FOLDER{MAIN;LATEST}) and

if you are planning to check in the structured elements. This specific rule will evaluate the content of the system folder only after you have checked in your structured elements. It will not display your structured elements. You will find more information about workarea and workarea rules in a paper by Lucas Jellema, called "Interior Oracle9i Designer" published for the ODTUG2000 and in Chapter 6 "Reorganize a migrated Oracle9i Designer repository". Note that you cannot reuse previously created workareas.

You can control, as in Oracle Designer 2.1.x and Oracle Designer 6.0, to a certain extent, the migration operation while it is in progress. The migration operation is divided into a number of stages. You can pause and restart the current stage, retry or skip a stage that failed, or abandon the whole operation.

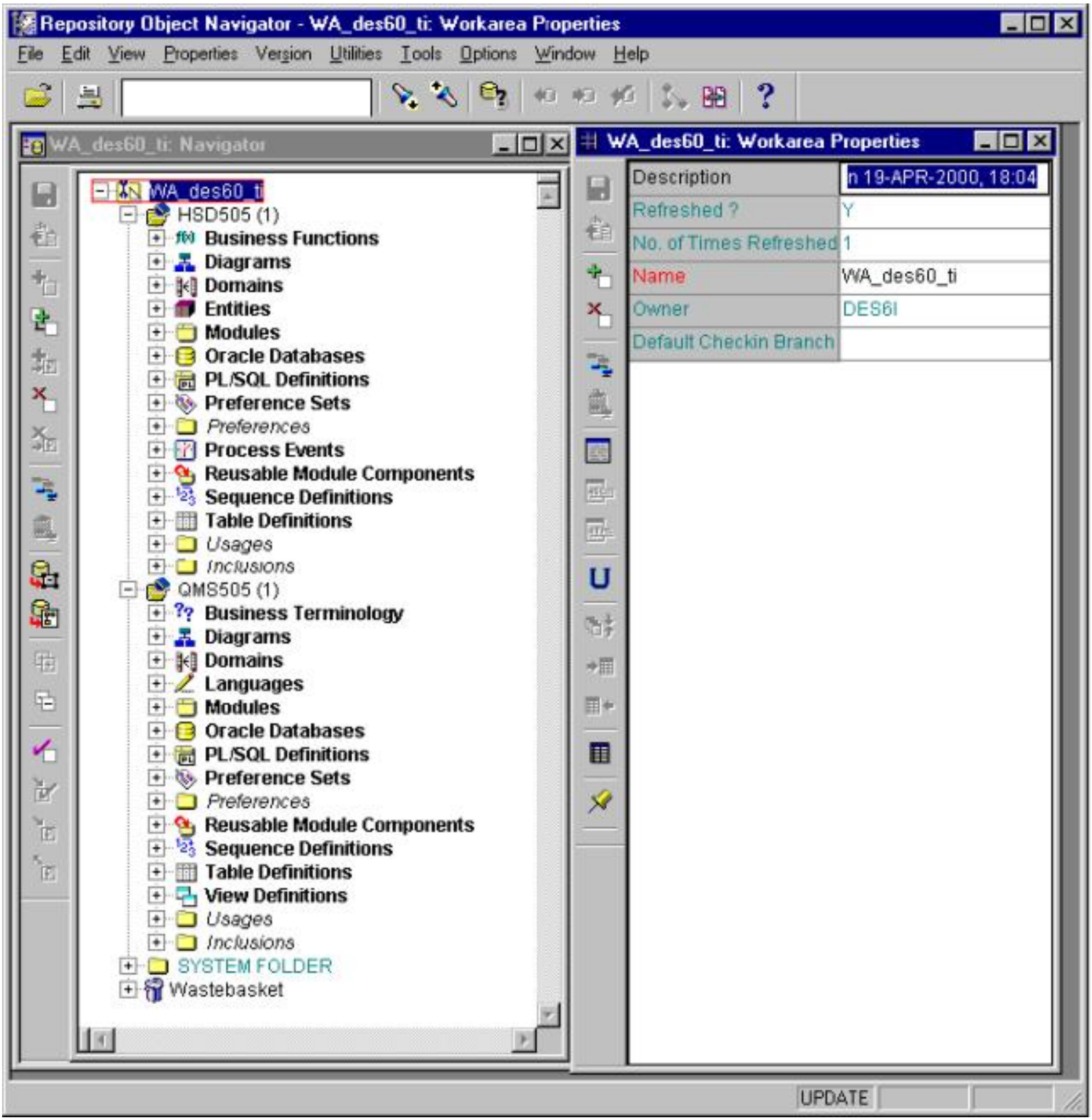
The Control Status dialog box is displayed while the upgrade is in progress, and includes a number of buttons for the different controls.

For example, if you ran out of extents in the rollback segment during an upgrade, you could pause the upgrade at its current stage, allocate a larger rollback segment, then restart the upgrade from the stage where the failure occurred.

It is highly recommended that you verify the content of the migration log files, written to the ORACLE_HOME\repadm61\logs directory, during or after the migration.

Application systems are migrated as application systems suffixed with the application version number (e.g. "HSD505 (1) or QMS505 (1)") and not as (root) folders. There is no option (yet) to transfer application systems as (root) folders (or vice versa).

The Oracle9i Designer repository content may look as follows just after the migration of your structured data:



Check in structured elements

If you do not want to enter the versioned world (yet) you may skip this activity.

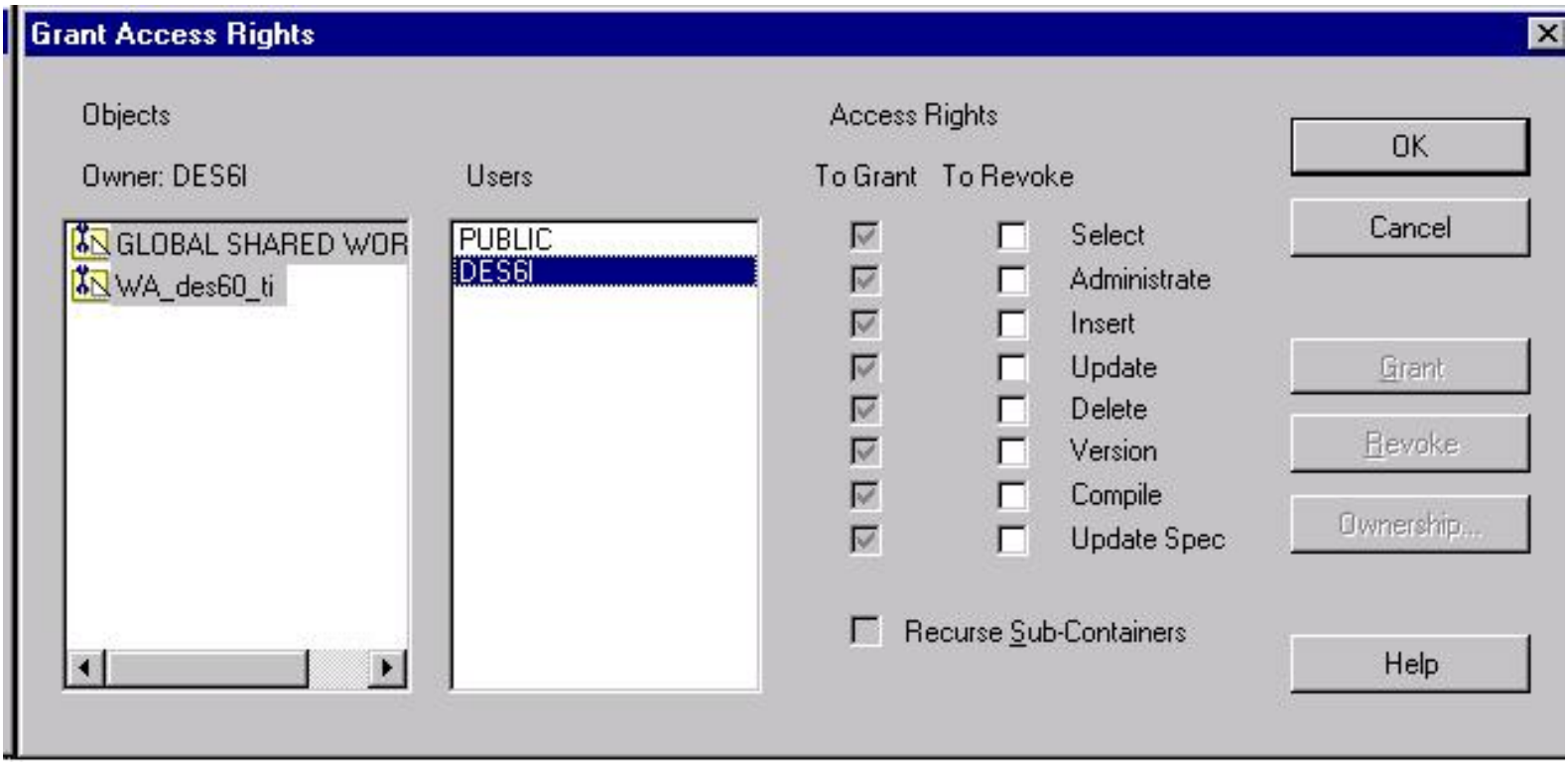
The migration wizard groups all migrated application systems initially either in a default workarea called GLOBAL_SHARED_WORKAREA or in a specific migration workarea (e.g. WA_DES6I_DES6I) if you have a version enabled repository.

Execute the following steps if you do want to enter the versioned world.

- Verify the access rights of the repository owner against the workareas and the underlying containers.
- Check in all structured objects via the "List Checkouts" option

Verify the access rights

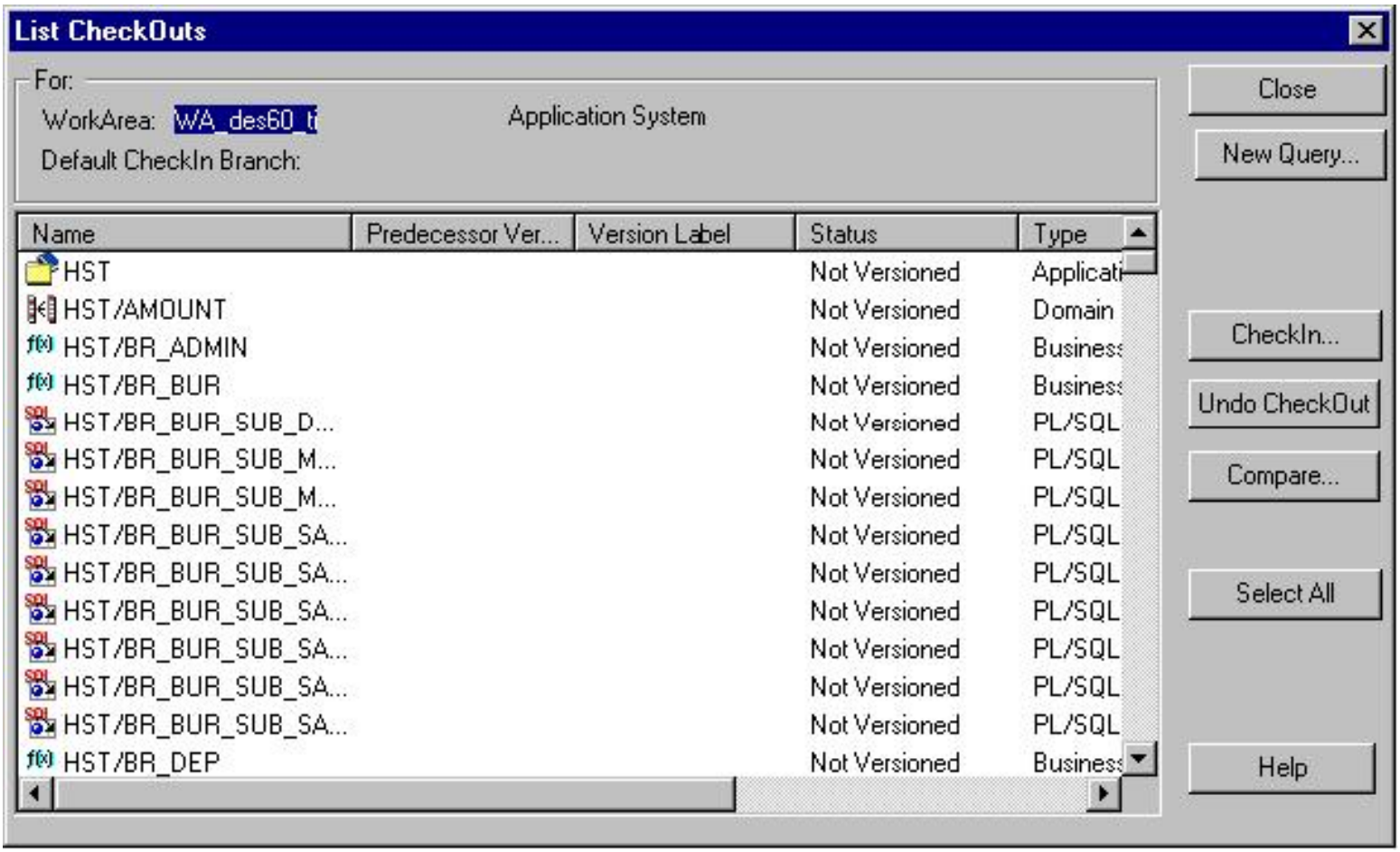
Verify the access rights against the (default) workareas and migrated application systems of the Oracle9i Designer repository owner. The Oracle9i Designer repository owner must have the version privilege for both objects in order to check in the objects within the context of workareas. The following shows access rights against workareas and application systems:



Note that you can select multiple workareas and application systems at the same time - via a discontinuous select - to review and define the access rights for the specific user.

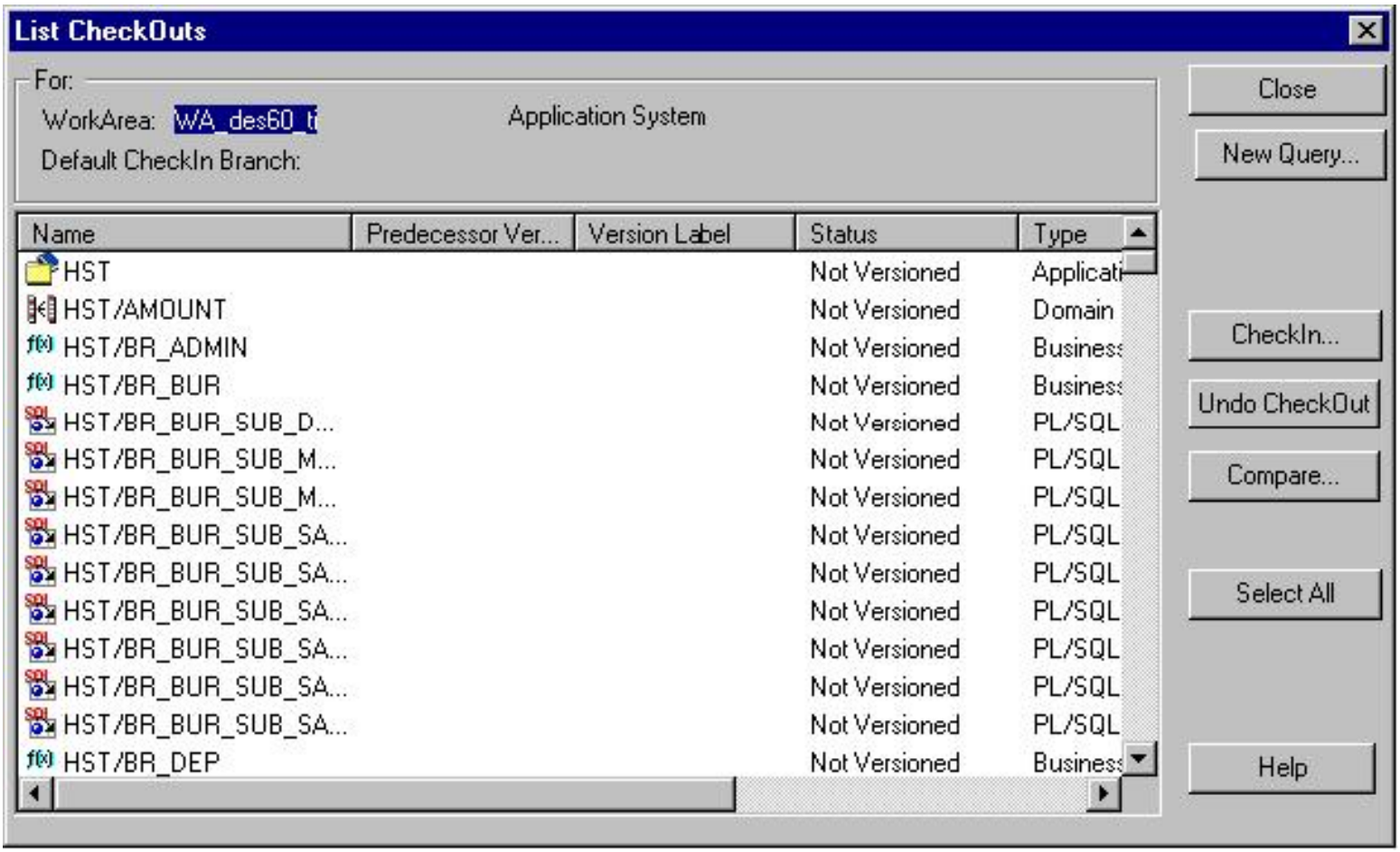
Check in structured objects

The migration wizard has loaded your structured elements in the repository, but it did not check in the objects. The Oracle9i Designer Repository Object Navigator has an option to check in *all* non-versioned objects simultaneously. This "List Checkouts…" option is available for each container via the right mouse button and will launch the List CheckOuts Criteria dialog:



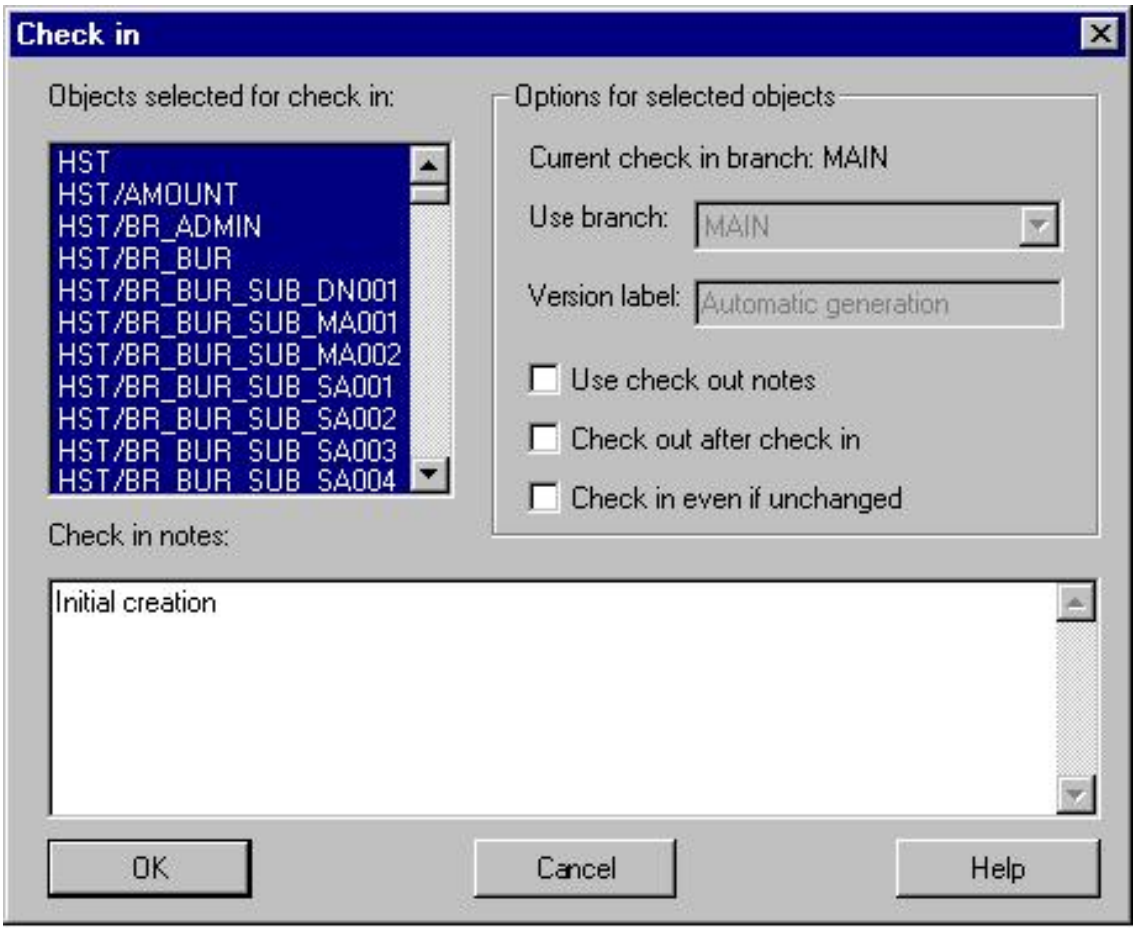
In the context of migration you are only interested in a list of "non-versioned" objects.

Clicking the OK button will launch a window with all non-versioned objects for the highlighted container:



You can check in only owned non-versioned objects. You cannot check in non-owned short-cuts. These objects should not be highlighted in the list, while checking in all non-versioned objects.

It is very useful to apply the same checkin notes (e.g. "Initial creation") for all non-versioned objects in the checkin windows, as illustrated below:



You should realize that once you have checked in your first object(s) you actually have entered the versioned world. From that moment on you cannot re-enter the non-versioned world - other than by recreating a fresh Oracle9i Designer repository and re-starting the migration in a non-versioned mode. The checkin operation is an irreversible action.

Checkingin of the subordinate non-versioned objects via the "List Checkouts" option does automatically imply a checkin of the application system. Most likely you would like to apply your naming standard for application systems or containers in general. For example, you could change the default migration application name "HSD505 (1)" into HST and "QMS505 (1)" into QMS just before the batch checkin of all subordinate objects. Note also that you have to check out an object to apply any changes - including the name of the application system.

Post migration steps for structured data

This stage covers activities for structured data that are not handled by the migration wizard:

- Remove the suffix “(1)” from your migrated application system name.
- Translate (old) parent application systems to a nested container structure.
- Build a version tree for structured data.
- Verify your migration result.
- Reevaluate preference sets.

Remove the suffix “(1)” from your migrated application system name

Remove the suffix “(1)” or any other version indication suffix from your migrated application system name. The command line tool may not cope with suffixes appropriately.

Translate (old) parent application systems to a nested container structure

Skip this activity if you have not defined a hierarchy of application systems (also known as parent application systems) in previous releases of Oracle Designer.

You should realize that the implementation of application systems hierarchies is quite different in Oracle9i Designer to the implementation of application system hierarchies in releases of Oracle Designer before 6i. Oracle9i Designer allows you to define a hierarchy of containers: application systems may contain other application systems or folders. Folders may contain other folders or application systems. All containers - no matter their position in the hierarchy - could contain subordinate objects (e.g. entities, tables) in Oracle9i Designer as opposed to previous releases in Oracle Designer. In previous releases of Oracle Designer only subordinate application systems could contain subordinate objects.

Reconsider therefore the application system hierarchy before and after the migration. Note that you should have removed the parent/child relationship between application systems prior to migration.

Building a version tree for structured data

Skip this activity if you do not want to enter (yet) the versioned world or if you do not (yet) want to have multiple object versions.

The only way to store multiple object versions in releases of Oracle Designer before 6i was by the creation of an additional application system via archive/restore, or by versioning an application system. Within these additional application systems you could store the same object (name) of the same type with a different definition. These different object definitions represented another object version.

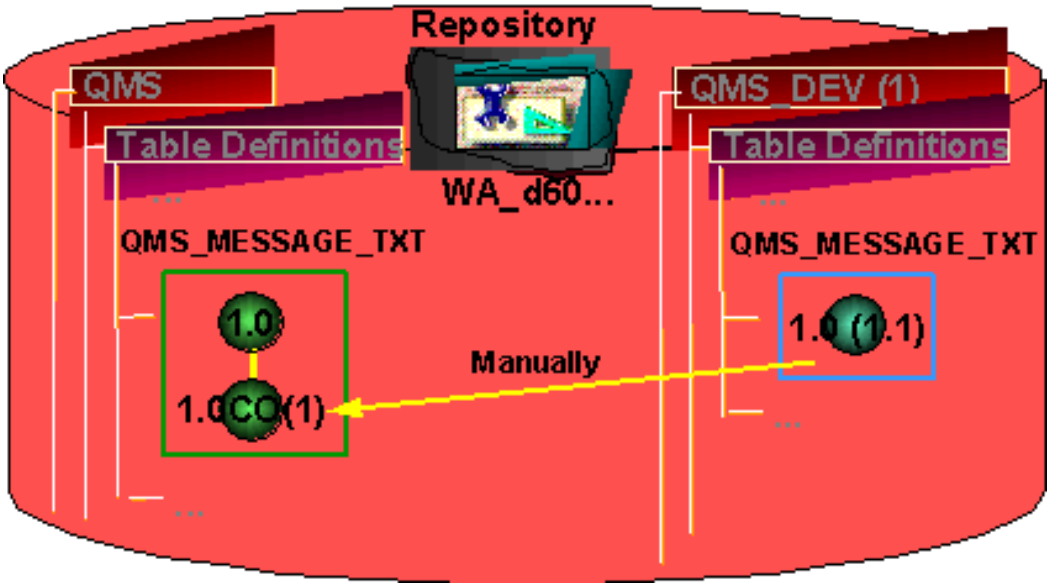
The migration wizard will not migrate these different object versions as a version tree. The migration wizard will handle these object versions as totally different objects.

Build a version tree manually

The manual build of a version tree of structured elements comprises the following steps (see also the illustration that follows):

1. Check out the context element (with the lowest version definition) from the default migration workarea (e.g. QMS_MESASAGE_TXT [1.0]).
2. Apply the changes manually based upon the definitions captured in QMS505_DEV (1)\ QMS_MESASAGE_TXT .
3. Compare the two versions with the compare utility and verify this result with the Oracle Designer 1.3.2 environment. You can launch the compare prevision version utility directly via the checked out object.
4. Check in the manually changed version of the object (e.g. QMS_MESASAGE_TXT [1.1]).

Repeat steps 1 to 4 until no more additional versions need to be built manually.



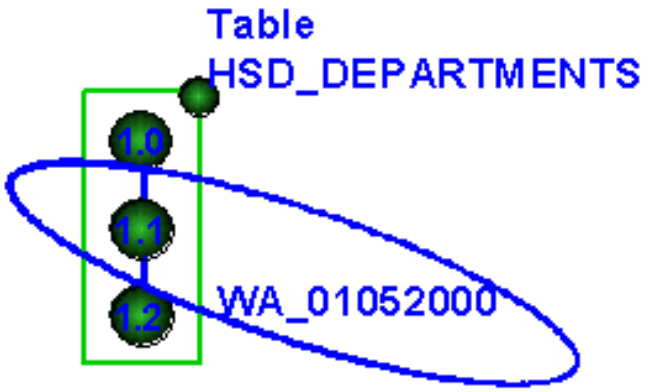
Post migration steps for structured elements within a staged migration approach

The next section will discus the merge steps for structured elements if you have adopted a staged migration approach (as discussed in the section "Migrate a sub-selection of application system", above).

Merge interim changes in structured elements

Apply the following steps for structured elements to bring the interim changes forward in Oracle9i Designer.

1. Create a workarea called WA_<ddmmyyyy> with rule LATEST_BEFORE that represents a snapshot of your repository just after the initial checkin of your structured elements (and files). Note that this workarea also represents the content of your "original" repository before any interim changes have been applied. You should not allow anyone to make any changes via this workarea.
2. Isolate the changed structured elements in a User Defined Set in your non-9i repository.
3. Isolate the changed structured elements in a User Defined Set in your Oracle9i Designer repository.
4. Migrate all application systems which contain a specific User Defined Set to Oracle9i Designer via the migration wizard.
5. Check out (if possible) all objects that are associated with the specific User Defined Set in Oracle9i Designer.
6. If the object is already checked out then you have to merge the interim changes with the changes applied in Oracle9i Designer.
7. If the checked in object version is higher than the object version in WA_<ddmmyyyy> you also have to merge the interim changes. In the following illustration, you have to merge the interim changes since the interim changes are based upon version 1.1.



8. If the checked in object version equals the object version from WA_<ddmmyyyy> then you can bring all interim changes forward in Oracle9i Designer.

Step 8 may be supported by the export/import method as described earlier in section Build a version tree for structured elements via the export/import utility.

Step 6 or 7 (if applicable) must always take place manually. You can however effectively use the compare utility between the latest version of the object and the similar object in the migrated application.

Verify the migrated structured data

Apply the following checklist for your migrated structured data:

- All selected application systems in your Oracle Designer 6.0 repository should also be available in the Oracle9i Designer repository.
- All shares should be translated as "short cuts".
- Your version tree - if applicable - should be comparable with your "object versions" captured in diferent application systems in previous releases of Oracle Designer.

Compute statistics

To avoid a degradation in repository performance, we recommend that you run the Compute Statistics utility (available at the Oracle9i Designer RAU tab sheet) regularly, and especially after an operation that significantly affects the size of the repository.

Note that the migration automatically launches a compute statistics stage. Therefore you do not have to run the compute statistics directly after a migration.

In Oracle9*i* Designer, you can run the utility directly from the Repository Administration Utility window by clicking the Compute Statistics button:

In most cases, we recommend using a higher figure than the default (e.g. 50%). However, if your repository has large tables, it is especially important to use lower figures for the sample size, as higher figures will cause the compute operation to take a long time to complete.

In addition it is highly recommended to implement a daily batch mechanism that computes the statistics of the dynamic repository tables.

Reevaluate preference sets

The handling of the preference sets reevaluation on application system level or lower for your Oracle Developer and WEB PL/SQL components takes place in Part 4 and Part 5 of this guide respectively.

Chapter 4 Migrate files from third party source control tools

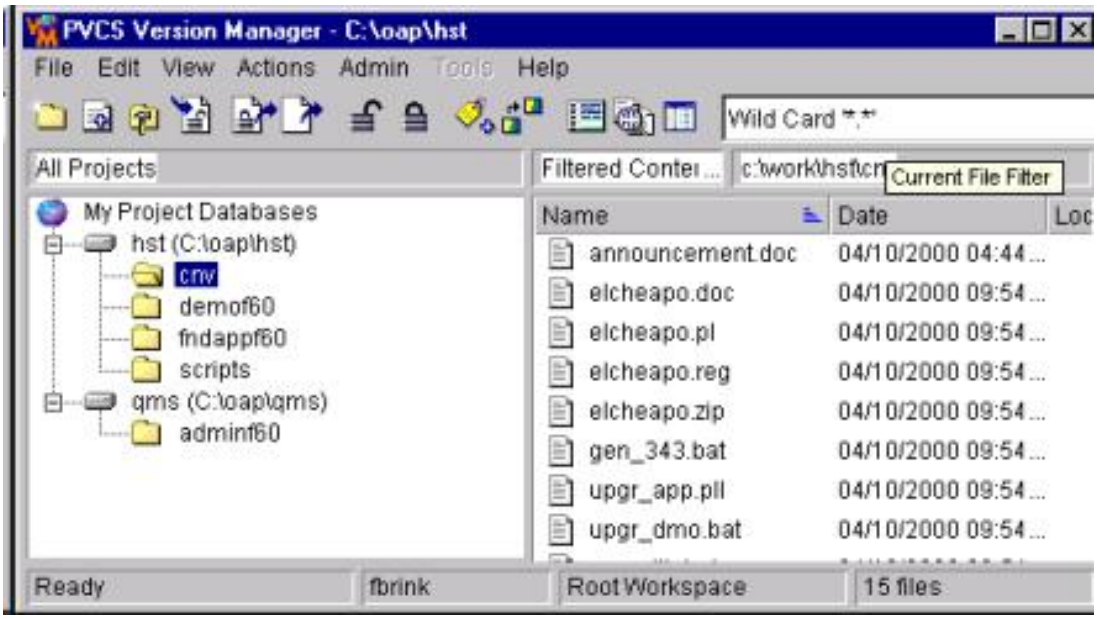
The migration of files, stored in a third party source control tool, should typically start after the migration of the structured elements. Only then are you able to integrate your files with the structured elements - both type of elements will then share the same root-container.

The migration of files into Oracle9i Designer comprises the following steps:

- Populate the file system with a baseline set of files
- Populate the Oracle9i Designer repository

Skip this chapter if you have not stored your files in a third party source control tool. (e.g. PVCS/VM, ClearCase, Visual Source Safe).

The migration of files stored in a third party source control tool will be clarified using an abstract of 'Headstart 5.0.5' files. It is assumed that the files are captured in two project databases in PVCS/VM : HST and QMS respectively (as illustrated below).



PVCS/VM is a source control tool like ClearCase or VisualSource that supports the storage of multiple file versions, workspace procedures and several other configuration management procedures.

Populate the file system with a baseline set of files

The migration of files will be based upon the file upload functionality in Oracle9i Designer. The migration strategy of files will not be based upon an interpretation of the third party source control repository.

The migration steps for files stored in a source control tool will be illustrated by using PVCS/VM as an example. Note that you could apply the same migration strategy for files stored in other source control tools (e.g. ClearCase, Visual Source Safe). Most likely these other tools have similar options available to publish your repository content - via a specific filter - on the file system.

The publication of a baseline file set should typically be organised in the following stages:

- Get an overview of your third party source control tool repository.
- Clean up your third party source control tool repository.
- Determine a generic directory structure.
- Publish the baseline set of files on the file system.

Get an overview of your third party source control tool repository

Your third party source control may have several reports or options available to visualise the repository content of file versions. Such an overview will effectively support you with the publication of a baseline set of files.

For example PVCS/VM offers an option to export the administrative information (e.g. projects, files, file versions, group association with file versions) into an Oracle database. Subsequently you can build reports (using SQL*Plus) to generate an overview of your PVCS/VM repository content that may look as follows:

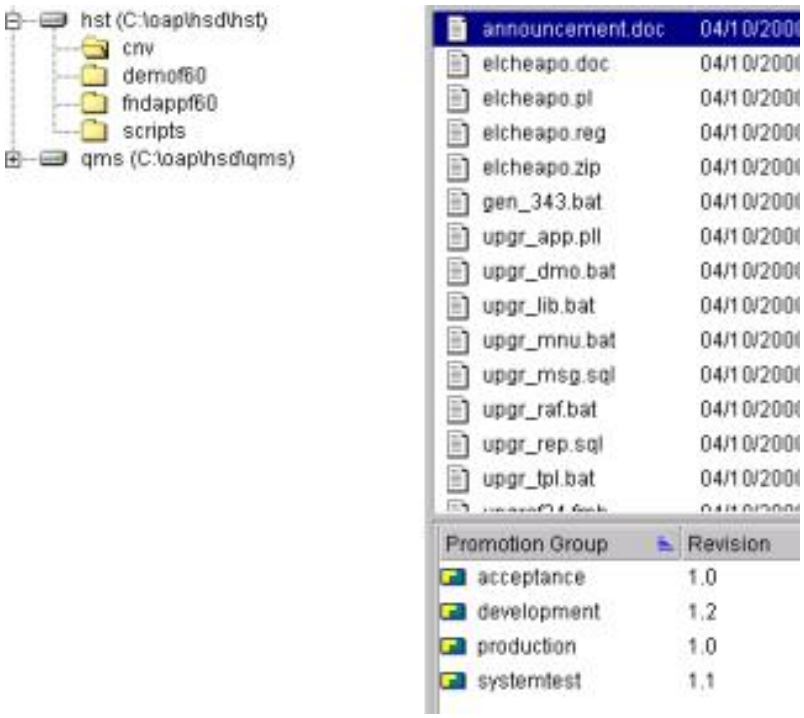
```
PRJ Filename                    Develop  Test  Accept  Prod

...
HST upg501.sql                  1.0
HST announcement.doc            1.2      1.1   1.0     1.0
HST elcheapo.pl                 1.0      1.0   1.0     1.0
...
QMS qmsolb50.olb                1.7      1.6   1.6     1.6
...
```

Note: Appendix A contains an example of a SQL query - based on the PVCS/VM administration tables - that has generated the above report.

The above report is based upon the functionality in PVCS/VM that enables you to associate one or more promotion groups - see report headers - to a specific file version. In PVCS/VM you can also associate one or more release labels to a specific file version. You could generate a similar report with release numbers in the column headers instead of promotion groups.

The PVCS/VM content for the file announcement.doc should look as follows:



Note that the exported PVCS/VM administrative information is static. It does not reflect any changes after you have exported the data from the PVCS/VM repository. You should therefore export the administrative data multiple times if there are any cleanup activities based upon the first output - and that is most likely to happen.

Clean up your third party source control tool repository

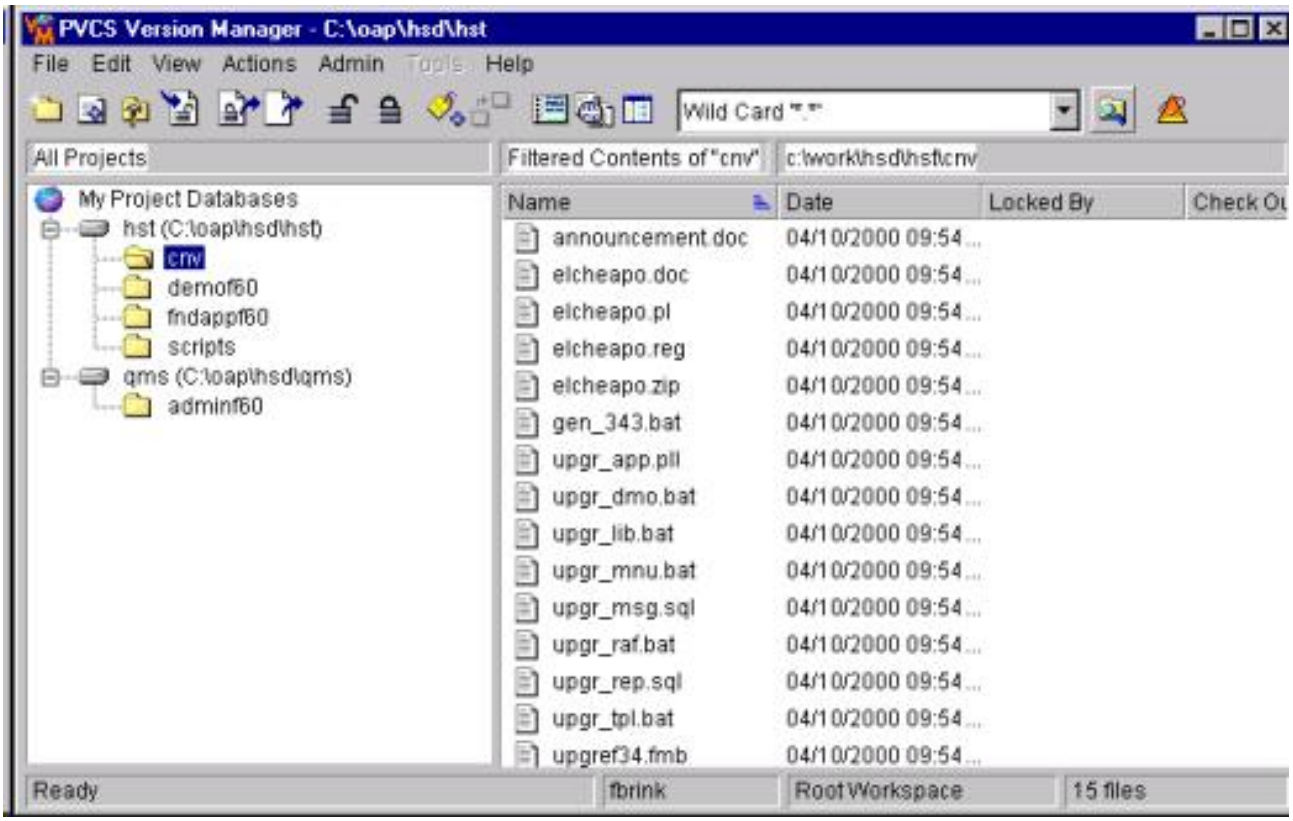
The upload is an excellent opportunity to clean up your repository content of the third party source control tool. You can apply the following clean-up checklist for files captured in third party source tools, using the overview, as discussed in the previous section:

- Remove unused containers and/or sub-containers. The equivalent of a container in PVCS/VM is called a project database and a sub-container is called a project.
- Remove obsolete files.
- Remove obsolete (interim) object versions. Candidate obsolete or interim file versions are file versions that are not used or associated with a release or a promotion level.

Determine a generic directory structure

The upload will not only capture the files, but also the owning directories. You should therefore investigate the content of the source control tool for each specific project (or any other root grouping vehicle) in terms of (sub)folders and file types. Subsequently you can determine a generic directory structure based upon this investigation.

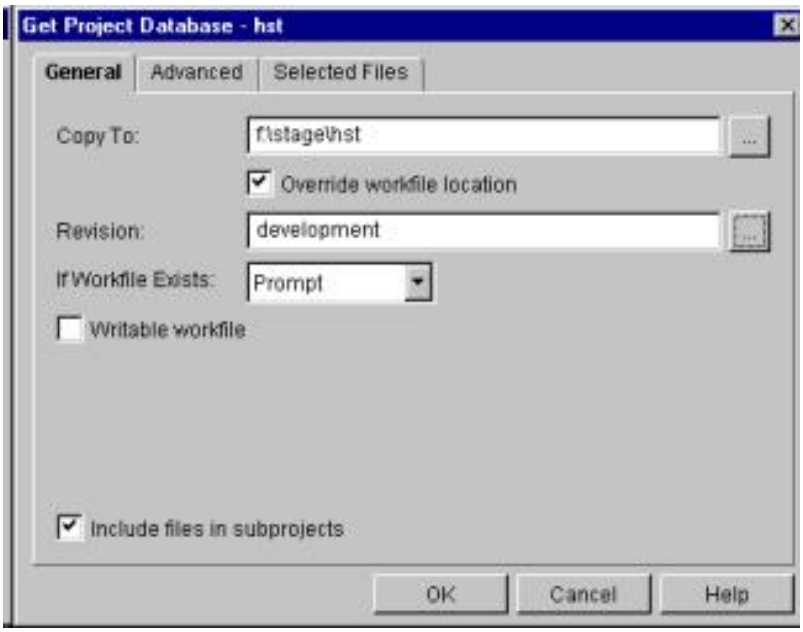
Example: Your files in a specific project database could already been organized in a specific (nested) folder structure (see the illustration below) or you could group your files in one or more sub-folders based upon the file extension if your project database is not organized in sub-folders.

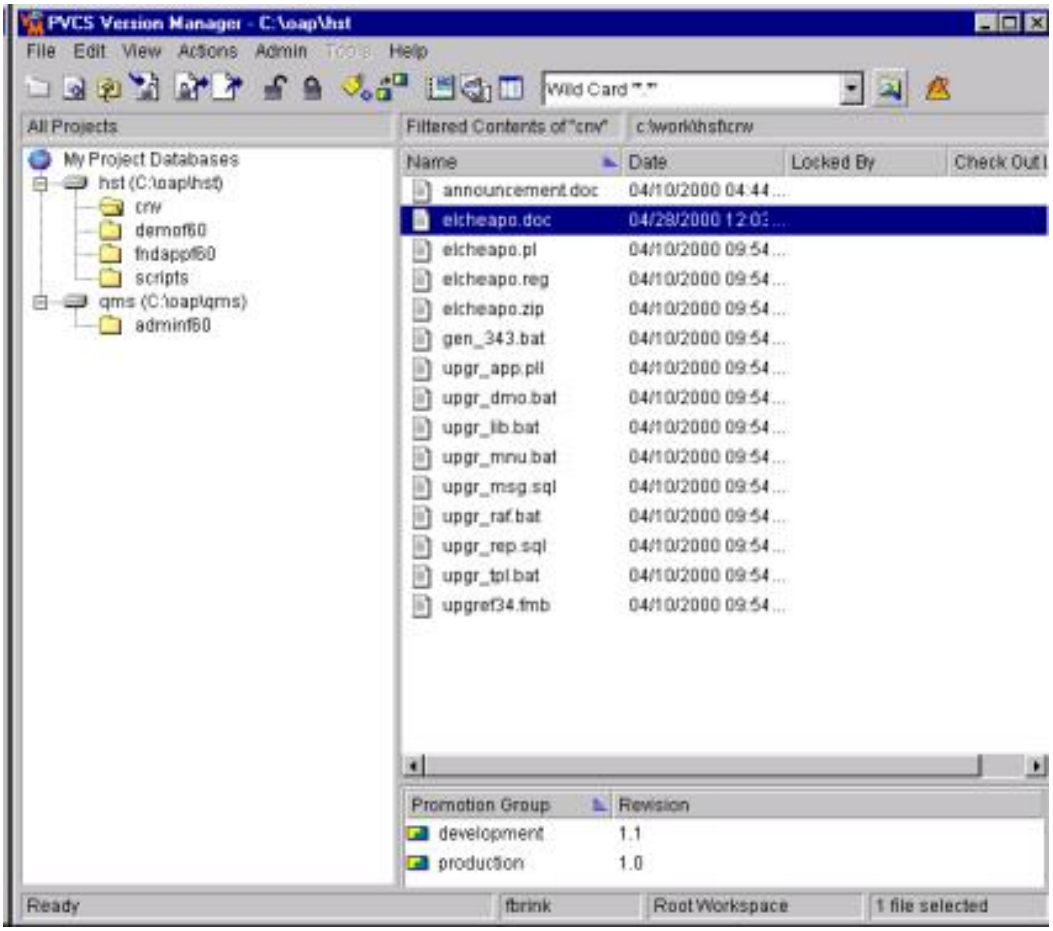


The illustration above shows that a directory structure for HST and QMS is already available. If such a structure is not yet available you could set up a directory structure based upon your file system structure HST and QMS respectively.

Publish the baseline set of files on the file system

Your third party source control tool probably supports the functionality to publish a set of files of a specific version based upon the promotion group association or a release label association. Specifically, this "get" option is very effective for the migration of files. You can use this option to publish a baseline set of your files to a specific destination directory structure. For example you could publish all your file versions that are associated with the group 'development' (as shown in the dialog below) to the stage area 'f:\stage\hst\' or you could start to publish all your file versions that are associated with version label 5.0.5. The "Get" option in the screen dump below will publish all file versions and its owning folders and subfolders that are associated with the promotion group "development" to directory f:\stage\hst .

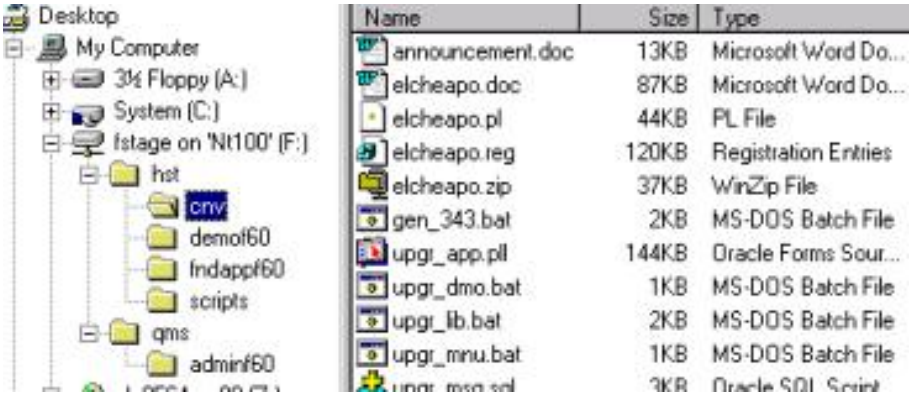




The window above depicts the promotion group association in PVCS/VM against file versions. The 1.1 version of the elcheapo.doc file is associated with the promotion group "Development" while the 1.0 version of is associated with "Production".

Note that if you want to bring forward multiple file versions (see the section "Building a version tree for files"), you have to start with a baseline or lowest version in the publish step. Your baseline version will probably be captured in the production environment. If you do not want to bring forward multiple file versions you have to "get" and upload the latest file versions. These latest versions are probably associated with the "development" promotion group.

Your (populated) file system may look as follows after the (first) publish operation for project HST for the sub-folder scripts.



Note that you may have to move your files after the publish operation to a specific sub-directory if you did not organize your sources in a specific sub-folder structure beforehand. The move operation should typically be based upon the output of your investigation to a generic directory (see also the section "Determine a generic directory structure").

Your third party source control tool may also have publish options that are accessible from the command line interface or via an API. For example PVCS/VM offers a GET option via a Command Line Interface in a 'DOS' session that allows you to get a multiple set of file versions either based upon a release label or a promotion group. Therefore you may prefer the Command Line Interface instead of the GUI interface to execute the publish operation as a batch job.

Populate the Oracle9i Designer repository

In this activity you will populate the Oracle9i Designer repository with the (first) set of files. The populate operation is divided into two steps:

- Apply (root) folder mapping
- Upload the files - and check in if applicable

Every insert, update or delete operation on the Oracle9i Designer repository must be executed within the context of a specific workarea.

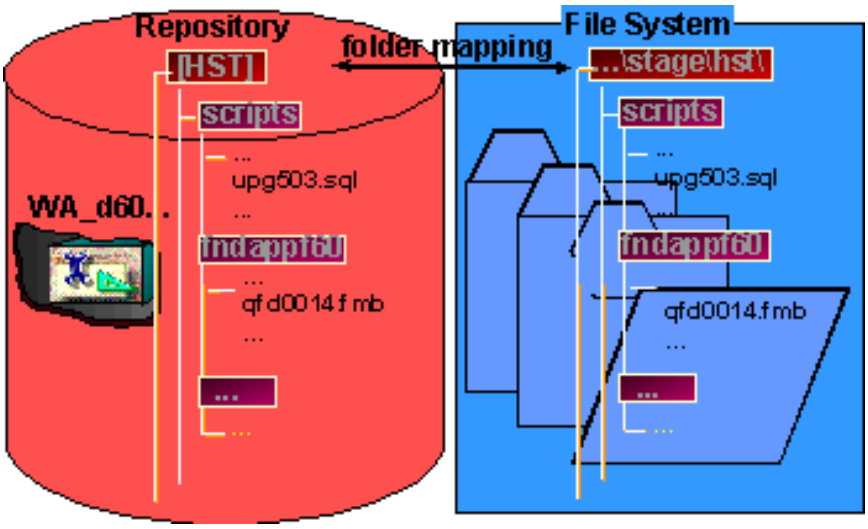
If you are not planning to enable the repository for versioning you have to use the default migration workarea GLOBAL_SHARED_WORKAREA.

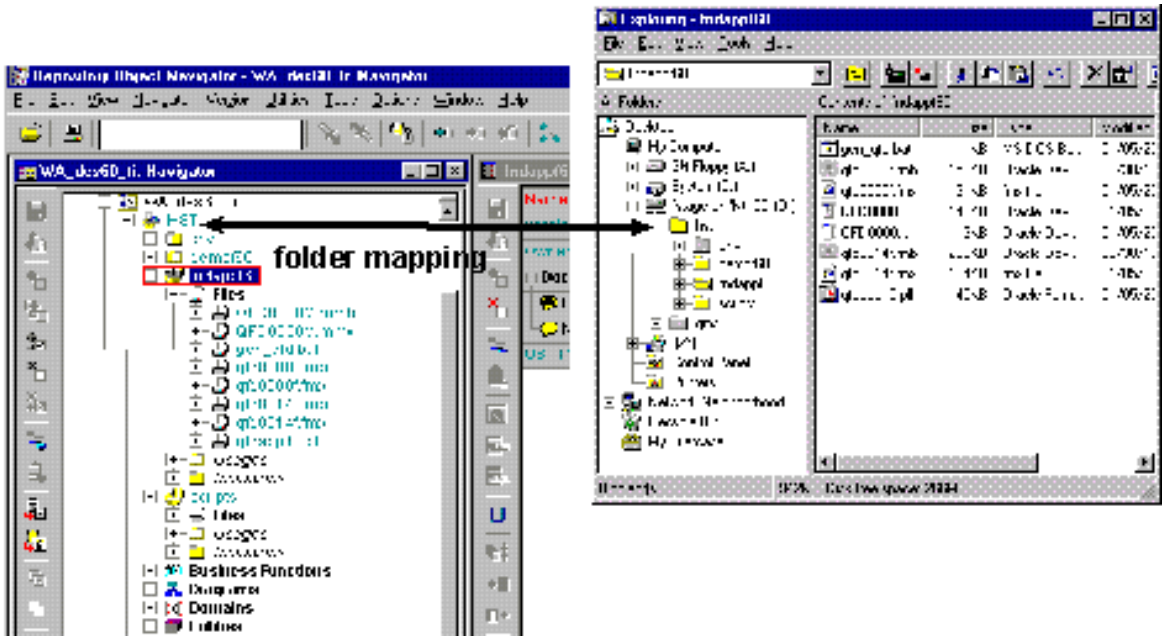
Otherwise you can use any workarea for the migration of files since you have the intention to check in the file objects. Note that every checked in object can be accessed eventually in every workarea (depending on the workarea rules).

Apply (root) folder mapping

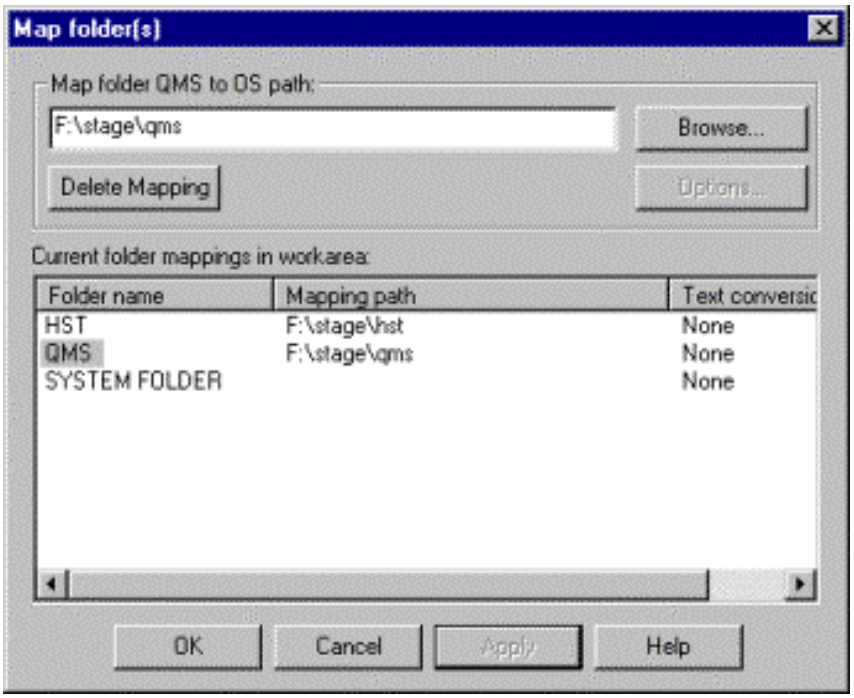
The upload (and check-in of files if applicable) must be preceded by a folder mapping. The "Map Folder to the File System" option is available for each root container - at the right mouse button for a specific root container.

This folder mapping option allows you to upload the file system content (directory and files) with the repository. See below.





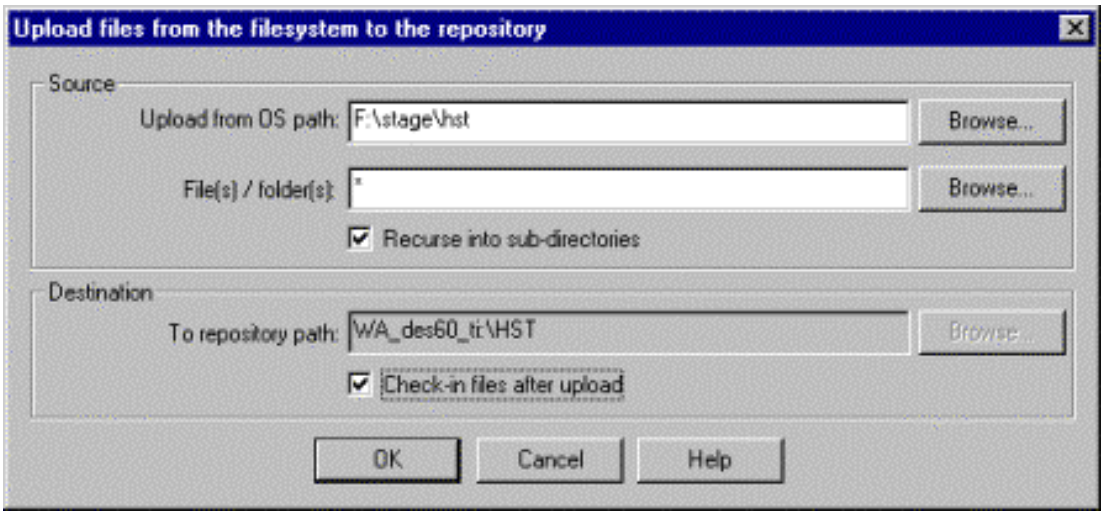
For example the directory "f:\stage\hst\" will be mapped to the HST container and the directory "f:\stage\qms\" will be mapped to the QMS container. See below.



Note that the files - and their folders and subfolders - in the Headstart example will be uploaded (and checked in if applicable) against the existing containers (e.g. HST and QMS). Obviously you could also bring the files forward in other or new "to be created" containers.

Upload of files

The upload will actually store the files in the repository. In addition it will also bring all directories and sub-directories forward if you have enabled the "Recurse into sub-directories" option (see the following dialog box). It is therefore important that you have applied a generic folder structure, since that structure will be loaded in the repository as well.



Checkin of files

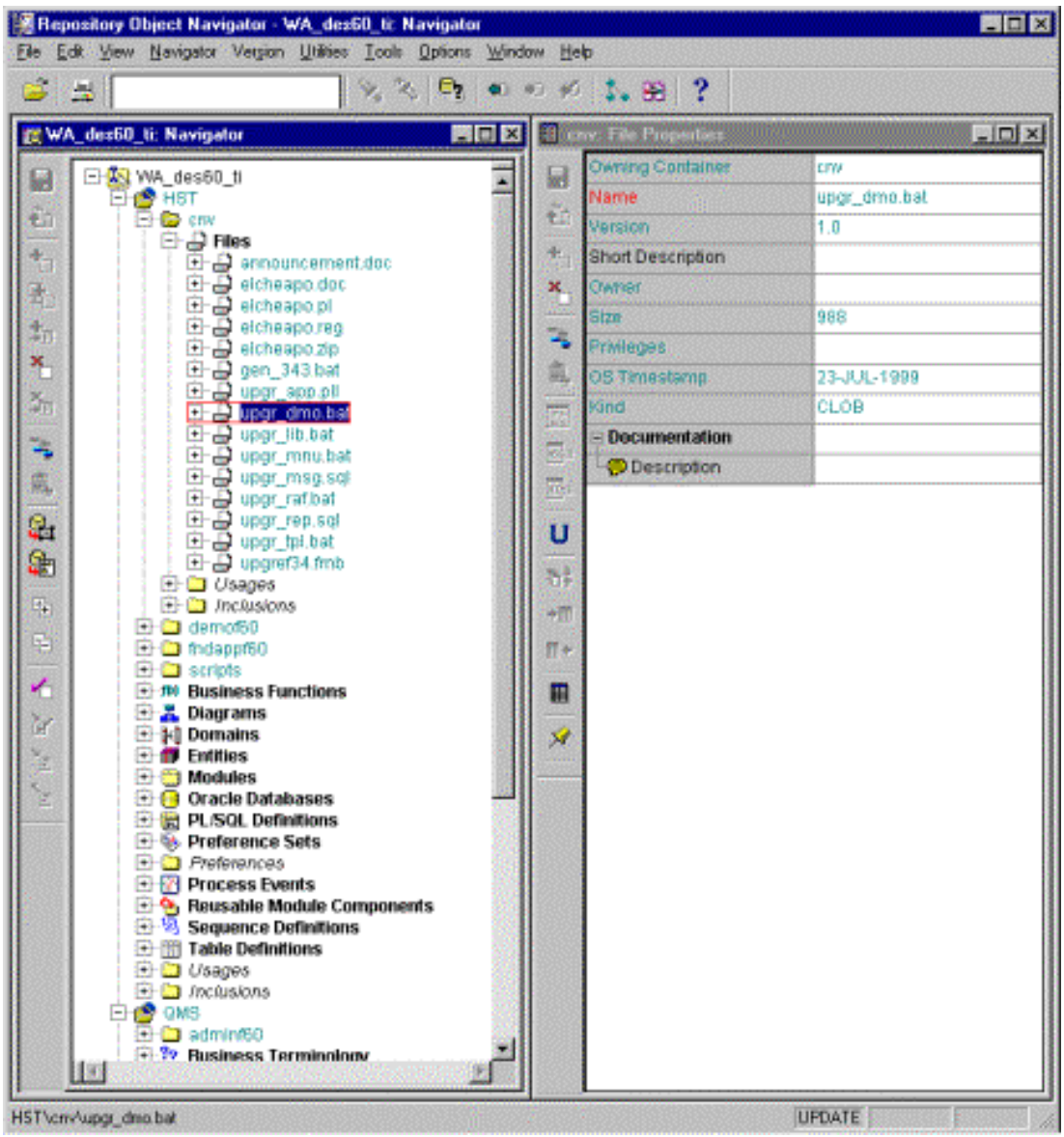
Skip this section if you do not want to enter the versioned world.

If you want to check in your files, you can either combine the upload with a check-in or you can use the "list checkouts…" option (as with the structured elements). The check-in of files will actually apply a version number to the stored files.

As with structured elements you should realise that once you have checked in your first file(s) you have actually entered the versioned world. From that moment on you cannot re-enter the non-versioned world - other than by re-creating a fresh Oracle9i Designer repository and re-starting the migration in a non-versioned mode. The check-in operation is an irreversible action.

Note that you can verify the "check-in" status of all file elements (including folders) via the "list checkouts…" option per root-container. Eventually this option should not feed back any file elements.

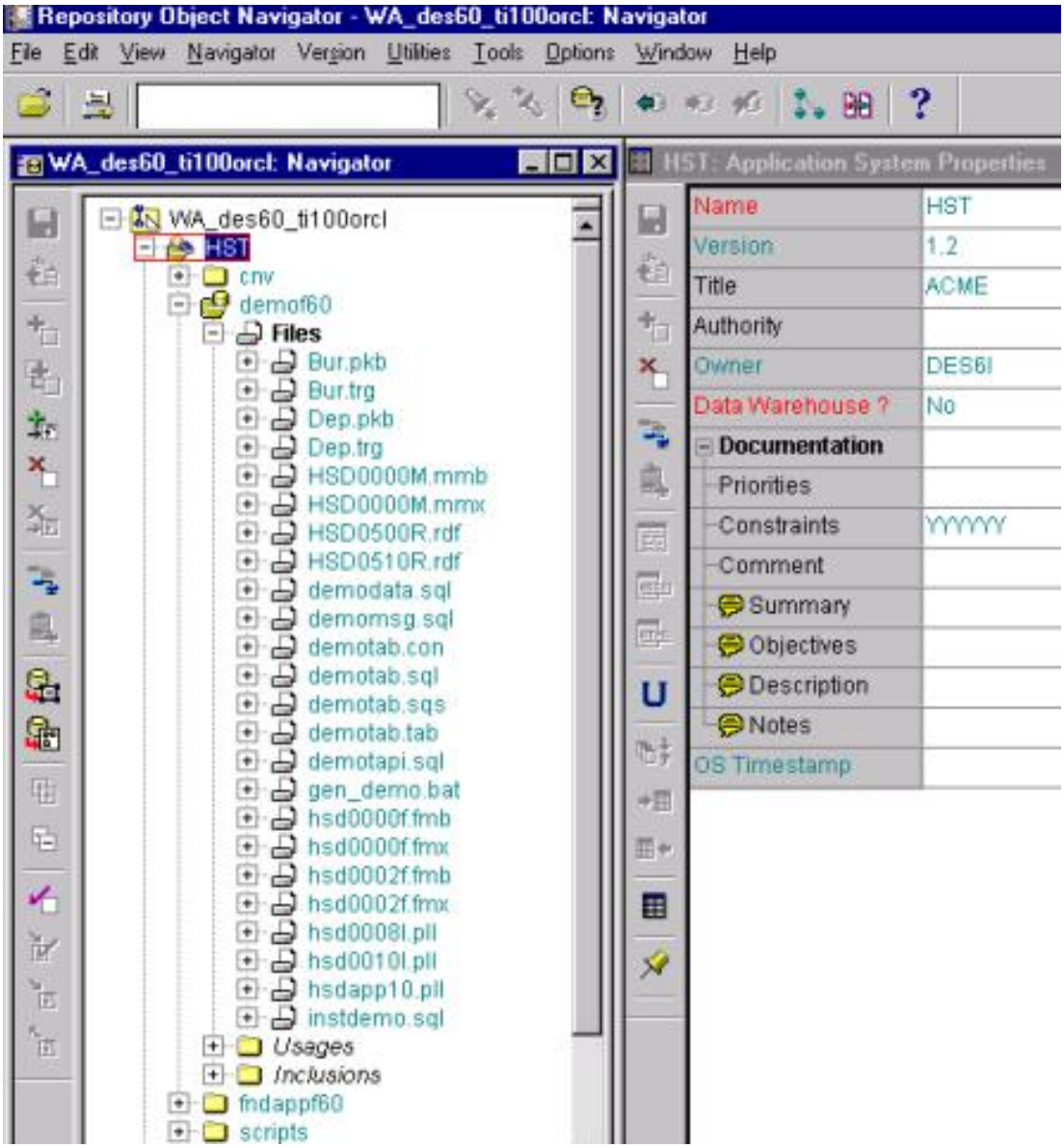
The container and sub-container structure for application system HST looks as follows after the upload and check-in operation:



Upload and check in sources and executables

Next to the storage of structured elements you may consider the upload and check-in of the derived sources and corresponding executables. For example you could also upload the derived Oracle Forms FMB source (e.g. HSD0004F.FMB) and corresponding executable (e.g. HSD0004F.FMX) next to the migrated structured form itself (Oracle Forms 60 module HSD0004F). In addition you could also upload multiple representations of your sources and executables, for example Forms50 and Forms60. You can organise your multiple representation either via different folders (e.g., demof50, demof60) or via the use of different labels.

With all representations available in the repository you could instantly build your environment (via the download option) from the repository content without the derivation (or generation) step from a structured format into a derived source and executable. You can then gradually replace the source and the executable with the new generated and compiled representation over time. Your Oracle9i Designer repository can act as a single point of control just after the migration - it captures all your application objects necessary for a 'runtime' environment, as below:



Post migration steps for files stored in a third party source control tool

There are the following (possible) post migration steps for files:

- Build a version tree for files
- Verify the migrated files
- Compute statistics

Building a version tree for files

It is quite easy to build a version tree for files stored in a third party source control tool (compared with structured elements).

Skip this activity if you do not (yet) want to enter the versioned world or if you do not (yet) want to have multiple object versions of files.

If you want to bring forward multiple file versions you have to start with a baseline or lowest version in the publish step as discussed above. Obviously the version tree should be built starting from the lowest version.

If you are planning to build a version tree, you can build it either:

- immediately, or
- in time.

The build of a version tree for files should be based upon an overview as was presented in the section "Get an overview of your third party source control tool repository", above. Such an overview of the third party source control content, i.e. which file versions are available and with which promotion groups or release labels are these file versions associated, is not only very productive, but also quality-effective for verification purposes.

Building a version tree immediately

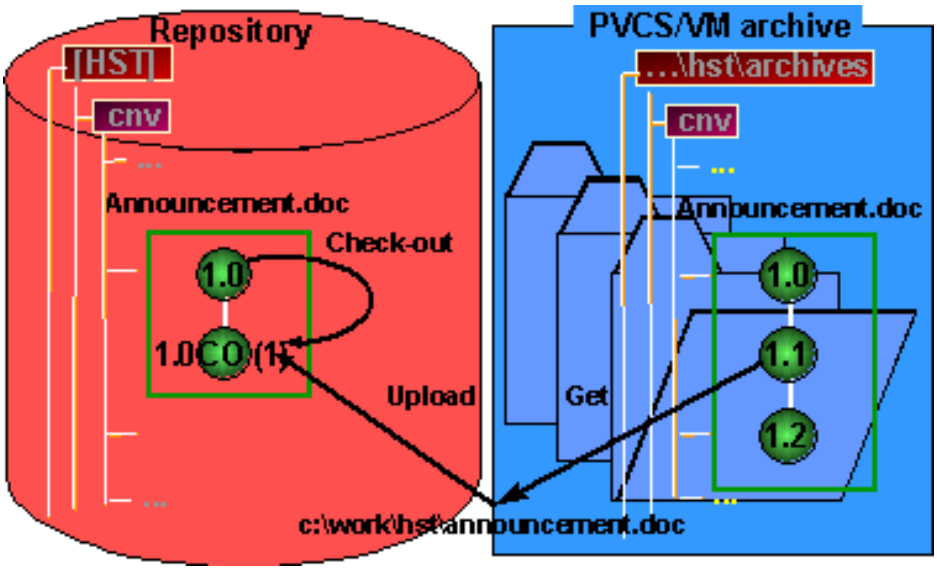
In this activity you will bring one or more file versions forward immediately from your third party source control tool to Oracle9i Designer.

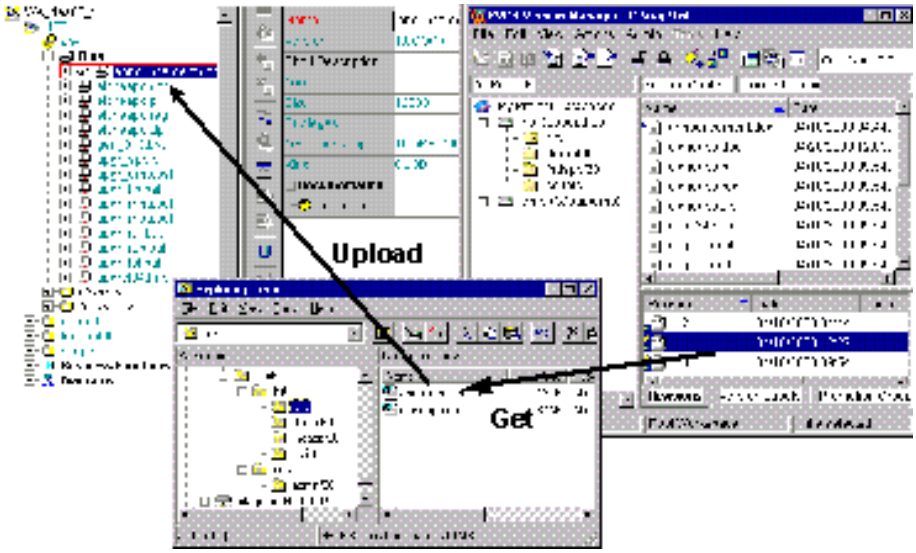
Building a version tree (immediately) comprises the following task steps:

1. Check out the relevant file in Oracle9i Designer (with a lock) that has multiple versions in your third party source control.
2. Get the file from your third party source control (without a lock) and overwrite the file on the file system in the prepared directory.
3. Upload and check in the file in Oracle9i Designer.

These steps are illustrated below for the file Announcement.doc.

Repeat these steps for each file that needs multiple versions and for which no more versions are available in your third party source control tool. For example file Announcement.doc has 3 versions (1.0, 1.1 and 1.2) and therefore you have to repeat this activity twice.





Building a version tree for files (immediately) should typically be performed by a developer. You therefore have to give intermediate access to developers that are responsible for these post-migration activities. Consult Chapter 7 "Migrate users and assign access privileges" for guidelines on repository access rights to the repository itself, to workareas and to containers.

Note that it is not necessary to run the compare utility after a build of another version. The build of another version is not as complex as with structured objects. Note also that the Oracle9i Designer repository has not implemented any kind of delta storage method while storing another file version. Another file version is just stored completely. In addition you can store a file in the repository either compressed or uncompressed.

Finally note that the above scenario can be very time consuming if you have many files with many file versions. You can then speed up the process by checking in only relevant file versions. For example only document1.doc version 1.0 and version 1.2 are used, while document version 1.1 is not used anywhere anymore. In this specific example you only need one extra activity to check in document1.doc version 1.2.

Building your version tree in time

During this activity you are building the version tree in time. You execute the check-out and check-in steps (as described and illustrated in the previous paragraph) whenever you need an additional version of a file. Your third party source control tool will be kept 'alive' (in read-only mode) as long as you need additional versions.

If you apply this migration method, you will also start with the lowest available file version, since you are eventually interested in the version tree. You can only build a version tree starting from the initial version.

You have to bring forward the additional file versions in the following circumstances:

- Setup of another promotion environment that involves versions other than those already available in Oracle9i Designer.
- Setup of another release that involves versions other than those already available in Oracle9i Designer.
- The continuation of development of one or more object versions that are not yet available in Oracle9i Designer.

Summary of migration strategies for building a version tree for files

Several options for building a version tree for files stored in a third party source control tool were described in the previous paragraphs. The following strategies are relevant if you combine these options:

1. The build of a version tree for files is not yet planned.
2. The build of a version tree for all file versions will be built immediately.
3. The build of a version tree for only a limited number of file versions will be built immediately.
4. The build of a version tree for only a limited number of file versions will be built in time.

The table below presents an overview of these options, which baseline version (lowest or latest) should be loaded initially and when you should choose for a specific strategy.

#	Build of version tree?	Baseline version	Multiple versions?	Rationale
1	Not planned (yet)	Latest	Single	Oracle9i Designer will not enter the versioned world yet because (most) projects are within development. In addition there are only a limited number of different versions
2	Immediately	Lowest	Multiple (all)	The number of files and different versions is limited or the migration can and will be organized via batch jobs (despite the large numbers of files and versions). The investment in collecting a batch utility for migrating your files can very well be justified. Note. A batch job for downloading the files from PVCS/VM and subsequently uploading the files into Oracle9i Designer - to build the version tree - can be based upon the available Command Line Interface of PVCS/VM and the Command Line Interface of Oracle9i Designer respectively.

3	Immediately	Lowest	Multiple (limited number)	There are only a few differences in object versions between the different environments and/or releases. The build effort of version trees is therefore limited.
4	In time	Lowest	Multiple (limited number)	The number of different file versions is substantial, but there is on the other hand a time constraint on the migration period.

Note that you can only select the most optimal strategy if you have full knowledge about the content of your third party source control tool. Again the report presented in the section "Get an overview of your third party source control tool repository", above, is a very productive way to support your strategy for building version trees.

Verify the migrated files

You can verify your migration effort for files for the following dimensions:

- Verify the file existence
- The existence of the appropriate number of additional versions (version tree)

Verify the file existence

The Oracle9i Designer repository should contain the same number of files as in the third party source control tool, unless one or more files are deliberately not brought forward to the Oracle9i Designer repository.

You can use the Repository Object Navigator to view the file objects (see for example the illustration in "Check-in of files", above) or even better you can use the API to produce a list of file objects for each root-container.

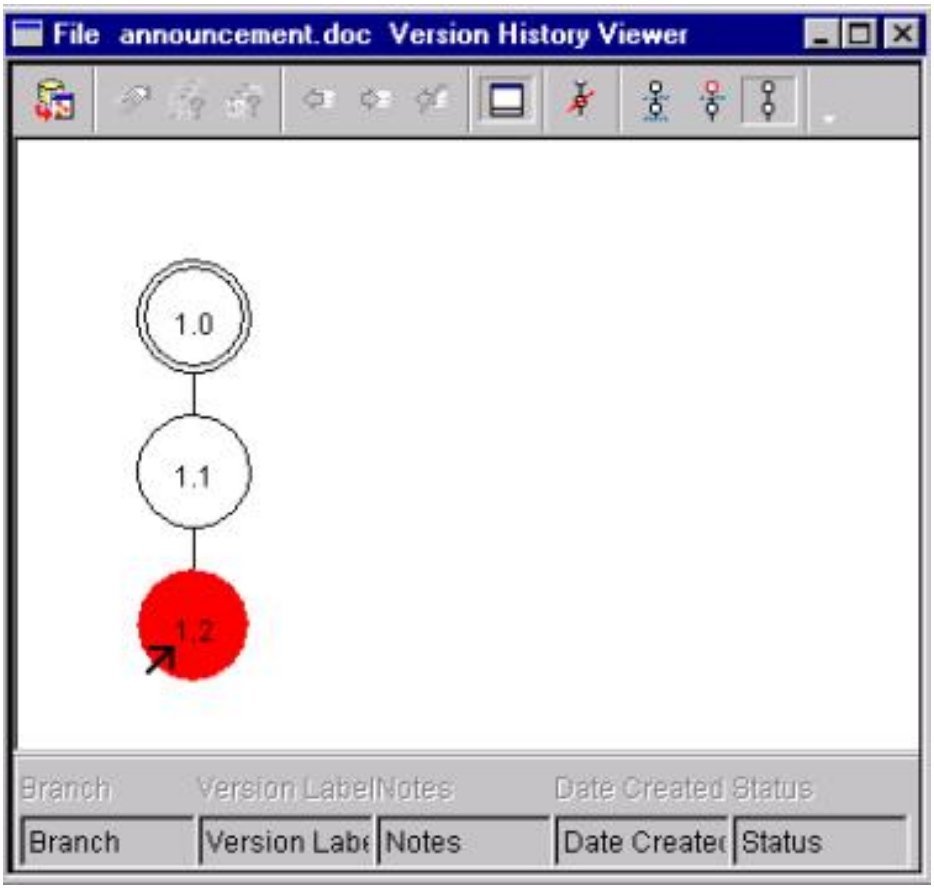
The existence of the appropriate number of additional versions (version tree)

The Oracle9i Designer repository should contain the appropriate number of file versions (or version tree) as was available in the third party source control tool.

Note that the exact number of file versions may deviate from the original number of file versions depending on the specific migration strategy for building a version tree for files (see previous paragraph).

Note also that the version number itself may deviate from the file version number if you have not brought forward all file versions and at the same time you have opted for the automatic numbering policy.

You can use the Version History Viewer (see below) to look up a version tree. Subsequently you can use the report presented in "Building a version tree for files" to compare the result in Oracle9i Designer with the content of your third party source control tool.



Compute statistics

Now is a good time to rerun "Compute Statistics" since your repository content is expanded with file objects (and maybe additional file versions). Running "Compute Statistics" will avoid a degradation in repository performance.

In Oracle9i Designer, you can run the utility directly from the Repository Administration Utility window by clicking the Compute Statistics button:

Chapter 5 Migrate files that are stored on the file system

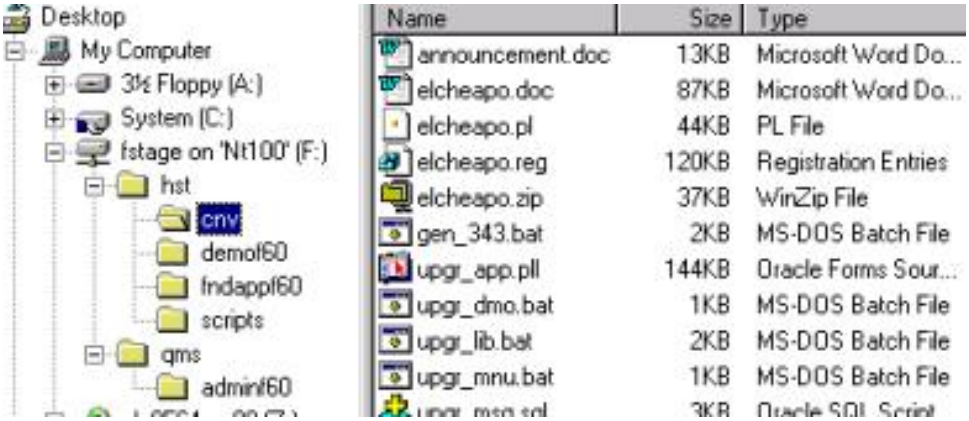
The migration of files, stored on the file system, should typically start after the migration of the structured elements. Only then are you able to integrate your files with the structured elements - both type of elements will then share the same root-container.

The migration of files into Oracle9i Designer comprises the following steps:

- Populate the file system with a baseline set of files
- Populate the Oracle9i Designer repository

Skip this chapter if you have already migrated your files from a third party source control tool or you do not (yet) have any files to migrate.

The migration of files stored on the file system will be clarified using an abstract of 'Headstart 5.0.5' files. It is assumed that the files are stored on the file system as depicted below:



Populate the file system with a baseline set of files

The migration of files will be based upon the file upload functionality in Oracle9i Designer.

The publication of a baseline file set should typically be organized in the following stages:

- Restructure your file system content
- Publish the baseline set of files on the file system.

Note that your file structure containing your candidate files could be stored on a local drive, a network drive or even on a Unix file system. Obviously in the case of a Unix file system you should have installed certain software (e.g. Samba) to make the Unix file system transparent - presented as a network drive - on your NT or Windows95/98 workstation.

Restructure the file system

The upload is an excellent opportunity to restructure your file system. It is highly recommended to use a staging area - a separate directory structure - in order not to interfere with ongoing development, testing or even production.

The restructure of the file system comprises the following stages:

- Determination of a generic directory structure
- A clean-up of your file system

Determination of a generic directory structure

The upload will not only capture the files, but also the owning directories. You should therefore investigate the file system content for each specific root directory (e.g. HST, QMS) in terms of (sub)folders and file types. Subsequently you can determine a generic directory structure based upon this investigation and create this generic structure in the staging area.

Clean-up checklist

Consult the following checklist to limit the number of files for uploading.

- Obsolete objects
- Temporary files

- Compiled object versions
- Multiple instances
- "out of context" objects

Applying the above checklist will result in an effective base line set, that you can upload and check in to Oracle9i Designer.

It is assumed that the content in the staging area for HST and QMS, depicted at the beginning of this chapter, is satisfactory (for example, no clean-up actions are necessary).

Populate the Oracle9i Designer repository

The procedure for populating the repository with files stored on the file system in Oracle9i Designer is the same as for files stored in a third party source control tool, once you have populated the files in the staging area.

Consult therefore "Populate the Oracle9i Designer repository" in Chapter 4 for a complete overview of the population steps.

Post migration steps for files stored on the file system

There are the following (possible) post-migration steps for files:

- Build a version tree for files
- Verify the migrated files
- Compute statistics

Build a version tree for files

Skip this activity if you do not (yet) want to enter the versioned world or if you do not (yet) want to have multiple object versions of files.

Compared with the storage of files in a third party source control tool, it is much harder to build up a version tree for files that are stored on the file system. It is more difficult to build an overview (see the illustration below) because additional versions of a file must either be stored under a different name or at a different location.

If you want to bring forward multiple file versions, you have to start with a baseline or lowest version in the publish step as discussed above. Obviously the version tree should be built starting from the lowest version.

The build of a version tree for files should be based upon an overview of your file system content, i.e. which file versions are available in which specific directories.

Your overview may look as follows:

```
Dir Filename                          Version
…
HST announcement.doc
HST \hst\prd\doc\ announcement.doc     1.0
HST \hst\acc\doc\ announcement.doc     1.0
HST \hst\tst\doc\ announcement11.doc   1.1
```

```
HST \hst\dev\doc\ announcement12.doc 1.2
...
QMS qmsolb50.olb
QMS \qms\prd\demof60\ qmsolb50.olb   1.6
QMS \qms\acc\demof60\ qmsolb50.olb   1.6
QMS \qms\tst\demof60\ qmsolb5016.olb 1.6
QMS \qms\dev\demof60\ qmsolb5017.olb 1.7
...
```

Probably the only way to build such an overview is to investigate the file content file-by-file, based on the different file versions (either captured using another file name or in another location).

Building a version tree for files should typically be performed by a developer. You therefore have to give intermediate access to developers that are responsible for these post-migration activities. Consult Chapter 7 for guidelines on repository access rights to the repository itself, to workareas and to containers.

Note that it is not necessary to run the compare utility after a build of another version. The build of another version is not as complex as with structured objects. Note also that the Oracle9i Designer repository does not implement any kind of delta storage method while storing another file version. Another file version is just stored completely. In addition you can store a file in the repository either compressed or uncompressed, to minimize the size of your repository.

If you are planning to build a version tree for files, you can either build it:

- immediately or
- in time.

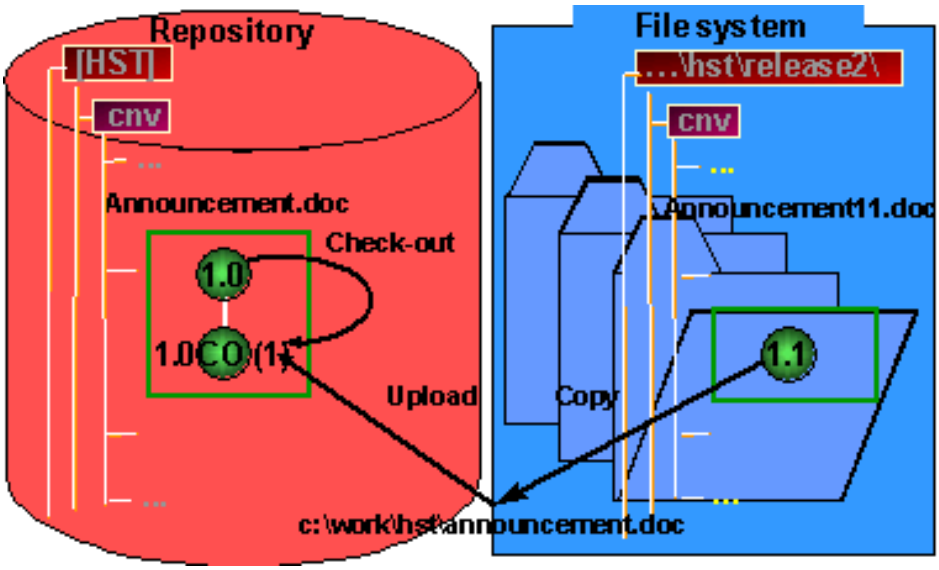
Building a version tree immediately

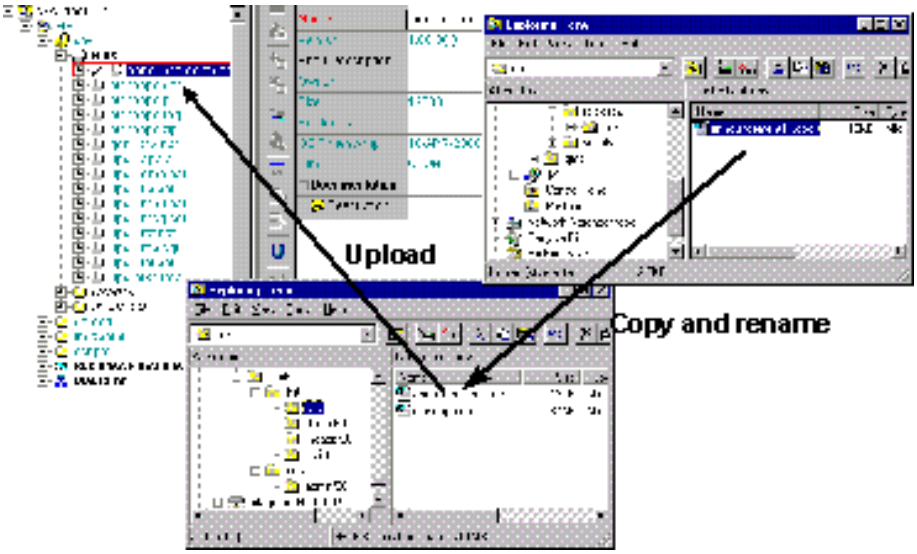
In this activity you will bring one or more file versions forward immediately from your file system to Oracle9i Designer.

Building a version tree (immediately) comprises the following task steps:

1. Check out the relevant file in Oracle9i Designer (with a lock) that has multiple versions in your third party source control.
2. Copy the 'next version' to your work directory. You will deliberately overwrite the file on the file system in your work directory. Note that you may have to rename the file if another version was captured with a different name.
3. Upload and check in the file in Oracle9i Designer.

These steps are illustrated below for the file Announcement.doc.





Repeat these steps for each file that needs multiple versions and for which no more versions are available. For example, file Announcement.doc has three versions (1.0, 1.1 and 1.2) and therefore you have to repeat this activity twice.

Building your version tree in time

In this activity you will bring one or more file versions forward in time from your file system to Oracle9i Designer.

Building a version tree (immediately) comprises the following task steps:

1. Check out the relevant file in Oracle9i Designer (with a lock) that has multiple versions in your third party source control.
2. Copy the 'next version' to your work directory. You will deliberately overwrite the file on the file system in your work directory. Note that you may have to rename the file if another version was captured with a different name.
3. Upload and check in the file in Oracle9i Designer.

These steps are the same as for building a version tree immediately. Consult therefore also the illustration in "Building a version tree immediately", above.

Repeat these steps for each file that needs multiple versions and for which no more versions are available. For example, file Announcement.doc has three versions (1.0, 1.1 and 1.2) and therefore you have to repeat this activity twice.

Finally note that the above scenarios can be very time consuming if you have many files with many file versions. You can speed up the process by checking in only relevant file versions. For example only Announcement.doc version 1.0 and version 1.2 are used, while announcement.doc version 1.1 is not used anywhere anymore. You only need one extra activity to check in Announcement.doc version 1.2.

Summary of migration strategies for files stored on the file system

Several options for building a version tree for files stored on the file system were described in the previous paragraphs. The following strategies are relevant if you combine these options:

1. The build of a version tree for files is not yet planned.
2. The build of a version tree for all file versions will be built immediately.
3. The build of a version tree for only a limited number of file versions will be built immediately.
4. The build of a version tree for only a limited number of file versions will be built in time.

The table below presents an overview of these strategies, which baseline version (lowest or latest) should be loaded initially and when you should choose a specific strategy.

#	Build of version tree?	Baseline version	Multiple versions?	Rationale
1	Not planned (yet)	Latest	Single	Oracle9i Designer will not enter the versioned world yet because (most) projects are within development. In addition there are only a limited number of different versions
2	Immediately	Lowest	Multiple (all)	The number of files and different versions is limited.
3	Immediately	Lowest	Multiple (limited number)	There are only a few differences in object versions captured in different directories. The build effort of version trees is therefore limited.
4	In time	Lowest	Multiple (limited number)	The number of different file versions is substantial, but it is too time consuming to bring all versions forward at once.

Note that you can only select the most optimal strategy if you have full knowledge about the content of your file system. The report featured in "Building a version tree for files", above, is a very productive way to support your strategy for building version trees for files.

Verify the migrated files

You should verify your migration effort for files for:

- the file existence
- the existence of the appropriate number of additional versions (version tree)

Verify the file existence

The Oracle9i Designer repository should contain the same number of files as stored on the file system. Unless one or more files are deliberately not brought forward to the Oracle9i Designer repository.

You can use the Repository Object Navigator to view the file objects (see for example the illustration in "Check-in of files", above) or even better you can use the API to produce a list of file objects for each root-container.

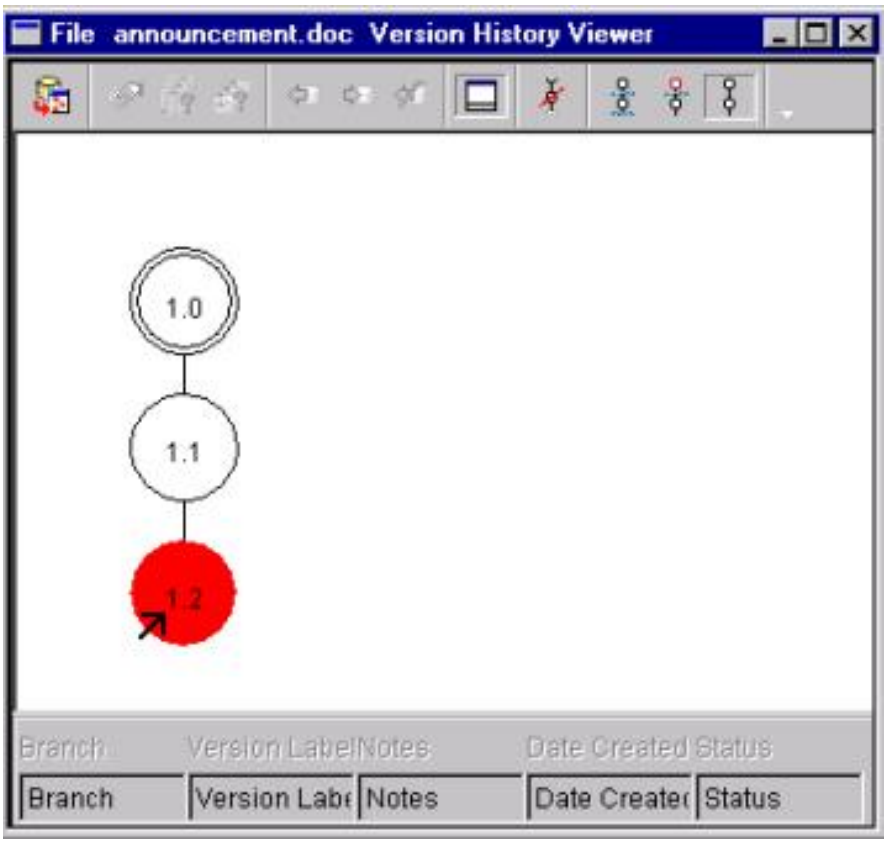
The existence of the appropriate number of additional versions (version tree)

The Oracle9i Designer repository should contain the appropriate number of file versions (or version tree) as was available in the third party source control tool.

Note that the exact number of file versions may deviate from the original number of file versions depending on the specific migration strategy for building a version tree for files (see previous section).

Note also that the version number itself may deviate from the file version number if you have not brought forward all file versions and at the same time you have opted for the automatic numbering policy.

You can use the Version History Viewer (see below) to look up a version tree. Subsequently you can use the report presented in "Building a version tree for files" to compare the result in Oracle9i Designer with the content of your third party source control tool.



Compute statistics

Now is a good time to rerun "Compute Statistics" because your repository content is expanded with file objects (and maybe additional file versions). Running "Compute Statistics" will avoid a degradation in repository performance.

In Oracle9i Designer, you can run the utility directly from the Repository Administration Utility window by clicking the Compute Statistics button:

Chapter 6 Reorganize a migrated Oracle9i Designer repository

The migration of structured elements and/or files should be followed by reorganisation activities. These reorganization activities involve the implementation of a specific implementation of your promotion model.

The following subjects are discussed in detail:

- Define workareas and configurations
- Remove obsolete elements
- Compute statistics

There are only a limited number of reorganization options available in a non-versioned world. For example you cannot create additional workareas next to the default 'GLOBAL_SHARED_WORKAREA'. You can only populate checked in objects in another workarea, but there are no checked in objects in a non-versioned world. Also you cannot create configurations since the configuration membership is based only upon checked in objects. Again no objects are checked in in a non-versioned world. You can only (re)organize your repository through (nesting of) containers in a non-versioned world.

You will find other and more comprehensive uses of workareas and configurations in a paper by Lucas Jellema called "Interior Oracle9i Designer" for the ODTUG2000 conference.

Define workareas and configurations

A promotion model can be implemented with a combination of workareas and configurations. If you adopt this specific implementation then basically all workareas - except the development workarea - will be based upon configurations.

In this section we assume that your promotion model looks as follows:

Development

System test

System integration

Production

Production fix

Obviously your promotion model may have more or fewer promotion levels.

In this specific promotion model implementation you will create workareas that represent promotion environments. In addition you may want to restrict each workarea in each promotion environment to one or more sub-systems. For example you will create a workarea for the development environment that contains the latest folder content of folders HST and QMS. A workarea for the test environment will be applicable to the same sub-systems, but it is based on different workarea rules. For example the workarea WA_TST_HSD represents the test environment for HST and QMS and is based on base line and patch configurations of HST and QMS. The term base line and patch configurations will be explained below.

The following workarea rules are applicable in the above described promotion model:

"LATEST(MAIN)" for the development workarea or "INCLUDE_FOLDER" if you break up your repository in different areas

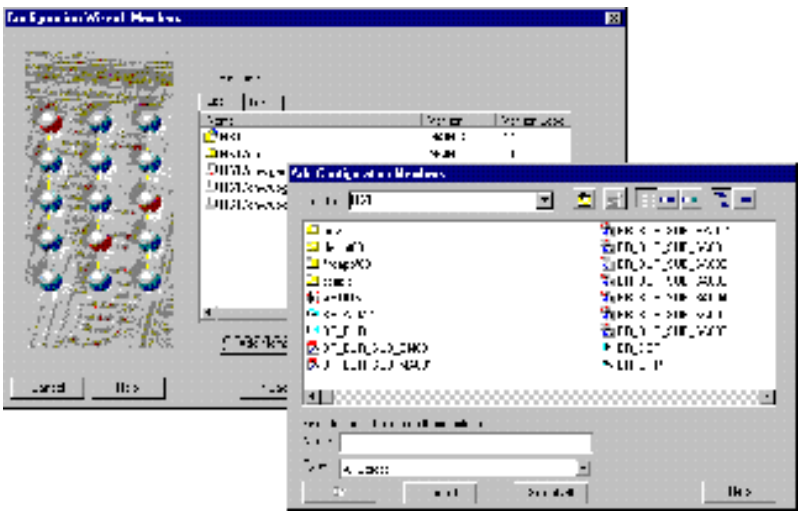
"INCLUDE_CONFIG" for example for the production or system test workarea

All non-development workareas will be based on configurations. You could classify two types of configurations:

A configuration representing a specific (full) release or base line of a specific root container (e.g. HST_505).

A configuration representing a patch consisting of a limited number of configuration items owned by a specific root container and applicable for a specific base line (e.g. HST_505_10).

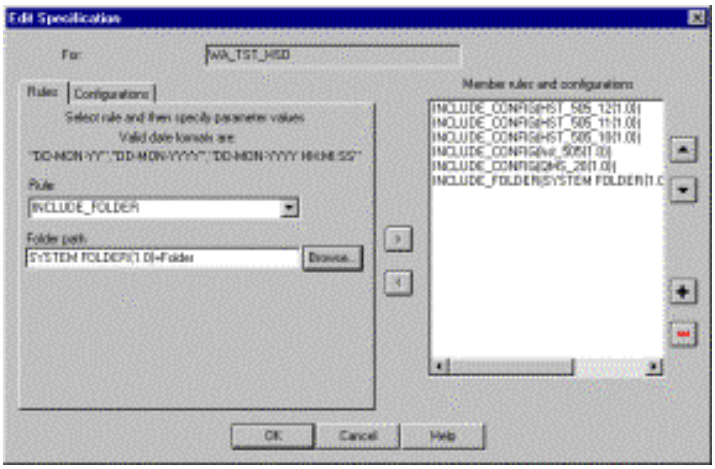
Obviously you can create the non-development workareas only if you have defined one or more configurations. You can create configurations (baselines and patches) via the configuration wizard.



You can either populate a configuration via a combination of rules - similar to the population of workareas - or you can just select objects from the repository (see the illustration above). Configuration rules are evaluated once and result in a set of specific object versions. The configuration rules are not saved as workarea rules are.

You should verify that your configurations not only include the structured elements but also the files (captured in folders). You will use the folders and files inclusion later on to build (or synchronize) your file system in each of your promotion environments (see Chapter 8).

With one or more configurations you are able to create your non development workareas with the workarera wizard.



"Base line" configurations can effectively be created via configuration rules, while "patch" configurations can be created via selecting the individual object versions.

The order in which you specify the workarea rules is very important. The workarea rules are evaluated from top to bottom. Another object version will not be evaluated whenever an object version is already populated by a 'higher' order workarea rule.

Your workarea for a specific promotion environment should eventually include one or more baseline configurations and/or one or more corresponding patch configurations of corresponding interrelated containers, as illustrated above.

Omitting one or more configurations may result in an incomplete definition of your components. For example your module definition will be incomplete if you omit the configuration that captures your preference sets that were initially defined (in the development environment) against your module.

Note that your development workarea ('WA_DEV_HSD') has a much more dynamic character than your 'other' workareas since your development workarea is based upon the 'simple' rule LATEST(MAIN) or FOLDER_LATEST_CONTENT(HST{1.2}). Your development workarea is changed each time an object is checked in. The 'other' workareas are only changed for each promotion. Each promotion results either in a additional configuration and/or a change of the workarea definition.

Note that the Oracle9i Designer security model fully supports the implementation of promotions via workareas. For example a developer may have write access on the container HST. At the same time he will probably have only read access to workarea WA_PRD_HSD and only read access to folder c:\oap\hst\prd. He or she will never be able to change the repository content in the production environment and will not be able to synchronize the file system in the production environment.

The number of configurations will grow significantly during the life time of an application. You should therefore adopt a naming standard for configurations that identifies at least the following components in the configuration name: <root container name>_<base line release nr>_(<patch release>). For example HST_505_11 represents patch 11 for base line 5.0.5 of container HST.

Different folder mappings in the development workarea for each developer

You may skip this section if you do not (yet) want to enter the versioned world.

The above folder mappings for the development workarea should typically be defined by the project configuration

manager - Chapter 7 gives a detailed description of this specific user group - who is responsible for the content of the development environment in the repository and eventually on the file system.

Developers however should not adopt this specific folder mapping for each root container in the development environment. They should adopt a private workspace for each container. These will not interfere with each other when they use a private workspace for changing the files. This private workspace can either be set locally (e.g. c:\work\hst) or on a share (e.g. f:\work\hst).

Note that the above "private" folder mapping is supported by Oracle9i Designer since the folder mapping is uniquely defined per workarea, container and workstation. The folder mapping is saved in the registry node HKEY_CURRENT_USER\Software\Oracle\FileSystem\… on each workstation.

Remove obsolete containers and other objects

Skip this activity if you do not (yet) want to enter the versioned world

In the previous tasks you may have created the following candidate obsolete objects:

- workareas that support the migration process (e.g. WA_des6i_ora8i)
- additional application systems that captures additional object versions (e.g. HSD_TST)

The initial workareas were replaced by workareas that support the development and deployment process (e.g. WA_DEV_HSD, WA_TST_HSD).

The additional application systems (not checked in) are no longer necessary once you have built the version trees of one or more structured objects and you have verified the differences via the compare utility.

You can remove these application systems via the delete application system option (available while clicking the right mouse button). However, note that you have to follow a specific order to delete an application system effectively. For instance you cannot delete an application system that has outgoing short-cuts.

Subsequently you can remove the 'migration' workareas. Note that all objects should be either checked in or removed to successfully delete a workarea.

Note also that the Oracle9i Designer repository has adopted a wastebasket strategy. Deleted objects will appear in the wastebasket. You should subsequently empty the wastebasket to definitely remove the objects from the repository.

Compute statistics

Now is a good time to rerun the "Compute Statistics" since your repository content is reorganized (and cleaned up). The "Compute Statistics" run will avoid a degradation in repository performance.

In Oracle9i Designer, you can run the utility directly from the Repository Administration Utility window by clicking the Compute Statistics button:

Chapter 7 Migrate users and assign access privileges

Your Oracle9i Designer repository should by now contain different workareas, containers and sub-containers, configurations and of course structured elements and files. These objects are however not yet accessible to subordinate users. Guidelines for creating subordinate users and the distribution of the appropriate access rights on the previously mentioned objects will be discussed in this section.

The concepts "workareas", "containers" and "configurations" are new for Oracle Designer 6i/Oracle9i Designer. In addition there are new system repository access rights for subordinate repository users. Therefore users and access rights are not migrated via the migration wizard. Subordinate users can be maintained in the Repository Administration Utility.

The following subjects are covered in this chapter:

- Classification of users in user groups
- Access rights per user group per repository, workarea, container and configuration
- A description of the Oracle9i Designer repository system access rights
- Database access rights for subordinate users
- Provided migration user scripts

The system repository access rights, specific access rights to the GLOBAL_SHARED_WORKAREA and container access rights are also valid in a non-versioned world. The configuration access rights are only meaningful in a versioned world

Classification of user groups and subsequent access rights

The table presented below combines the following dimensions:

- Configuration Management Role (e.g. Developer, DBA)
- Tasks per role - in a specific environment (e.g. development, test, production)
- Access rights per workarea, container and configuration

These user groups or roles may typically be represented in your organization to develop and deploy your (Oracle) applications. Obviously one specific user could be associated with one or more user groups. Typically you could reuse your access guidelines based on your user group classification.

Application life cycle tasks and subsequent access rights against workarea, containers and configurations

Role	Application life cycle tasks	Workarea	Container	Configuration
	Development			

Security Manager	Maintain the Development environment (owner of development workarea) in the Oracle9i Designer repository - via a personal key Approve repository users in the repository database instance and send request to DBA group for user creation Create Orace9i Designer repository development users and associate system repository access rights - via a generic key (schema owner of the Oracle repository) Provides access rights - via ROB roles - to the members (e.g. PCM, Developers, DBA) of the development domain (development workarea)	SAIUDVCU **(1)** Owner of workarea	NA	NA
Project Configuration Manager (PCM)/ Team leader (TL)	Plan, distribute and monitor minor and major change requests Plan, distribute and monitor bug-fixes Maintain the directory structure (owner of containers in the repository) and provide access rights to all participants to the directories Prepare changes for deployment into major and minor releases (configurations) Maintain configurations (owner of configurations) and provide access rights to all participants to the configurations Define branch labels to	SxIUDVCU	SAIUDV **(2)** owner of containers	SAIUDV **(3)** owner of configurations

	support parallel development			
Developer	Check in and check out elements and apply revision information as a result of change requests and bug fixes Perform impact analysis - for scooping purposes- leading to the additional storage of dependencies between structured elements and files Isolate elements into sets as a result of minor and major change requests and bug fixes Maintain the application database objects in the development database Maintain the application files on the development file system Maintain the dependency information Purge insignificant object versions	SxIUDVxx	SxIUDV	SxIUDV
DBA	Create repository users on request of SO and provide database system access rights in the repository database instance Create and maintain a development database instance Create application specific database administrative elements (storages, tablespaces) for the development database Prepare application specific database implementation properties (e.g. storages, tablespaces) for the development, test, acceptance and production domain	SxIUDVxx	SxIUDV	SxIUDV

	Check in and check out elements in the development domain to apply specific database implementation properties Review the generated and adapted database implementation scripts (captured in configurations). The review can only result in a approval or disapproval of the entire configuration. The DBA should not modify the content of the configuration Approve or disapprove the database implementation scripts (captured in configurations)			
Quality Manager	Look up the repository content to monitor the usage of CM standards and guidelines	Sxxxxxxx	Sxxxxxxx	Sxxxxxxx
	Release Management Preparation Workarea(s)			
PCM/TL	Owner of the personal release management preparation workarea	SAIUDVCU **(1)** Owner of workarea	NA	NA
	Test			
Security Manager	Maintain the Test environment carrier (owner of test workarea) in the Oracle9i Designer repository - via a personal key Create Oracle9i Designer repository test users and associate system repository access rights - via a generic key (schema owner of the Oracle repository) Provides access rights - via ROB roles - to the members of the test domain (test	SAIUDVCU **(1)** Owner of workarea	NA	NA

	workarea)			
Developer (as a tester)	Accept (or refuse) major and minor changes (packaged in releases or configurations) in the test domain Bring change requests (captured in releases) forward to the test environment Bring bug fixes (captured in releases) forward to the test environment Build the application database objects in the test database Build the application files on the test file system Verifies the repository content (what should be installed in the test domain) with the actual content on the file system and database in the test environment	SxxxxxCU	Sxxxxxxx	Sxxxxxxx
DBA	Create and maintain a test database instance Create application specific database administrative elements (storages, tablespaces) for the test database	NA	NA	NA
Application Support	Look up the content of the test domain	Sxxxxxxx	Sxxxxxxx	Sxxxxxxx
Quality Manager	Look up the repository content to monitor the usage of CM standards and guidelines	Sxxxxxxx	Sxxxxxxx	Sxxxxxxx
	Production			

Security Manager	Maintain the Production environment carrier (owner of production workarea) in the Oracle9i Designer repository - via a personal key Create Oracle9i Designer repository production users and associate system repository access rights - via a generic key (schema owner of the Oracle repository) Provides access rights - via ROB roles - to the members of the production domain (production workarea)	SAIUDVCU **(1)** Owner of workarea	NA	NA
Application Support	Accept (or refuse) major and minor changes (packaged in releases or configurations) in the productiont domain Bring change requests (captured in releases) forward to the production environment Build the application files on the production file system Verifies the repository content (what should be installed in the production domain) with the actual content on the file system in production environment	SxxxxxCU	Sxxxxxxx	Sxxxxxxx
DBA	Create and maintain a production database instance Create application specific database administrative elements (storages, tablespaces) for the production database Build the database objects in the production database	Sxxxxxxx	Sxxxxxxx	Sxxxxxxx

	Verify the repository content (what should be installed in the production domain) with the actual content in the production database			
Developer	Look up the content of the production domain	Sxxxxxxx	Sxxxxxxx	Sxxxxxxx
PCM/TL	Look up the content of the production domain	Sxxxxxxx	Sxxxxxxx	Sxxxxxxx

The specific abbreviations for access rights against workareas, containers and configurations are explained in the table below. Note however that not all access rights are applicable to each of these (administrative) elements. For example "Compile" and "Update Spec" are only applicable in the context of workareas.

Table 2. Overview of access rights for workareas, folders and configurations

Access right	Meaning
Select	Select element(s)
Administrate	Determine access rights for other users
Insert	Insert element(s)
Update	Update element(s) and object properties
Delete	Delete element(s)
Version	Version an element
Compile	Refresh a workarea (workarea only)
Update Spec	Change workarea rules (workarea only)

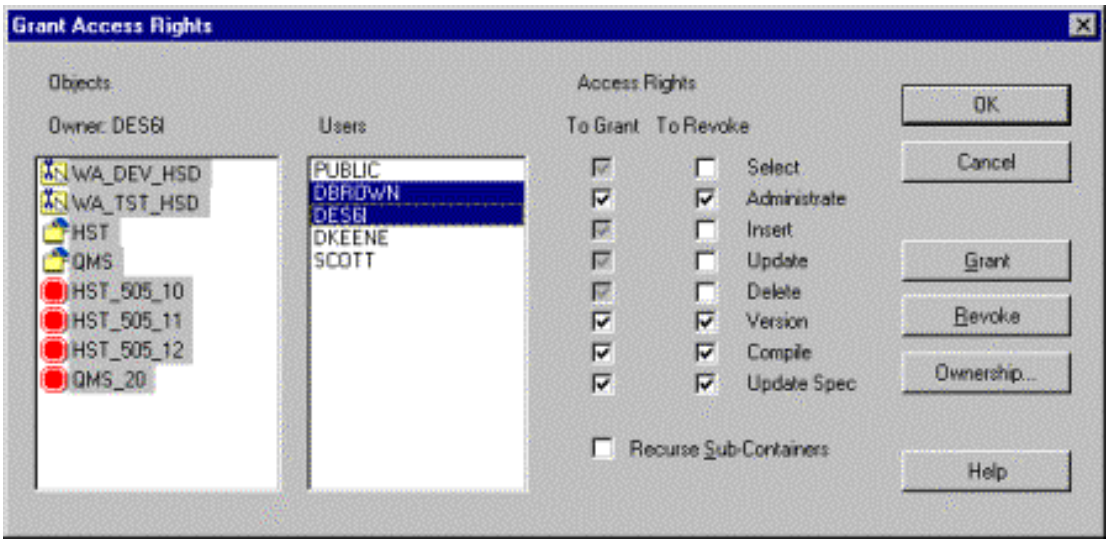
Note that you should also address the roles - and their derived access rights - that are involved with maintenance of the Oracle9i Designer tool stack. The table presented below addresses these roles and subsequent tasks - without the derived access rights.

Oracle Tool stack tasks

Role	Task
Security Manager	Set temporary password of the Designer repository owner for DBA to apply repository patches
Quality Manager	Design and implement migration strategies Monitor the usage of Configuration Management (CM) standards and guidelines in the repository Maintain CM Handbook & Standards and guidelines Distribute CM procedures

Support maintenance	Solve notifications (from first line support) via Oracle Customer Support Notify the availability of new Maintenance releases and patches of Oracle9i Designer (client side) Forward Oracle9i Designer server patches to DBA group
First contact point	Record all issues with the usage of Oracle9i Designer (first line of support) Notify the issues to second line support
Desktop support	Distribute new Oracle9i Designer versions and patches on NT client
DBA	Distribute Oracle database software versions (base lines and patches) Create Oracle9i Designer repository database objects (base lines and patches) on the database server Tune the Oracle9i Designer repository

The access rights against workareas, containers and even configurations can be (re)defined simultaneously - via a discontinuous select - as illustrated below.



You can launch the above screen by first performing a discontinuous select of the workareas, containers and configurations , subsequently choosing the grant access rights option - available on the file menu and define the access rights for one or more users simultaneously.

Note that you have to set access rights for each root container and its sub-container(s). The container grant access rights utility therefore has an option, called Recurse Sub-Containers, to set the same access rights for all sub-containers.

Note also that the final access rights for a specific object (e.g. insert, select, delete or update of an entity, a table or a module) are determined by an evaluation of both the access rights against the workarea and the container(s). For example if an Oracle9i Designer repository user has select rights on the container HST and select, insert, update, delete and version on the WA_DEV_HSD workarea, in the end he or she can only select any object within the container HST (despite his or her extended access rights against the workarea).

System repository access rights

The system repository access rights are significantly changed. You can define the following system repository access rights for a specific user:

Abbreviation	System Oracle9i Designer allowances
Management	
WA	Workareas
CFG	Configurations
CTR	Containers (Folders or application systems)
USR	Users
BRL	Branch Label
DPD	Dependencies
REG	Registration
Perform	
SPY	Set Policy
FRC	Force
PRG	Purge
GPR	Global Purge
Connection	
RON	Repository Object Navigator
RAU	Repository Administration utility
MTX	Matrix Diagrammer
RPTL	Reporting Tool
CLI	Command Line Interface
VHV	Version History Viewer
VEV	Version EventViewer
MIG	Migration wizard

Note that the following system repository access rights are applicable only in a versioned world: Containers, Users, Dependencies, Registration, Force, Repository Object Navigator, Repository Administration Utility, Matrix Diagrammer, Reporting Tool and Command Line Interface.

You could create another table that combines the above access rights and the earlier presented roles: system repository access rights per role.

Database system privileges

Any subordinate user - no matter what repository role - needs only the following system privileges on the repository database instance:

- Create Session
- Alter Session

With the above database system privileges, any subordinate user can access all Oracle9i Designer tools and diagrammers and populate the repository with (new) elements.

User migration scripts

Oracle9i Designer comes with two dynamic scripts that may support the user migration:

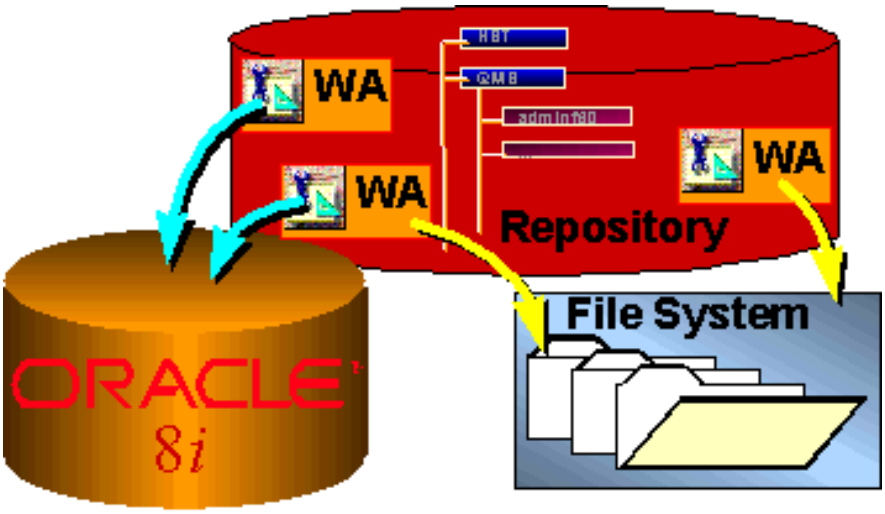
- ckgenusr.sql - This sql script creates the database user script.
- ckgenprv.sql - This sql script sets a default set of the system repository access rights per user.

You can find both scripts in the <oracle_home>\repadm61\utl directory. These scripts must be executed in the source repository database instance while the results must be executed in the target repository database instance.

It should be clear that - after execution in the target repository - you need to revise the system repository access rights for each user - depending on his or her role. Subsequently you must define the access rights per workarea, container and configurations - based upon the classification presented above. The ckgenprv.sql script does not set these latter access rights since they are new in Oracle9i Designer. Note also that both scripts may bring forward obsolete users - users that were not removed in the source repository.

Chapter 8 Synchronize the file system and the database with the repository content

In the previous steps you have populated and reorganized the repository and you have made it accessible in a controlled way to subordinate users. A significant part of your repository content will ultimately populate your databases and file systems. This chapter will discuss the options available to synchronize the file systems and the databases with the repository content (see the illustration below). These synchronization options allow you to adopt model-based development and deployment during the complete lifecycle. You can effectively use the repository content as a reference model for all your promotion environments. You are able to rebuild your file system and database in each environment on any specific moment. Moreover you will have a variety of impact analysis tools available to solve the issues effectively in any environment since you can use any diagrammer and the dependency analyzer in the context of any workarea.



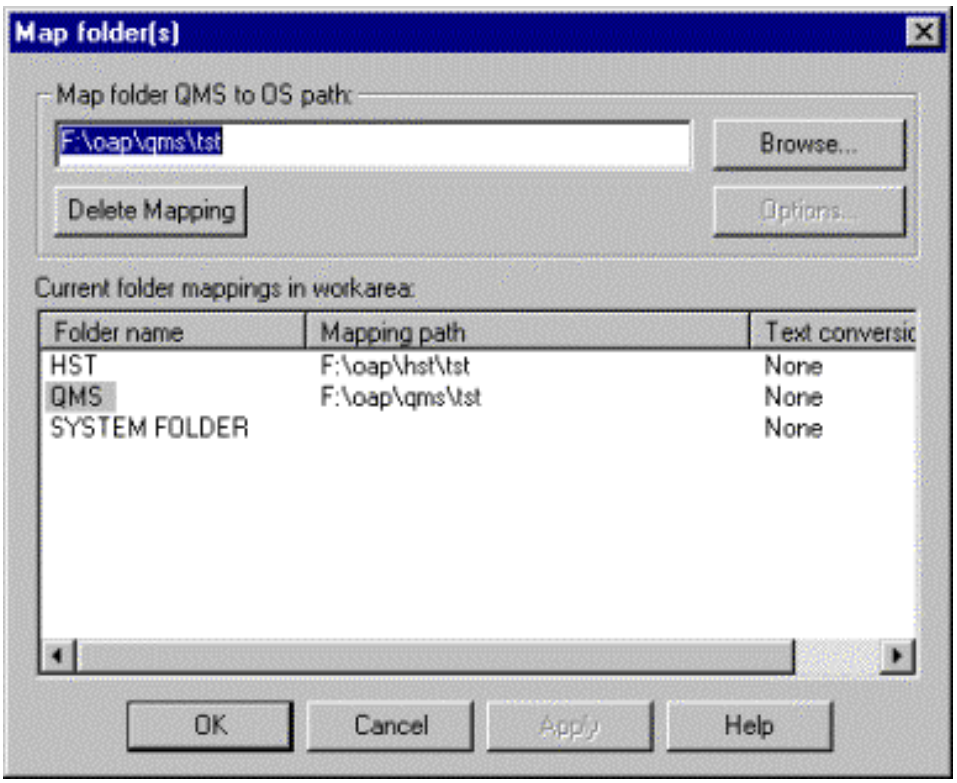
Both synchronizations will be discussed in detail in the following sections.

Effectively you can only synchronize the file system and database in the development environment (via the GLOBAL_SHARED_WORKAREA) if you have not yet entered the versioned world, because you have only one object version available.

File system synchronization

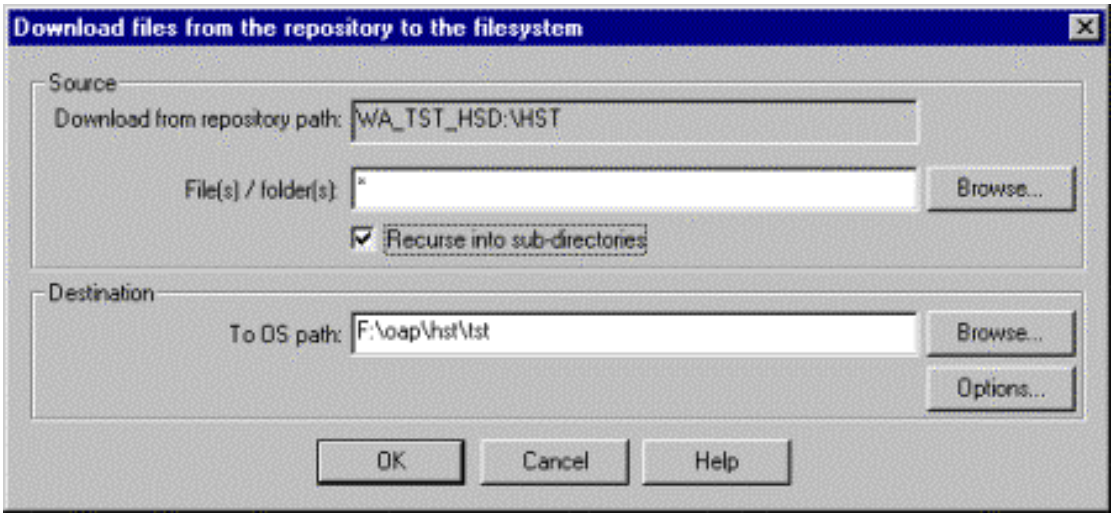
Each promotion environment for a specific application can be represented on the file system by a specific root directory. For example the test environment for container HST can be represented by the directory f:\oap\hst\tst. At the same time the directory f:\oap\hst\prd represents the production environment of HST. This allows you to test the form HSD0002F version 2.0 without interference of version 1.0 of HSD0002f in production - captured somewhere below f:\oap\hst\prd.

The synchronization of the file system with the repository content for each promotion environment is based upon the folder mapping option.



You are able to apply a different folder mapping for each workarea. Remember that each workarea was representing a specific promotion environment. For example the folder mapping for container HST in the development environment - within workarea WA_DEV_HSD - can de defined as f:\oap\hst\dev. While the folder mapping for container HST in the test environment - within workarea WA_TST_HSD - can de defined as f:\oap\hst\tst.

Once you have established the folder mappings for each container in each workarea you can use the download option to actually synchronize the repository content with the file system. The download option extracts all files from the repository and copies these files to the file system (illustrated below).



Note that the download option not only synchronizes the current directory but also all recursive directories.

Remember that most workareas (except the development workarea) were based upon one or more configurations. Each workarea content change - as a result of configuration membership changes - should therefore be succeeded by a synchronization step on the corresponding file system to keep the repository content in sync with the file system.

File synchronization options

Oracle9i Designer has implemented the following options to communicate with the file system and the repository in the context of files.

- Upload
- Download
- Synchronize
- Check-in

The meaning of these file options will be explained in detail in the following sections.

Upload

The upload option allows you to store files and/or folders in the repository from a specific operating system path - either based upon the default folder mapping or session specific value.

Note that the upload option was used during the migration to populate the repository initially with files.

Download

The download option allows you to publish files and/or folders from the repository from a specific folder path on the operating system in a specific operating system path - either based upon the default folder mapping or session specific value.

Note that the download option can be used to synchronize the repository content with the file system.

Synchronize

The synchronize option updates files in the repository with the contents of the corresponding files in a mapped file system or vice versa, depending on which is the later version.

Checkin

The check-in option versions the (uploaded) file, i.e. the file is already stored in the repository and in addition it will be populated with version information.

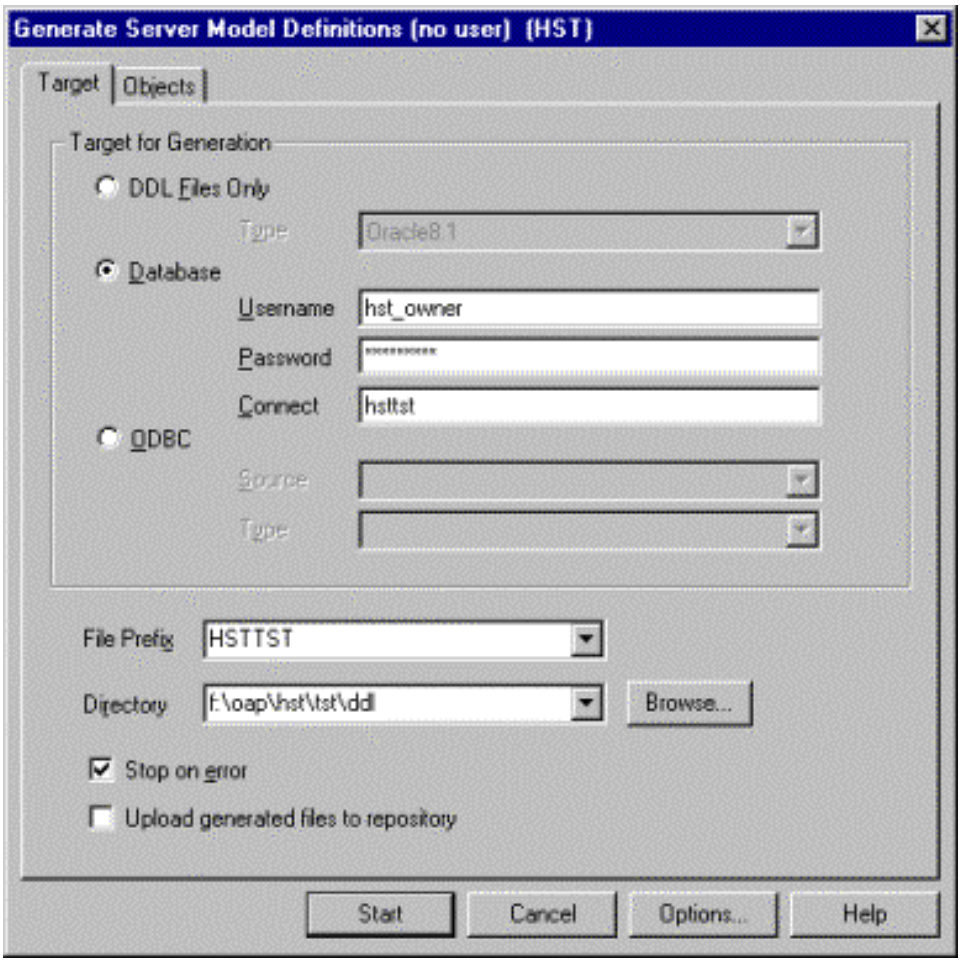
Note that the check-in action is applicable only in a versioned repository.

Database synchronization

Each promotion environment for a specific application can (preferably) be represented via a database schema in a separate database instance. For example the database objects in the test environment for container HST in workarea WA_TST_HSD can be represented by a schema owner HST_OWNER in a specific database instance called HSTTEST. At the same time the database schema HST_OWNER in database instance HSTPROD can represent the database objects in the production environment of HST in workarea WA_PRD_HSD in another database instance. This allows you to test table HSD_PROJECTS version 2.0 in the test database instance without interference of version 1.0 of HSD_PROJECTS in production - captured in a production database instance - against the HST_OWNER schema owner.

The synchronization of the database with the repository content for each promotion environment can be implemented via the DDL generator or even the DDL generator in batch - in the context of a specific workarea. Note that the DDL generator not only generates "create database object" scripts but also reconcile scripts or "database delta scripts" if you generate against a schema that already contains all or part of the application database objects. Further note that Oracle 8*i* supports the "alter table drop column" statement. The database synchronization in each database instance can therefore be supported to a great extent with the DDL generators in each workarea, which makes it very attractive to adopt this promotion model.

The proposed promotion implementation model also supports the handling of database objects. For example a developer may have write access on the container HST. At the same time he will probably have only read access to workarea WA_PRD and only read access to schema HST_OWNER in the production database. He or she will therefore never be able to change the database structure in the production environment.



The above screenshot shows the DDL generator in the context of workarea WA_TST_HSD for the schema owner HST_OWNER in a test database instance - via HSTTEST connect string. Note that you can run the DDL generator for all database objects as well as for a sub-selection.

Synchronization add-ons

Both synchronizations (repository content versus file system and repository content versus database) can be verified with the following add-ons: Keyword Expansion kit and CheckRelease.

Keyword Expansion Kit
"Keyword expansion" refers to functionality that acts upon objects that are being checked into the repository. The keyword expansion utility will look for keywords in the checked in object. Supported keywords are expanded or replaced with actual values of check-in parameters, such as author, version label, date and check-in notes. This is functionality very common to most Source Code Control tools. It allows for complete identification of files downloaded from the repository: through keyword expansion, every file can and should always contain the appropriate Version Label assigned in the repository.

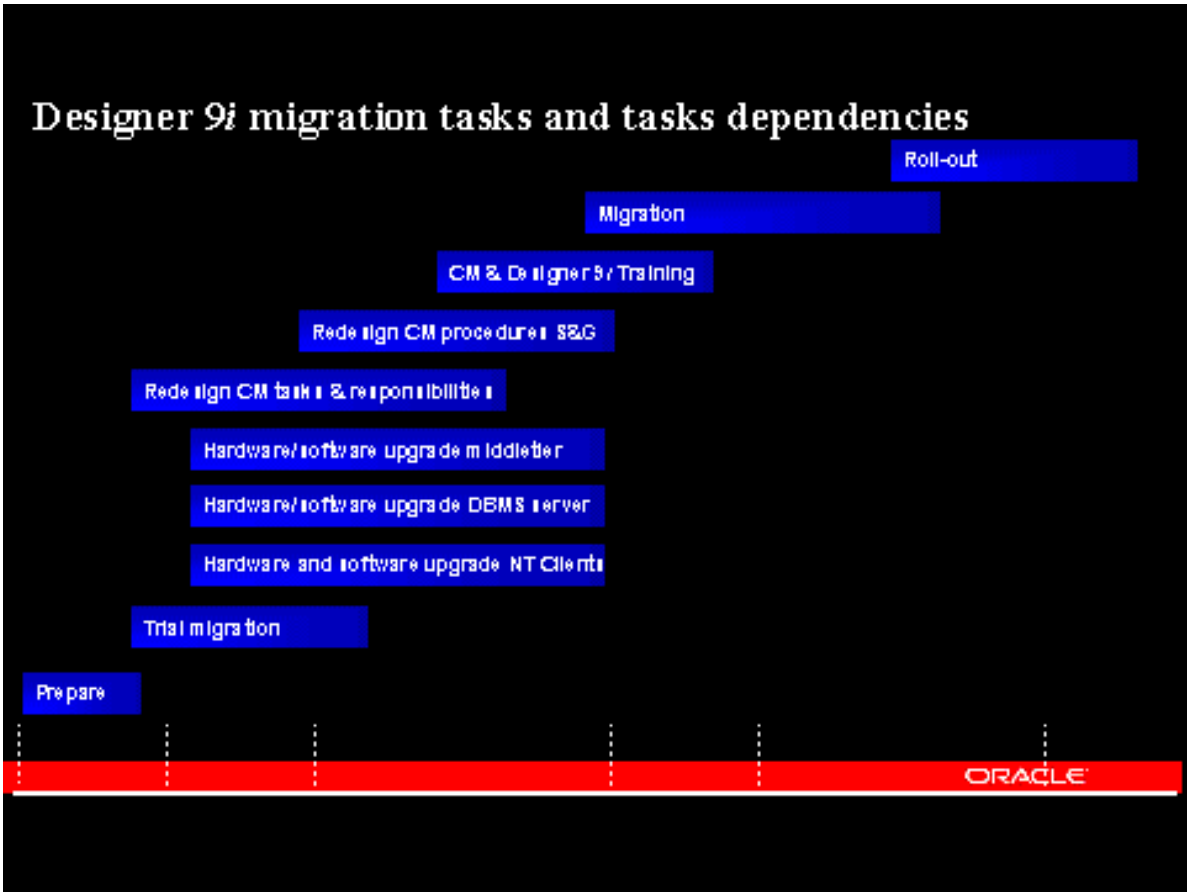
CheckRelease
The CheckRelease utility comprises a GUI (Oracle Forms Module) that allows you to confront the content of one or more application systems (e.g. tables, views, files) in the context of a workarea with a build of these application systems on a file system and database. It uses therefore one or more intermediate release tables that are populated with repository information like the name, type and version number of the repository objects. This release table content represents what should be built on the file system and database in a specific environment.

Both add-ons are available via the Supplement Option. Visit www.otn.oracle.com for more information about the

Supplement Option (e.g. What it is, pricing and conditions).

Chapter 9 A typical migration plan

The previous chapters have given a technical insight in the migration efforts - represented by the blue Migration square (the task before Roll-out) in the sheet below. As you can see in the sheet below there are many more tasks to fulfill to finish an Oracle9i Designer migration project successfully. This chapter will give an overview of these other migration tasks, their characteristics and their dependencies.



The introduction of Oracle9i Designer in your organization can effectively be organized via the following phases:

- Preparation
- Trial migration
- Hardware and software upgrades Win32 clients
- Hardware and software upgrades DBMS server
- Hardware and software upgrades middle-tier
- Redesign Configuration Management (CM) tasks and responsibilities
- Redesign CM procedures and standards & guidelines
- CM and Oracle9i Designer training
- Migration
- Rollout

Preparation

The preparation stage should deliver your migration plan. Such a plan should cover the following decisions:

- Big bang migration of structured elements or a stepwise migration
- Bring in files in the repository as well?
- Introduction Oracle8i/ Oracle9i functionality
- Migration strategy for database objects, Oracle Developer components and Web PL/SQL modules as described in Parts 3, 4 and 5
- Migration or implementation of business rules
- Scope of migration in the Configuration Management area
- Redistribution of tasks as a result of a different way and writing method

It should also comprise:

- Opportunities, Objectives & Scope
- A detailed task overview with resource usages
- Project organization
- Costs and benefits

Note that a detailed task overview can be based on the overall tasks presented above.

Trial migration

A trial migration can be planned on separate hardware platforms - client, middletier and server - to evaluate the repository size and performance aspects. In addition you can reuse the trial migration environment for educational purposes, to develop your set of configuration management procedures and standards & guidelines, and it can provide effective output for the subsequent hardware and software installation tasks on the client, middle-tier and server platforms.

Hardware and software upgrades Win32 clients

Your current clients (Windows 98/NT/2000/XP) may not be sufficient in terms of disk space, processing capacity and memory to run the Oracle9i Designer software and subsequent components like Developer. A hardware and software upgrade may take some time, specifically if you have a significant number of win32 clients. In addition you can prepare the automation of the software distribution of the Oracle9i Designer tool stack via for example the Novell Application Launcher or other client distribution tools like SMS.

Hardware and software upgrades - DBMS server

The hardware and software upgrade of the database server covers - at least - the database instance for the Oracle9i Designer repository. It could also cover the upgrade of the databases that capture your applications.

Oracle9i Designer is certified with 81700 or higher and Oracle9i patched to 9.0.1.2. Note that a database upgrade can also implicate an upgrade of the operating system. Therefore not only the DBA department but also the department that manages server platforms must be involved.

Hardware and software upgrades - middle-tier

Oracle9i Forms and Oracle9i Reports - part of the Oracle Internet Development Suite (iDS) - do not support a client/server architecture. It is therefore highly recommended that you introduce or update your middle-tier platform with new software components - or an upgrade - to Oracle 9iAS. In addition to hardware upgrade issues, it may also involve a new version of the underlying operating system. In addition - specifically if you introduce a middle-tier - you must deal with management (support) and knowledge transfer of the middle-tier software. In practice, this stage and the organizational impact proves to be the most underestimated.

Redesign Configuration Management (CM) tasks and responsibilities

The introduction of Oracle9i Designer and the underlying repository brings along a new writing and working method for a wide range of participants, since the repository will be used for development as well as deployment. Your big challenge is to cover all the tasks (see also the matrices for application life cycle tasks and tool stack maintenance tasks presented in Chapter 7 "Migrate users and assign access rights") in your specific organization. This stage is perhaps the most challenging one of your entire migration project - so do not underestimate it.

Redesign Configuration Management procedures and standards & guidelines

As soon as the redistribution of tasks and responsibilities is well under way you can start with the production of the redesign of configuration management (CM) procedures and standards & guidelines.

You could try to produce a big fat configuration management handbook - that most likely at the end of the day will only be consulted by you. You could also try to adopt a “less is more approach”, i.e. produce multiple Quick Reference Cards that cover for example your CM procedures, CM Standards & Guidelines and tool support. You could think of Quick Reference Cards candidates:

- Repository Terminology
- Check-in and Check-out Quick Reference Card
- Overview of Designer/Repository Tools
- Naming Standards (workareas, configurations, ddl scripts)
- Database synchronization and DDL generation guidelines
- Guidelines for the use of the Database Design Transformer
- Management of Repository Access rights
- Setup of a parallel environment
- Setup of a bug-fix environment

A few examples of the above listed Quick Reference Cards are presented in Appendix B.

Configuration Management and Oracle9i Designer training

An effective and productive migration and subsequently the effective usage of the Oracle9i Designer tool stack and subsequent CM procedures relies heavily on well trained participants. By now it should be clear that not only developers need training but also other participants, such as project configuration managers, application support representatives and the security department that provides access to the repository.

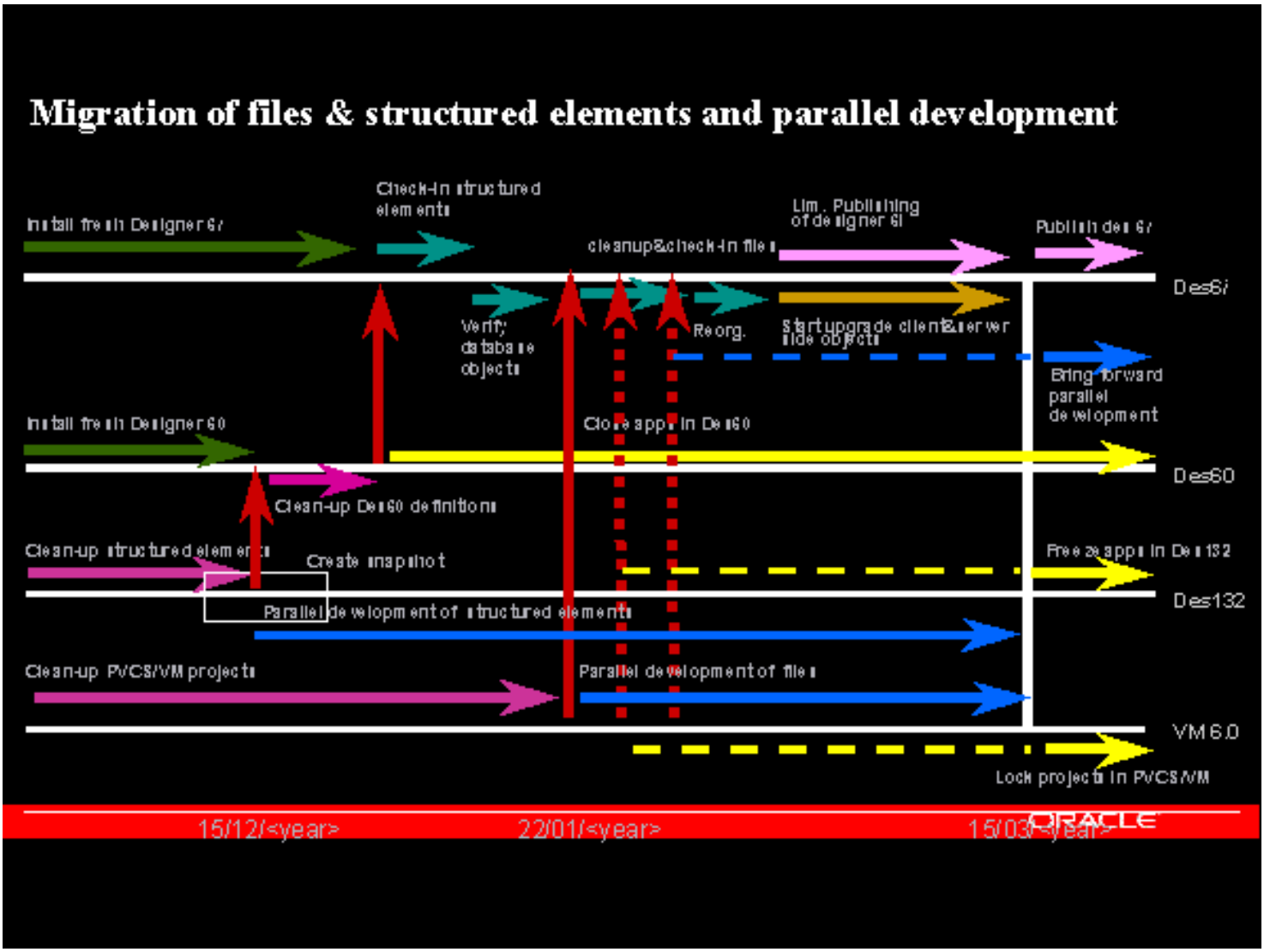
An effective CM and Oracle9i Designer training plan therefore should be based on the profiles or roles as covered in the matrices presented earlier. Therefore you cannot start the training task before the first results of the redistribution task are available. Subsequently you should evaluate for each profile the need to cover one or more of the following training subjects:

- Oracle9i Designer new database features
- Oracle9i Designer Forms and Reports generation new features
- Oracle9i Designer WEB PL/SQL modules new features
- Oracle9i Designer new Configuration Management features
- Business Rule implementation via CDMRule Frame
- Oracle9i Designer Forms and Reports generation via Headstart
- Knowlegde transfer of one or more custom Configuration Management procedures - perhaps captured in the Quick Reference Cards
- Subsequently you can fill in the identified training needs per profile with a combination of standard available courses and/or custom courses.

Migration

The technical migration aspects are already discussed in detail in the previous chapters of this part (Part 2). In addition more specific migration details on an object-type-by-object-type basis will be provided in Parts 3, 4 and 5. The migration task can only start - as will be pointed out below - with skilled developers. Your migration start date is therefore not only dependent of the hardware/software availability, but also on a skilled staff that can perform one or migration activities.

The coherence between these different migration steps of structured elements and files in the different source repositories and Oracle9i Designer repository and the inevitable parallel development as a result of these migration steps can be visualized as follows:



From top to bottom the following repositories are depicted:

- PVCS/VM source repository for files - could also be another third party source control
- Designer 1.3.2 repository - source repository
- Designer 6.0 repository - intermediate repository
- Oracle9i Designer repository - target repository

From left to right a time period is depicted - somewhere between 1 week and 3 months depending on the size, number and complexity of the application systems and/or projects and not in the least on the specific migration scenarios for the specific object types - as discussed in Parts 3, 4 and 5.

To explain the parallel development challenge you should start at the lower left corner in the Designer 1.3.2 repository and the third party repository (e.g. PVCS/VM repository) with a clean-up (*depicted by the purple arrows*).

Parallel with the clean-up stage you can start with a fresh installation of Designer 6.0 - to capture intermediate application systems - and Oracle9i Designer - to capture the target application systems and files (*depicted by the dark green arrows)*. If your application systems already live in Designer 6.0 you should replace all references to Designer 1.3.2 with Designer 6.0 and you can skip the fresh installation of Designer 6.0. After the clean-up you export a coherent set of application systems into the intermediate Designer 6.0 repository (*depicted by the lower vertical red arrow*). At the same time you import this set back in the Designer 1.3.2 repository under different application names. This set will be used as a snapshot of your migration start.

Developers can continue with their work in Designer 1.3.2 although you should not encourage them to apply a lot of changes, since these changes are by definition not captured in the imported set. In the intermediate Designer 6.0 repository, clean-up activities like re-connect share links can be picked up (*depicted by the upper purple arrow*). As soon as these clean-up activities are finished you can launch the migration wizard in Oracle9i Designer, connect to the Designer 6.0 repository and make the selection (*depicted by the upper vertical red arrow*). The migration wizard will bring the application set forward in the Oracle9i Designer repository. Subsequently you can freeze the application systems in Designer 6.0 (*depicted by the yellow arrow*) and start with a check-in of the structured elements in Oracle9i Designer - to capture the fresh migration result (*depicted by the light green arrows*). You could also leave them un-versioned and apply your migration scenarios for database objects as described in Part 3.

While one or more of your developers is executing the database migration scenario you can start with the upload of files on a project-by-project basis from the PVCS/VM repository as described in Chapter 4 (*depicted by the right upper - solid and dotted - vertical red arrows*). You can close down each PVCS/VM project as soon as the files are uploaded in the Oracle9i Designer repository (*depicted by the dotted yellow arrows*). The parallel development time for files can be very short - less than a day - since you do not have to verify the generation capability as with structured elements.

While the files from one or more PVCS/VM projects (or file system) are uploaded you should focus on finishing the migration steps for database objects as described in Chapters 3 and 4 of Part 3 (*depicted by the right light green arrow - verify database objects*). By now you should have reached the reorganization phase in your target Oracle9i Designer repository. In this phase you will organize your repository in one or more development workareas - containing the migrated application systems and files - and you will bring forward your subordinate users with the appropriate access rights, see also Chapter 7 (*depicted by the right light green arrow*). With more subordinate users - developers - you can start effectively with the migration scenarios for Development components and WEB PL/SQL modules as described in Parts 4 and 5. In addition you may want to apply one of the database new features migration steps as described in Chapter 5 of Part 3 (*depicted by the brown arrow*).

While applying these migration steps in the target repository you will also bring forward all the changes from the source repository since the start of the migration. You could use the snapshot very effectively to narrow down the exact changes since the start of the migration. As soon as you have brought forward the source repository changes - manually - in the target repository you can close down the source repository application system (*depicted by the dotted blue arrow*). The real challenge here is to close down the source repository application systems as quickly as possible or to phrase it differently to minimize the parallel development period. Note that you must bring forward more changes with a longer parallel development period. You can accomplish a short period by basically bringing in as much “qualified” resources as you can to execute the migration scenarios of Parts 3, 4 and 5.

Rollout

Finally - after finishing the migration of structured elements and files as described in Part 2 and after finishing the migration scenarios of Parts 3, 4 and 5 - you can roll out Oracle9i Designer to all participants that are involved in development and deployment. Basically you are offering on-site support during first time Oracle9i Designer use by all participants during this phase. The support activities must elaborate on the training effort. They should definitely not replace the training courses. It turns out that the acceptance and effective usage of Oracle9i Designer will speed up significantly if you can find enthusiastic and skilled representatives in each department. These early adopters in each department can soon take over the support task and take care of wide effective usage of the Oracle9i Designer tool.

Prev Next

Contents

Oracle9i Designer Migration Guide
Part 3. Generating and migrating existing database designs to Oracle9i Designer


Contents

Chapter 1 Introduction

This part (Part 3) of the migration guide provides the information necessary for upgrading, to Oracle9i Designer, the database definitions that you designed and generated using earlier releases of Oracle Designer.

Part 3 discusses database migration from the following earlier Designer releases:

- 1.3.2
- 2.1.2
- 6.0

Part 3 assumes that you have already installed Oracle9i Designer and migrated your repository. (See instructions in Part 2 of this Migration Guide.) Part 3 then explains steps you have to take so that you can:

- generate your database design definitions from Oracle9i Designer and achieve the same generated results you had from earlier releases, and
- take advantage of new features that have been added to Oracle Designer since your previous release.

Throughout the document, special mention is made of any migration issues known at the time of publication of this document.

Note that this Part 3 fully replaces an earlier published database migration guide for Oracle Designer 2.1x/6.0 and Designer 6*i* (October 2000).

Part 3 starts with an introduction (Chapter 1) that gives an overview of the possible migration scenarios for database objects:

- Scenario 1. Migrate, Regenerate All, No Redesign, and
- Scenario 2. Migrate, Regenerate All, With Redesign.

Subsequently, Chapter 2 "New database features" describes all changes/new features since 1.3.2 with respect to database design that you need to know to execute your migration steps for either scenario 1 or scenario 2. The new features chapter is structured by specific Oracle Designer release: Migrating from Designer 1.3.2, Migrating from Designer 2.1.x, etc.

Chapter 3 discusses general database migration issues, integration steps for database objects you always have to consider despite your database scenario.

Chapter 4 "Scenario 1. Migrate, Regenerate All, No Redesign" describes the necessary regeneration steps for scenario 1.

Chapter 5 "Scenario 2. Migrate, Regenerate All, With Redesign" describes in addition specific redesign steps for database objects.

Note that Part 4 "Forms Migration" and Part 5 "Web PL/SQL migration" is organized with a similar structure.

This part of the migration guide, for database objects, continues where the second part of the migration guide has stopped. It is considered that you have completed all the steps from the second part: your Oracle9i Designer repository contains freshly migrated structured objects (e.g. database objects) and files. Note that it does not make any difference for this part if files and/or structured objects are checked in ("versioned") or just stored.

The database objects in Oracle9i Designer are compliant with Oracle8*i*, i.e. you are able to store, version, generate and - partly - visualize all your database specifications in Oracle9i Designer into an Oracle8*i* database and at the same time make use of specific Oracle8*i* functionality. Note that at the same time you are still able to generate Oracle7 and Oracle8 syntax.

There is a lot more new functionality introduced in the Oracle 8*i* database, specifically if you are coming from Oracle 7.3 (Oracle Designer 1.3.2 was compliant with Oracle 7.3).

For example Oracle 8.0 has introduced the following new features:

- Partitioned Tables
- Partitioned Indexes
- Oracle Object Types
- Object Tables
- Object Views
- Oracle Collection Types
- Nested Tables
- Deferrable constraints.

You could already define the above new database features in Oracle Designer 2.1.x /6.0.

Oracle 8*i* (aka release 8.1.6 or above) has introduced for example - on top of Oracle 8.0 - the following (sub-set of) new features:

- Index-only Tables
- function-based indexes
- Java in the database
- advanced queuing
- compute statistics option for indexes
- deterministic clause for PL/SQL functions.

Note that advanced queuing was already available in Oracle 8 (Enterprise Edition). However, you could not document queue definitions directly in Oracle Designer 2.1/6.0, let alone generate code to populate the queues.

All these new features are available somewhere in new database objects (e.g. object views) and/or in the properties of existing database objects in the Oracle Repository, either on the primary level (e.g. Table properties) or on the secondary level (column properties). You should be aware that these new features are not automatically enabled by the migration wizards of Oracle Designer 6.0 and Oracle9i Designer.

Your migration activities for database objects to Oracle 8*i* may be ranked in the following categories:

- Update the freshly migrated database objects by adding references and/or associations, change default values or set specific values.
- Add new relational and object database object definitions.
- Replace existing relational definitions by object definitions.

This part (Part 3) of the migration guide for database objects focuses specifically on the first category of post-migration changes. If your category of changes is more like the second and/or third category then at least the following sections from the Oracle 8*i* documentation provide valuable support:

- Oracle 8*i* Concepts, Part No. A76965-01
- Oracle 8*i* Application Developer's Guide - Object Relational Features, Part No. A76976-01
- Oracle 8*i* Data Warehouse Guide, Part No. A76994-01
- Oracle 8*i* Enterprise JavaBeans and CORBA Developer's guide, Part No. A81356-01
- Oracle 8*i* Java Developer's Guide, Part No. A81353-01.

For more Oracle 9*i* new object-relational-xml features, see the overall PDF index document index.pdf in the root directory of the Oracle 9*i* documentation.

Assuming that your database migration is likely of the first category (update the freshly migrated database objects by adding references and/or associations, change default values or set specific values), the following migration scenarios are possible in bringing your Designer-generated applications forward into Oracle9i Designer:

- Scenario 1. Migrate, Regenerate All, No Redesign
- Scenario 2. Migrate, Regenerate All, With Redesign

Note however that there are general database migration steps, steps you always have to apply despite your source Designer version. These generic steps are discussed in Chapter 3 "General database migration issues".

Scenario 1. Migrate, Regenerate All, No Redesign

In this scenario, you will regenerate all database definitions of your entire application from Oracle9i Designer The goal of this scenario is to be able to generate your application out of Oracle9i Designer and achieve the same results you got when generating out of your previous Designer release. No attempt is made to redesign your existing application to make use of new relational features available in Oracle9i Designer.

This scenario has the following characteristics:

- Scenario 1 is fast and requires minimal changes in the database definitions within your application's Oracle Designer repository
- Scenario 1 does not take advantage of any new relational features in Oracle9i Designer. It is merely a 'technical' upgrade.

This scenario is appropriate when:

- your application is already in production
- your application is stable, no major functional modifications are expected
- maintenance is limited to simple bug fixing
- your physical database definitions are 100% derived from - or 100% generated from - the Designer repository, or post-generation modifications are minor.

Scenario 2. Migrate, Regenerate All, With Redesign

In this scenario, you will regenerate all database definitions of your entire application from Oracle9i Designer. As you regenerate each database definition, you will make use of new relational features as appropriate.

The goal of this scenario is to take advantage of the new features available in Oracle9i Designer. As with Scenario 1, you want to be able to generate your database definitions and get the same database structure from your previous release of Designer. However, several new database features have been added to Designer to improve performance (e.g. function based indexes) or functionality (e.g. multiple database implementations).

This scenario has the following characteristics:

- Scenario 2 requires modifications in several database definitions, and is therefore more time-consuming
- Scenario 2 fully leverages the new database features in Oracle9i Designer.

This scenario is appropriate when:

- your application is still in development
- your application is in production, but major functional modifications are to be made, or expected
- your application requires modifications that can only be implemented using new functionality in Oracle9i Designer.

Note that all scenarios result in an actual usage of Oracle 9i and Oracle9i Designer and therefore you will be optimally served by Oracle Support on your Oracle tool stack.

Chapter 2 New Database Features

Depending on which Designer release you are coming from, many features of Oracle9i Designer may be new to you.

This chapter presents a brief overview of new features that are of particular interest when migrating your database design from previous Designer releases. It is by no means an exhaustive list of all new features, and it does not try to explain each new feature in detail. Rather, it introduces the relevant features and points you to where you can find more information in the Oracle9i Designer online help.

The terminology of the different database element types is varied among the different Designer releases. The following table gives an overview of a subset of the available database object types in Oracle Designer 1.3.2, Oracle Designer 2.1.x/6.0 and Oracle9i Designer.

Database objects since Oracle Designer 1.3.2

#	Oracle Designer 1.3.2	Oracle Designer 2.1.2/6.0	Oracle9i Designer
	TABLE	TABLE	TABLE DEF.
	VIEW	VIEW	VIEW DEF.
	SNAPSHOT	SNAPSHOT	MAT.VIEW DEF.
	SEQUENCE	SEQUENCE	SEQUENCE DEF.
	CLUSTER DEF.	CLUSTER DEF.	CLUSTER DEF.
	PL/SQL MODULE	PL/SQL DEF.	PL/SQL DEF.
	<NA>	OBJ. TYPE	OBJ. TYPE
	<NA>	COLL. TYPE	COLL. TYPE
	<NA>	OBJ. TABLE	OBJ. TABLE DEF.
	<NA>	OBJ. VIEW	OBJ. VIEW DEF.
	<NA>	<NA>	JAVA DEF.
	<NA>	<NA>	QUEUE DEF.
	<NA>	<NA>	QUEUE TABLE DEF.

Note that you could not define all properties of Object Tables and Object Views in Oracle Designer 2.1.x/6.0.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all four sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from Designer 1.3.2

The following new Oracle9i Designer database features are applicable if you are migrating from Designer 1.3.2:

- Design Editor
- Database implementation
- Obsoleteness of the create property
- Usage of the batch generation and batch design capture
- Database capture changes (previously known as "reverse engineering")
- Separate menu for generating administrative objects

Design Editor

The functionality provided by individual Designer Release 1 design level tools is now incorporated into a single tool called the Design Editor. Each of the Designer Release 1 tools maps onto a Design Editor component. It is essential that you understand how to use the Design Editor before beginning your migration.

Release 1.x tool	Release 2.x Design Editor components
RON	Design Editor Navigator The Design Editor Navigator is a key component within the Design Editor. It is similar to the RON, but contains only objects specific to the design phase. You can use drag and drop, instead of clicking on menu options or toolbar buttons, to perform a wide range of tasks. For example, you can create a server model diagram from one or more table definitions by dragging the tables from the Design Editor Navigator component onto the worksurface. In addition, the database definitions within the Design Editor Navigator are spread across three DE tabs: Server Model - contains all “pure” database definitions like table definition properties (alias) columns, constraints and triggers DB Admin - handles the database implementation properties of existing data database definitions like storage, tablespaces and access rights Distribution - handles typical database distribution aspects like nodes, replication and (public) database links

	Note that at the same time all DDL generators are removed from the RON. You have to use the DE to generate DDL scripts.
RON Property Palette	Design Editor Property Palettes and Property Dialog boxes When you create and edit Repository definitions in the Design Editor, you can use either Property Dialog boxes (new for Release 2) or Property Palettes. Property Dialog boxes are wizard-style elements that walk you through complex tasks. They are especially useful when creating whole new tables and columns as they walk you through all of the required tasks in order. They are also useful as a tool for learning Designer. Property Palettes provide you with a direct way of entering/editing information for all of the properties that exist for an object. Property palettes are the quickest way of setting properties for existing objects, because all properties are displayed in a single palette.
Server Model Diagrammer	Server Model Diagrams Server Model Diagrams are created by dragging one or more tables from the Design Editor Navigator to the worksurface.
Preferences Navigator	Preferences Palette Preference values are set using the Preference Palette within the Design Editor. Note that you can define Server Generator preferences only at application level.
Module Logic Navigator	Logic Editor You can directly launch the module logic navigator - using the right mouse button option Edit logic - for a PL/SQL function, package or procedure. As a result a separate text editor is launched to add or modify the PL/SQL code.
Module Structure Diagrammer	Module Network/ PL/SQL Composition viewer A module network viewer enables you to display module networks horizontally in the Design Editor Navigator. This allows you to clearly see the relationships between the PL/SQL modules in a network.

For more information about the Design Editor, see also the Design Editor online help topic “Features added in previous releases”.

Database implementation

Designer 2.1.x has introduced a significant change in the meta-model for database objects also known as database implementation. To fully understand the significance of this change, first a description of the meta model of database objects in Designer 1.3.2 will be given, followed by a description of the database meta-model of Designer 2.1.x and above.

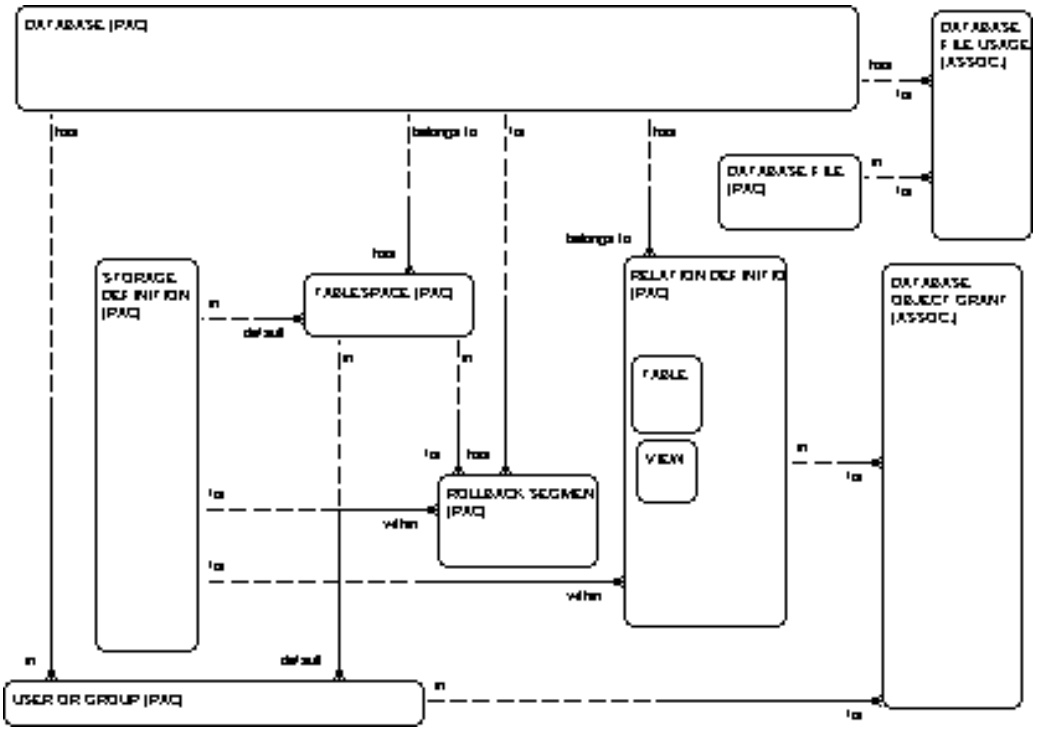
Database meta model of Oracle Designer 1.3.2

In Oracle Designer 1.3.2 - in the context of database implementation - the following elements were defined as Primary Access Elements:

- Database
- User
- Group (database roles)
- Relation definition (database implementation properties)
- Tablespace
- Rollback segment
- Storage definition.

At the same time the following elements were defined as Secondary Access Elements or association:

- Database objects grants for users
- Database objects grants for roles.



The above database elements and their category (PAC or SAC) are depicted in the figure below: Database meta model in Oracle Designer 1.3.2.

Database meta model since Oracle Designer 2.1.x

Since Oracle Designer 2.1x (and therefore also in Oracle9i Designer) only the following elements are defined as

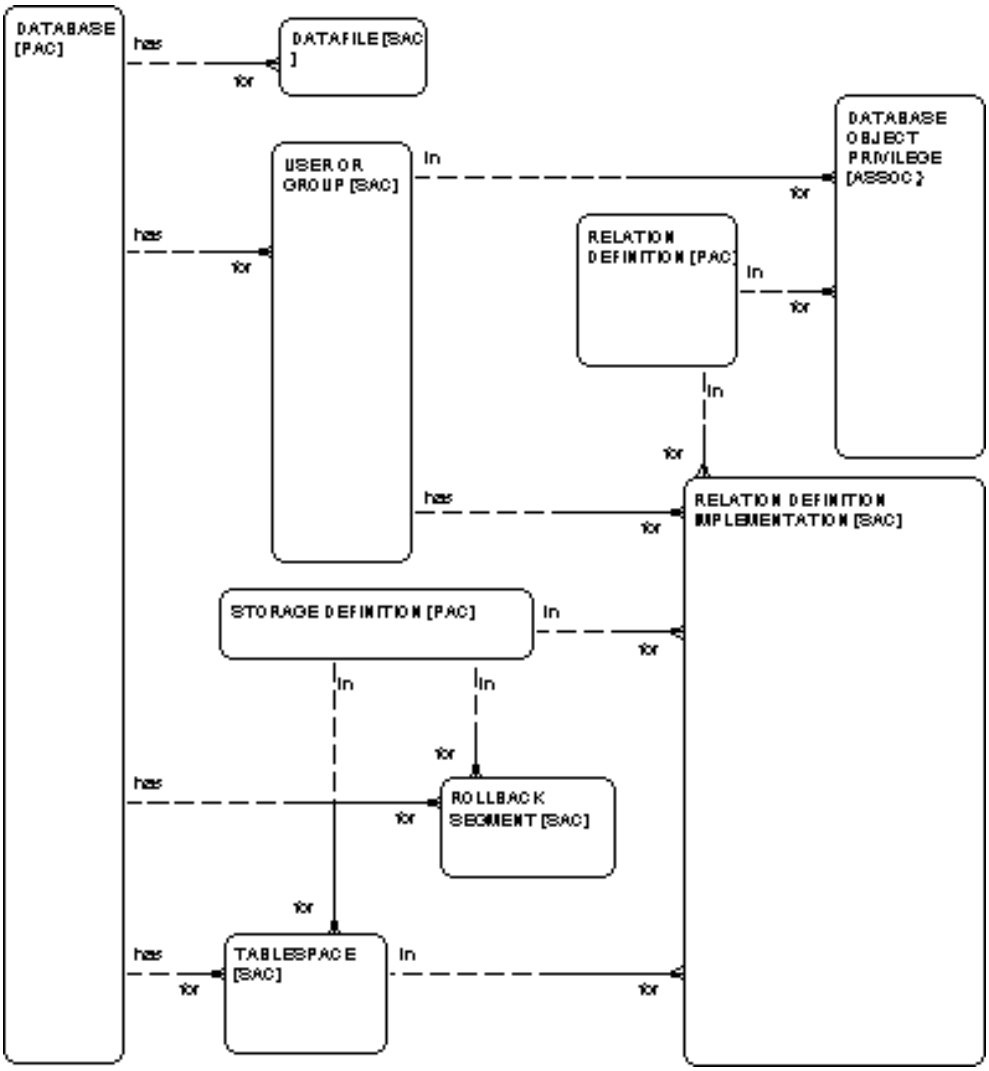
Primary Access Elements - within the context of database implementation:

- Database
- Storage definition.

At the same time, the following elements are defined as Secondary Access Elements or associations:

- User
- Group
- Relation definition implementation (database implementation properties)
- Database objects privileges for users or groups
- Tablespace
- Rollback segment.

The above database elements and their category (PAC or SAC) are depicted in the figure below: Database implementation meta model in Oracle9i Designer.



Note that this meta model change was introduced in Oracle Designer 2.1.

This significant meta model change (most database implementation elements and their properties are now defined in the context of a database) has the following consequences:

- Uniqueness. The above listed secondary access elements (and/or associations) are unique only within the context of a specific database. For example the user 'Scott' can exist within the context of database1 and

also within the context of database2.

- Check-out and check-in context. Any change to the database properties and/or to its secondary access elements or associations must be preceded with a check-out, if you have an enabled repository version of the database. The check-in/check-out context is extensive since there are so many secondary database implementation properties. For example a change in a relation definition implementation property like a storage clause must be preceded within a check-out of the owning database.
- Short-cuts (aka shares). You cannot short-cut (or share) a secondary access element, or association, separately. You can only short-cut primary access elements: the database and storage definitions. As a consequence, if you short-cut the database you also implicitly short-cut its secondary elements and associations without the possibility to reference these secondary elements individually.
- Post-migration database objects steps consequences. The migration wizard of Designer 2.1/6.0 will create specific migration database definitions like a R2_UPGRADE_DATABASE database and a database user R2_UPGRADE_USER. We will deal with these steps in Chapter 3 "General Migration Issues".

Note that the above mentioned consequences were also, to a certain extent, applicable to Oracle Designer 2.1/6.0. In these previous Oracle Designer releases you should replace check-in and check-out context with 'working context' and short-cuts with shares.

In Oracle9i Designer it is no longer necessary to share an element from another container (application system) first and then reference the shared element in the context container. You can reference an element from another container immediately as long as it is 'visible' in the same workarea. In addition you can make the referenced elements more 'visible' by creating short-cuts in the context container for these elements. Another advantage of creating short-cuts is that you "publish" the previously defined elements. It will most certainly stop you from creating a duplicate element.

Note also that the “migration wizard” of Designer 2.1.x and/or Designer 6.0 translates your data model to specific database implementations. These specific translations are described in detail in the next chapter.

Obsoleteness of the Create? property

The Create? property for each primary database object is no longer available, since Oracle Designer 2.1.x. For example you could use this property as a logical removal indicator. Instead of physically removing a specific database element you could set the create property to “No”. Similar behavior in Oracle9i Designer is available using database implementation (see also Chapter 2 “General database migration issues”).

Note that the Create? property on a secondary level like columns and constraints is transformed into the Complete? property. The migration wizard brings the Create? property value forward into the Complete? property value. Subsequently the Database Object Generator will not generate the syntax for secondary elements with a Complete? property set to 'No'.

Use of batch generation and batch design capture

Oracle Designer 2.1.x has introduced a batch utility for database object generation and design capturing that is available from the Tools menu (Tools > Batch Generate and Tools > Batch Design Capture).

These batch generators could be applicable in the following circumstances:

- To create a persistent set of database objects in a specific workarea stored in one or more gbu files.
- To generate DDL files for a persistent set of database objects on a different location and/or on a different point in time.

Batch generation for database objects comprises the following steps:

1. Choose your specific workarea in the Design Editor (File Menu; Change Workarea)
2. Launch the Batch Generate menu from the Tools menu.
3. Choose your container within a specific workarea.
4. Choose the group of objects you would like to generate for a specific session. The following groups within the context of database objects are available: Server Model (SRM); Database Administration (DBA); Module Component API (MAPI); Table API (TAPI); Reference Code (RF).
5. Enable/disable the runtime Database Object Generator options (if applicable) (see Chapter 2 "New Database features", section "Server Generator Options").
6. Highlight a specific user within the context of a specific database.
7. Select your database objects.
8. Save the generation details to a specific 'Database Generation Batch' file with for example the following naming standard <Workarea name>_<container>_SRM/DBA/MAPI/TAPI/RF_<database selection indication> .gbu, e.g. WA_DEVELOPMENT_QMS_SRM_SEL1.gbu. Note that the 'gbu' files only contain parameters, i.e. no database syntax.

   Repeat steps 4 to 8 until no more groups are necessary.
9. Collect all 'gbu' files.
10. Run dwzrun61 from the command line multiple times, for example from a different location, for each group using the specific 'gbu' file to generate all your different kinds of database objects.

You have to re-run the batch generator for database objects each time in the following circumstances:

- different Database Object Generator options
- different set of database objects.

Note that you do not have to re-run the batch generator (regenerate the gbu files) for database objects if one or more database object definitions are changed since it does not contain any syntax.

Database capture changes

Reverse engineering is called database capture since Designer 2.1.x. Database capture offers the following new functionality:

- General capture preferences and preferences for capturing specific database object types (e.g. views, PL/SQL definitions).
- Views and PL/SQL definitions can now be captured as declarative definitions instead of free-format text.

In addition, since Designer 6*i*, there is a set of General Capture preferences:

- [RECCOM] Capture Comments
- [RECGRT] Capture Grants
- [RECSYN] Capture Synonyms
- [DTAPIT] Capture TAPI Generated Triggers with Tables

Note that the design capture of database objects may need to be preceded with a check-out of matching objects in a versioned repository.

Separate menu for generating administrative objects

Oracle9i Designer has a separate tab (introduced in 2.1x) for generating administrative database objects (e.g. Databases, tablespaces). In Oracle Designer 1.3.2 part of these administrative database objects could explicitly (e.g. roles) and implicitly (e.g. database links) be generated from the DDL generator.

The Server Generator for database administrative elements supports - explicitly - the generation of the following administrative elements:

- Databases
- Database links
- Directories
- Profiles
- Replication Groups
- Roles
- Rollback Segments
- Tablespaces
- Users.

Migrating from Designer 2.1.2 and/or Designer 6.0

The following new Oracle9i Designer database features are applicable if you are migrating from Designer 2.1.x or above:

- Server Generator preferences - introduced in *6i*
- Database Generation notes and Server Generator generation tabs
- Overview of Table API changes
- Dependency Analysis - what happened to summary table usages? - introduced in 6*i*

Server Generator preferences - introduced in *6i*

Designer 6i introduced preferences for the Server Generator. These preferences are however only applicable at application level. You cannot set Server Generator preferences for one or more specific database objects.

The following general Server Generator preferences categories are applicable - available via the Server Generator Product Flavor:

- General
- Generation - General
- Generation -TAPI and view
- Design capture preferences for tables, views, materialized views, PL/SQL definitions, Oracle object types
- Reconcile report preferences

For a detailed overview of Server Generator preferences see the Preference Navigator and the online help.

Database Generation notes and Server Generator generation tabs

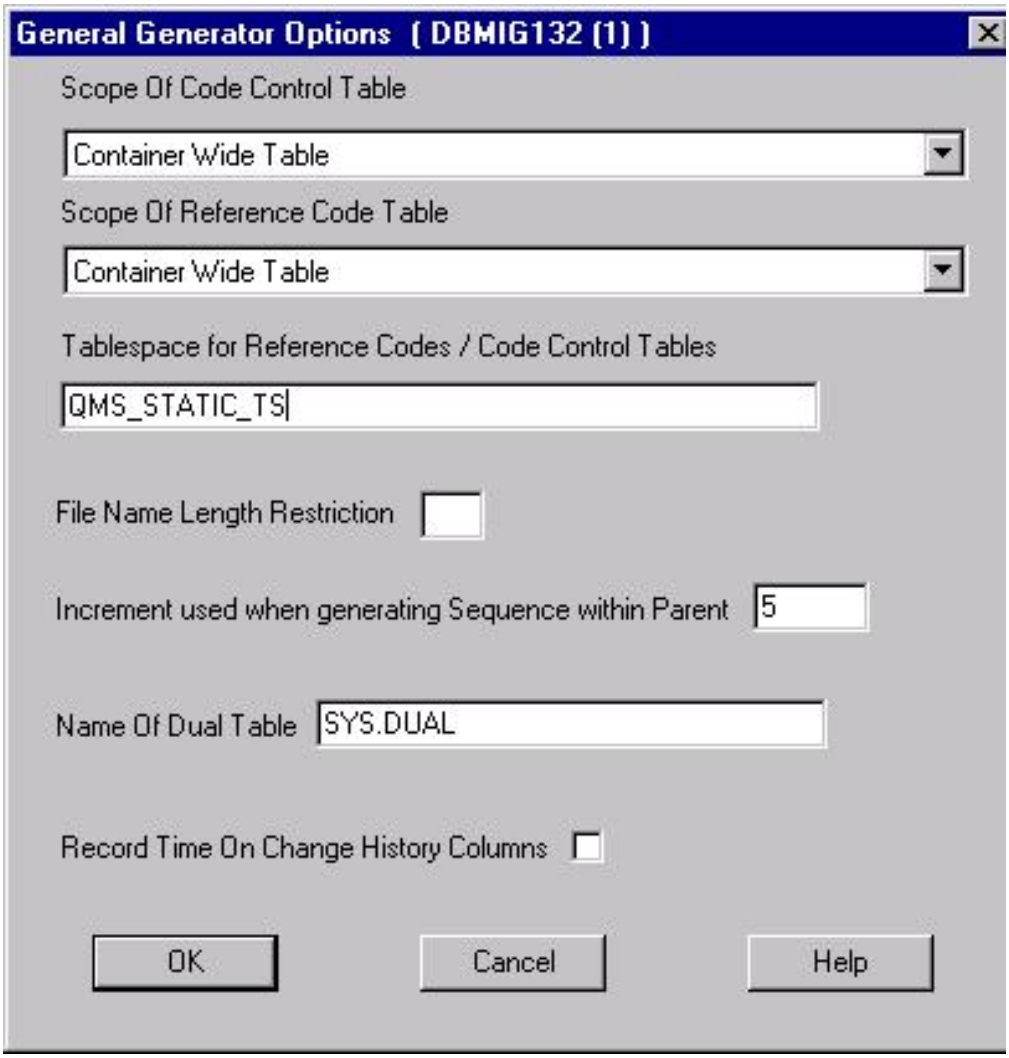
Database generation - implemented via a variety of “Tabbed” windows - is significantly changed since Designer 1.3.2 and some minor changes were introduced since Designer 2.1.x. This section discusses the following Database Generation subjects:

- General Generator Options
- Obsolete Database Generation options
- New or changed database object generation options

General Generator Options

Specific general Generator Options that previously were defined as preferences are now available as options accessible from the Design Editor.

You can launch the General Generator Options Windows (see below) from the menu Options > Generate Options > General in order to modify these general settings.



The fields on the above dialog are explained below.

Scope of Code Control Table

The Scope of Code Control and Scope of Reference Code are no longer preferences but settings that you can modify via the General Generator Options dialog (see above). The following scopes are valid for Code Controls:

- Container Wide Table
- Single Table
- Table for each Code Control (new).

The following scopes are valid for References Codes - no changes:

- Container Wide Table
- Single Table.

The 'create table' syntax for the Reference Code Table and the 'insert into ..' statements for the allowable values for reference codes can be generated via the Database Object Generation tab (see below).

Tablespace Of Reference Code Table

You can add to the 'create table' syntax for the Reference Code Table the preferred tablespace by adding here the tablespace name, e.g. QMS_STATIC_TS.

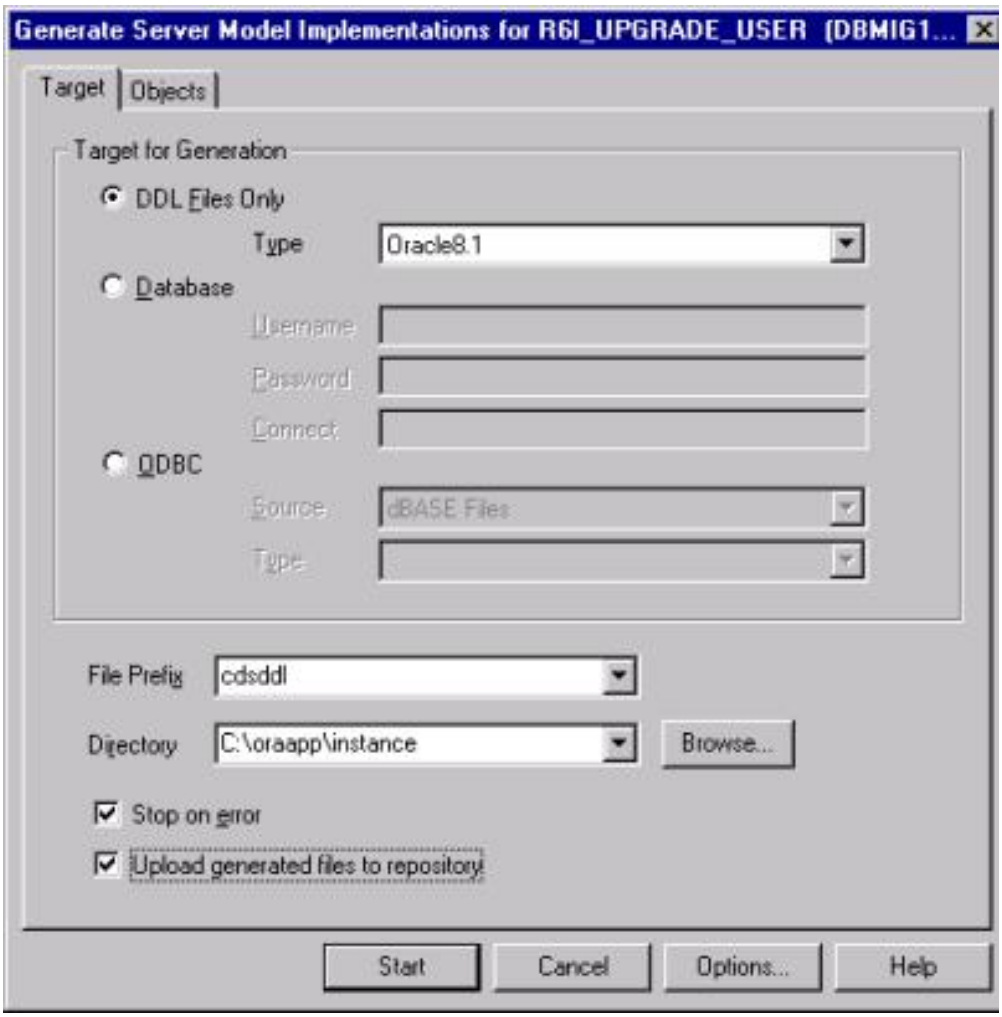
Obsolete options for Oracle9i Designer Database object generation options

The following Oracle Designer 1.3.2 DDL options are no longer available in Oracle9i Designer:

- Overwrite files. All files are automatically overwritten, there is no warning if the files already exist.
- Comment syntax. All comment in the DDL files starts by default with '—'.

Oracle9i Designer Database Object generation options (target tab)

This section will discuss the options on the target tab of the Generate Server Model Implementations dialog (opened from the Generate menu by selecting Generate > Generate Database from Server Model).



DDL files only option

The Database Object Generator will generate the 'full' DDL files if you enable the 'DDL files only' option. It ignores any existing database definition for a given schema.

Database option

If you enable the database option, then the Server Generator generates DDL scripts to create (or alter) database objects directly against a schema in an Oracle database, using a Net8 connection (using the username, password and connect string). Note that a separate reconcile option is no longer available (since Oracle Designer 2.1.x). You will receive a reconcile file called <**pre-fix**>.lis automatically that contains a full reconcile report between the Oracle9i Designer definition and the database for the given schema name.

In this mode the Database Object Generator will create specific DDL files based upon existing database definitions for the given schema name. For example if the table already exists but not a specific column, then an 'alter table add column …' statement is generated. If you would like the full 'create table …' definition you should choose the 'DDL files only' option.

Stop on error

Indicates whether DDL generation (and execution) should be terminated if an error is detected.

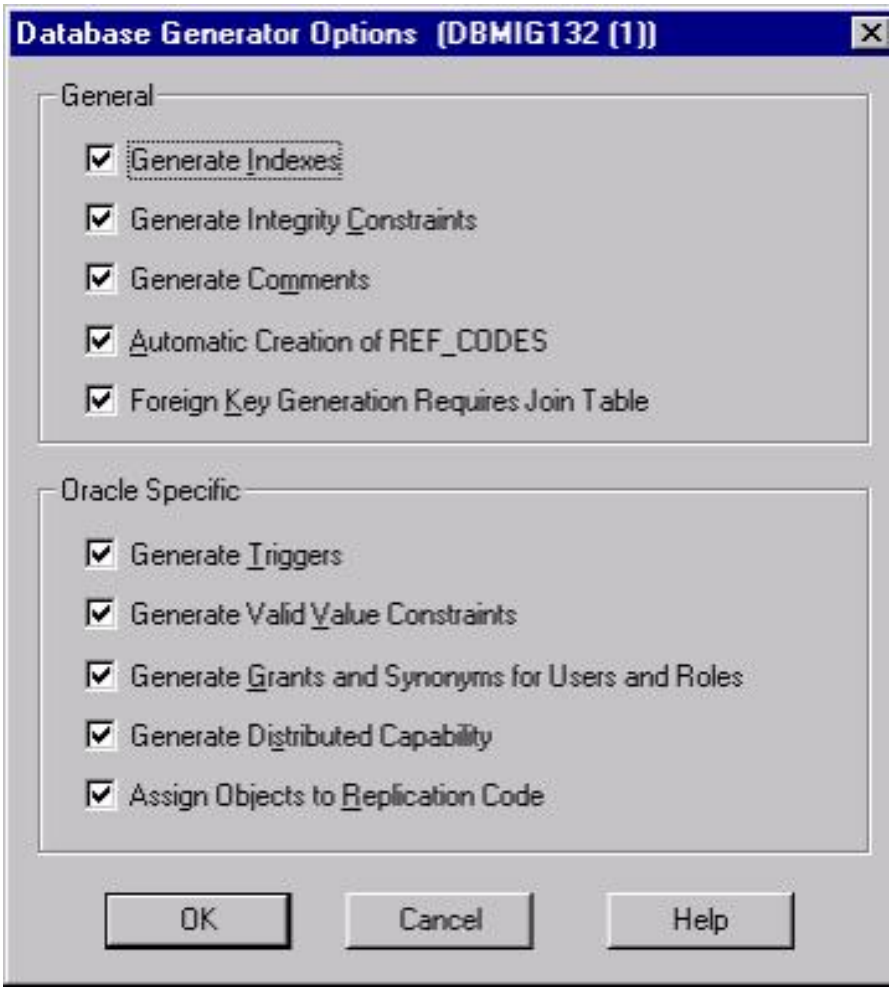
Upload generated files to repository

You can use this option only when the chosen directory - in the directory field - is mapped to the repository and the directory does **not** already contain downloaded DDL files that are read-only. Note that the files are not saved

in the repository if the directory is not mapped. You should careful consider the enabling of an automatic file upload after generation since you cannot control individual files and the DDL generator may stop because of existing read-only files.

New Oracle9i Designer Database Object generation options (Options button)

This section will discuss the fields on the Database Generator Options dialog, opened by clicking the Options button on the Generate Server Model Implementations dialog.



Generate Indexes

Indicates whether indexes defined against table/cluster/materialized view definitions are to be generated when the table, cluster or materialized view is generated. Note that you can no longer select specific indexes as in Oracle Designer 1.3.2.

Generate Integrity Constraints

Indicates whether integrity constraints defined against a table are to be generated when the table definition is generated, i.e. primary key constraints, unique key constraints, foreign key constraints and check constraints.

To prevent constraints from being generated when a table is generated, uncheck this option.

Automatic Creation of REF_CODES

Indicates whether reference code tables containing domain/column allowable values are to be generated automatically. A create table block in the '.tab' file for the reference codes and a separate insert script

(<pre-fix>.avt) with the reference code allowable values is generated if you enable the automatic creation of REF_CODES.

Foreign Key Generation Requires Join Table

Indicates whether one or both of the tables involved in a foreign key must be selected for it to be generated. Uncheck this option if you want all foreign keys definitions, independent of your context.

Note that if you previously relied on foreign keys always being generated when generating to file - despite your selection - you will need to uncheck this option.

Generate Triggers

Indicates whether triggers defined against table definitions are to be generated, when the table is generated.

Note that you can no longer select specific listed triggers as in Oracle Designer 1.3.2. You could however enable/disable specific triggers via the "Complete?" property of a trigger. Trigger syntax will not be generated if you disable the "Complete?" property.

Generate Grants and Synonyms for Users and Roles

Indicates whether any GRANT privileges and SYNONYMS defined for an object are to be generated to other users and roles. If a privilege has been granted for a user or role in another schema, a CREATE SYNONYM statement is generated for the object so that the object can be uniquely identified across different schemas. Note that you cannot select specific user grants and role grants as in Oracle Designer 1.3.2 from this Server Generator TAB. Note also that you can generate separately the 'create role syntax' and 'create user syntax' from the Generate Database Administration Objects utility.

Assign Objects to Replication Code

Indicates whether statements to associate database objects with a predefined replication group are to be generated.

If the target for generation is a database, the objects will be created in the replication group on the database. If the target for generation is a DDL script file, statements are generated to place the objects in the replication group.

Note that the replication group itself is generated using the Generate Database Administration Objects utility (see below).

Table API changes

The Table API creates a set of application-specific PL/SQL API packages that provide insert, update, delete and lock procedures for each application table. In addition it validates the data provided by the calling application and generates default values when appropriate before the table is modified or inserted. Note that a table API - with limited functionality - was already available in Oracle Designer 1.3.2 as part of the Webserver Generator.

The Table API in Oracle9i Designer covers the following functionality:

- validates constraints
  - validates arcs
  - validates allowable values in reference code tables
  - validates allowable values in domain tables
- auto-generates the following column values:

    - unique and sequential values for columns that derive their values from a sequence definition
    - pre-defined default values
    - derived values
    - change history information for AutoGen Type columns such as Created by, Modified by, etc.
- converts column values to uppercase
- maintains journaling information
- maintains denormalized columns

Note that the above overview does not distinguish between API functionality introduced in Designer 2.1 and 6.0 and that introduced in Designer 6*i*.

You can also add your own TABLE API logic before or after a specific DML operation. The Oracle9i Designer meta model captures the full event model of database logic - introduced in Designer 2.1.x. There are in addition to the table API table API triggers to complete server logic. You can generate the table API and table API triggers via the Generate table API menu.

Note that CDMruleframe (part of the Headstart Utilities) makes extensive usage of the TABLE API and the option to add specific application logic. For more information about CDMruleframe see www.otn.oracle.com.

Dependency Analysis or what happened to summary table usages?

Since Designer 6i a Dependency Manager tool has been introduced that allows you to store (additional) dependencies between structured elements and files and vice versa in a complete separate table structure - separate from the “normal” table structure for structured elements like tables, views, PL/SQL definitions, etc. Note that you also store dependencies between structured database elements only (like tables and pl/sql definitions) or dependencies between files only (like install scripts) and files that contain the syntax of a package. In general, dependency analysis gives you more control of your development and deployment environment by allowing you to efficiently manage your application development and release environment, thereby ensuring higher quality software.

You could ask yourself "Why do I need additional dependency information for database objects? They are already stored in a structured way, aren't they?". Yes they are, but not all dependencies are stored atomically. The table usage for PL/SQL definitions and the database object usage for files are examples of this.

As a consequence of the introduction of the Dependency Manager, summary table usages for PL/SQL modules (and other modules like Forms or Reports) are no longer available and you have to realize that the Oracle9i Designer migration wizard does not bring forward previously stored summary table usages as dependencies. For example, you have to rebuild the dependencies for packages - previously stored in the summary table usages - via the dependency manager. You will however receive much more dependency information than table usages only (e.g. procedure and function calls).

For more detailed information you can consult the online help for the Dependency Manager

Chapter 3 General Migration issues

There are a number of actions you must take regardless of which migration scenario you choose.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from 1.3.2

This section covers general database migration issues when migrating from Designer Release 1.3.2. If you are migrating from a later release, you may skip this section.

Multiple database implementations

Oracle Designer 2.1.x has introduced a distinction between database implementation independent properties and dependent properties of database objects - see also chapter 2 of this part (Part 3). All database implementation dependent properties are defined within the context of a database against a specific schema (e.g. the storage definition property of table implementation).

At the same time, all database implementation independent properties are - still - defined against the database objects (e.g. the alias property of a table).

This distinction between database implementation dependent and independent database properties allows you to define multiple implementations of the same database object. For example, you could introduce a schema in a specific database that represents a 'light' table implementation (test purposes) with associated tablespaces and (small) storage definitions. At the same time you could define another schema in another database that represents a 'production' table implementation with much larger tablespaces and storage definitions. Such a distinction was not available in Oracle Designer 1.3.2.

The migration wizard from Oracle Designer 2.1.x (or Oracle Designer 6.0) creates for each database object a database implementation. It creates a specific database (R2_UPGRADE_DATABASE) and a specific user (R2_UPGRADE_USER) during the migration process. The R2_UPGRADE_DATABASE will contain all existing users and a special user called R2_UPGRADE_USER. Subsequently the user R2_UPGRADE_USER within the existing database(s) contains one or more database objects and the associated database implementation properties if you have defined one or more database implementation properties in Oracle Designer 1.3.2.

We highly recommend that you reevaluate this migration result with respect to databases and database schemas. For example you could rename the migration database name R2_UPGRADE_DATABASE - or add another database - to a more meaningful name, reflecting the data collection within that database. Subsequently you could rename the upgrade user R2_UPGRADE_USER - or add another schema - within the database to a more meaningful schema name, reflecting the database elements owned by that schema.

These actions should take place at the 'DB admin' tab in the Design Editor. Subsequently you can create another database implementation - by creating a database schema in another database - if you need an additional database implementation for a specific database object.

It is however not necessary to create databases in Oracle9i Designer for each (promotion) database. The database implementation properties in each promotion database may not necessarily be different. For example the storage

clause for a specific table in the test database may be the same as in the acceptance database.

Note that you can still generate a 'skeleton' DDL script - a script without database implementation syntax - from the 'Server Model' tab in the Design Editor. Such a 'skeleton' script does not contain database implementation properties as tablespaces or storage clauses and may be useful for development purposes.

Obsoleteness of the Create? property

The Create? property for primary database objects (e.g. tables, views) has been removed while the create property at the secondary level (e.g. constraints, columns) is brought forward in the “Complete?” property. With the latter set to 'N' you circumvent the DDL creation of this secondary element.

You may adopt the following strategy for primary database objects to simulate a Designer 1.3.2 behavior with respect to the Create? property for primary database objects. First associate all database objects with a database schema or user within the context of a specific database. Secondly get a list of all database objects with the create property set to 'N' from your Designer 1.3.2 repository. Thirdly remove these database implementations from your database user. Subsequently generate only in the context of the DB admin tab - thus database implementations only. Note that this strategy has the disadvantage that your generated DDL will always contain implementation syntax like references to tablespaces and/or storage clauses.

Migrating from 2.1.2 or 6.0

This section covers general migration issues when migrating from Designer Release 2.1.2 or 6.0. These issues can also affect migrations from 1.3.2. The upgrade from these two releases is the same. If you are migrating from a later release, you may skip this section.

Server Generator preference PARSER and consequences for the generated syntax of PL/SQL definitions in combination with new pl/sql property Private Declaration

Oracle9i Designer comes with a set of server generator preferences - at application level only - as was already mentioned before. One of the Server Generator preferences - PARSER at the “Generation general” node - influences the generated syntax significantly by adding for example “BEGIN”, “END” and “DECLARE” strings in the PL/SQL code. The default value of this preference is set to "‘N" and this works fine for all free format PL/SQL definitions (packages, procedures, functions and triggers) - see also the section below about PL/SQL definitions and their storage methods. However if you have one or more PL/SQL definitions stored with the structured or non-free format method, then the PARSER Server Generator preference should be set to 'Y' and in addition you may have to move (parts of) PL/SQL code - specifically the declaration section - from the PL/SQL block property to the new Private Declaration property.

For example the correct trigger syntax is generated if you set the preference PARSER value to 'N' and if you move the following lines in the PL/SQL block property to the Private Declaration property:

```
l_rowid rowid := qms_rowid_queue.qms_get_rowid;
 l_empno qms_emp.empno%type;
  l_mgrno qms_emp.mgr%type;
  l_job qms_emp.job%type;
  cursor c_emp (p_rowid in rowid)    is
     select emp.empno
```

```
          ,emp.mgr
          ,emp.job
    from   qms_emp emp
    where  emp.rowid = p_rowid;
```

These lines together with the remainder in the PL/SQL block property is constructed by the DDL generator as:

```
PROMPT Creating Trigger 'QMS_EMP_AS'
CREATE OR REPLACE TRIGGER QMS_EMP_AS
 AFTER DELETE OR INSERT OR UPDATE
 ON QMS_EMP
DECLARE-- PL/SQL Specification
l_rowid rowid := qms_rowid_queue.qms_get_rowid;
  l_empno qms_emp.empno%type;
  l_mgrno qms_emp.mgr%type;
  l_job qms_emp.job%type;
  cursor c_emp (p_rowid in rowid)    is
     select emp.empno
          ,emp.mgr
          ,emp.job
     from   qms_emp emp
     where  emp.rowid = p_rowid;
-- PL/SQL Block
begin
  while l_rowid is not null    loop
    begin
      open c_emp(l_rowid);
      fetch c_emp
      into   l_empno
           ,l_mgrno
           ,l_job;
      close c_emp;
    end;
    if inserting or updating    then
       if not qms_has_job_mgr(l_mgrno)    then
          /* raise error stating    the employee is not a real manager */
          raise_application_error(-20000,'QMS-00049');
       end if;
    end if;
    if updating and l_job <>    'MANAGER' then
       if not qms_manage_emp(l_empno)    then
```

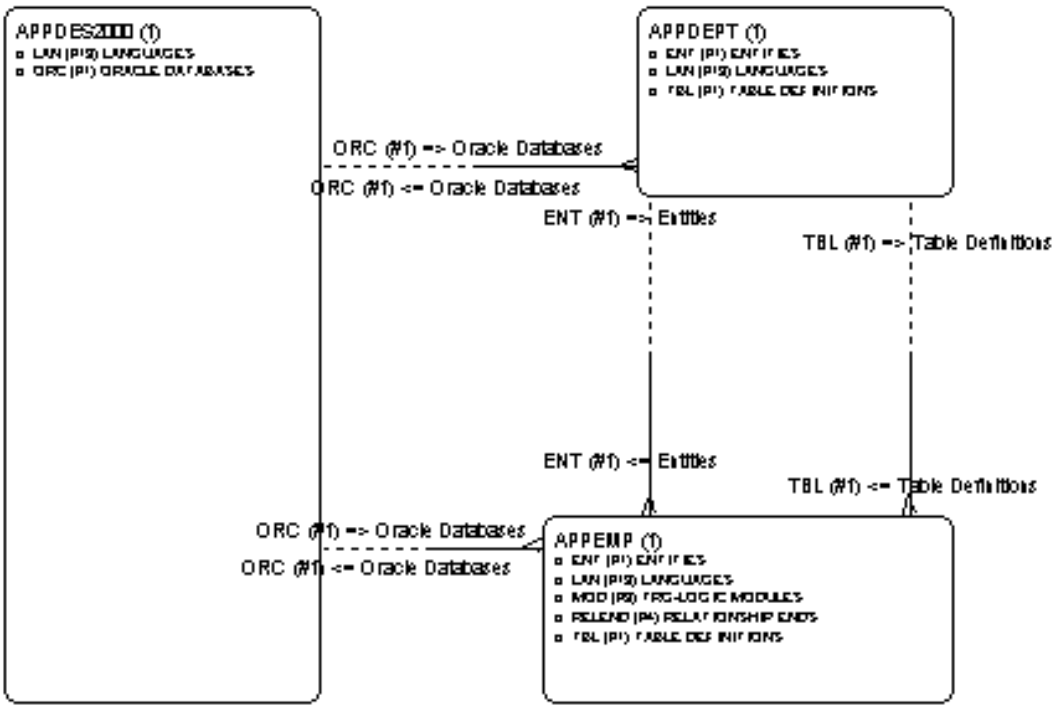
```
          /* raise error stating    the job cannot be changed because
             the employee still    manages some employee */
          raise_application_error(-20000,'QMS-00050');
       end if;
    end if;
    l_rowid:=qms_rowid_queue.qms_get_rowid;

  end loop;
  qms_rowid_queue.qms_clear_array;
exception
   when others then
     qms_rowid_queue.qms_clear_array;
     raise;
end;
/
```

Short-cut or reference strategy for database objects

The scope of short-cutting or referencing a database in Oracle9i Designer - previously known as sharing - has become much wider than sharing/referencing a database in Oracle Designer 1.3.2. Short-cutting a database in Oracle9i Designer not only implicates a single database short-cut but also a short-cut of all its secondary elements and associations.

In Oracle Designer 1.3.2 you could share a single database - captured in a single application system - with multiple other application systems indicating that all these application systems are implemented in a single database. Your generic share model could like this:



In the above example only one database is defined and shared out to the other application systems. At the same time the entity DEPARTMENT and the table DEPARTMENTS is shared with another application system. More specifically, application APPDEPT owns table DEPT and shares this table with APPEMP, that owns table EMP and receives the shared table DEPT. In addition table EMP has a foreign key to DEPT. Both applications receive a database share from application APPDES2000. This application provides only shares. The share model described above reflects that database objects from APPDEPT and APPDEPT are implemented in the same database. Moreover sharing DES2000 across these application systems was the only way to enforce the generation of the foreign key syntax for EMP with DEPT via the DDL generator since the foreign key definition is defined within the context of two applications. You could not generate the foreign key syntax if EMP and DEPT did not share the same database. This behavior is no longer implemented in Oracle9i Designer. A foreign key relation between table EMP and the shared table DEPT is enough to generate the foreign key syntax.

If you decide to keep this share model - or short-cut model in Oracle9i Designer - then all database implementation properties for database DES2000 are owned by application APPDES2000. If you want specific database implementation deviations for table EMP - e.g. another storage clause - then either you must create a local database in APPEMP or you must create another database in application APPDES2000.

You can keep the short-cut model described above if you exactly want to reflect the database situation in Oracle9i Designer and at the same time accept that all database implementation properties are owned by application APPDES2000.

You can circumvent the above situation by creating a local database in each application - each could have the same database name - and subsequently define the database implementations locally as opposed to centrally. You are however highly recommended to implement this specific database implementation model in Oracle Designer 1.3.2 before starting the migration steps as described in Part 2 of this migration guide. You could use Oracle Echo to reorganize your database properties effectively. Oracle Echo is a consultancy tool that you can use for reorganization purposes that comes with a consultancy service called 'repository reorganization services'. You can contact your local Consultancy Sales Representative for more information about this service and other Oracle Designer/Repository related services.

Storage method of PL/SQL definitions

PL/SQL definitions - introduced in Oracle Designer 2.1.x. and known as PL/SQL Modules in Designer 1.3.2 - can be stored in the repository in the following ways:

- Non free format - semi-structured. This method is automatically chosen if you capture one or more PL/SQL definitions from the database. The following PL/SQL PAC and SAC elements and their properties are used to store PL/SQL definitions semi-structured:
  - PL/SQL definition, property package specification: it contains only comment, the specification syntax is determined by the program units and their arguments
  - PL/SQL definition, property private declaration: it contains the private part of package body, function or procedure
  - PL/SQL definition, property PL/SQL block: it contains the PL/SQL text for functions and procedures
  - PL/SQL definition, property pragma restrictions: it is based upon pragma restrictions in package specification
  - Sub-program units: they are based upon functions and procedure sections in the package body
  - Arguments: they are based upon parameters in functions and procedures.

  Note that program data is not recognized separately, but populated in the private declaration section

and also that there is no separate property for the pragma 'autonomous transaction'. The pragma 'autonomous transaction' must be defined within the private declaration section.

- Non free format - fully structured. The PL/SQL definition is further broken down - manually - into the following elements - on top of the semi-structured method:
  - Sub-program units of type cursor
  - Program data - or local variables.
- Free format. The PL/SQL definition is stored unstructured in one or more text items. For example the package specification is stored in the text item PL/SQL specification and the package body in the text item PL/SQL block.
- In an uploaded file or files. As two separate files that contain the PL/SQL definition of the package specification (e.g. ECHO_UTIL.pks) and another file containing the package body (e.g. ECHO_UTIL.pkb) of the package ECHO_UTIL or as a single file containing specification and body.

The following table gives an overview of the advantages and disadvantages of the above mentioned PL/SQL definition storing methods:

Dimension	Semi-structured	a/d	Structured	a/d	Free format	a/d	As file(s)	a/d
Dependencies	Run Dependency Analyzer for additional usages (e.g. tables, views)	+	Run Dependency Analyzer for additional usages (e.g. tables, views)	+	All dependencies via Dependency Analyzer	-	All dependencies via Dependency Analyzer	-
Multi user	yes, multiple developers can build or change the package components	+	yes, multiple developers can build or change the package components	+	no, only one developer can build or change the package	-	no, only one developer can build or change the package	-
Reusability	Partly reusable components	+/-	e.g. Functions, Procedure, Cursors	+	No reusability	-	No reusability	-
Editor	via the Design Editor on - less multiple places	+/-	via the Design Editor on multiple places in different formats	-	via Design Editor either in text items PL/SQL block and/or specification	+	No usage of Design Editor. You do not have to edit the package on multiple places, but directly in a text editor	+
Productiveness	Little more productive in build phase	+	Probably less, changes must be defined declarative	-	Little more productive in build phase	+	Most productive in the build phase	+

CDMRule frame integration	Structured integration. You could include generated CDMrule PL/SQL definitions in text items and CDMrule could include custom PL/SQL definitions	+	Structured integration. You could call generated CDMrule PL/SQL definitions and CDMrule could call custom PL/SQL definitions	+	Non structured integration. You could include generated CDMrule PL/SQL definitions in text items and CDMrule could include custom PL/SQL definitions	-	Non structured integration. You could include generated CDMrule PL/SQL definitions in text items and CDMrule could include custom PL/SQL definitions	-
Compilation	Generates separate files. Compilation of body only is an option	+	Generates separate files. Compilation of body only is an option	+	Generates separate files. Compilation of body only is an option	+	Compilation of body only if necessary	+
Usage of Headstart utilities	Full usage of Headstart utilities	+	Full usage	+	Limited usage of Headstart utilities	-	No usage of Headstart utilities	-
Implementation properties (e.g. grants to users and/or roles)	Structured storage of implementation properties	+	Structured storage of implementation properties	+	Structured storage of implementation properties	+	Additional effort to store implementation properties like access rights	-
Usage of pragma autonomous transaction [see note below]	in Private Declaration text property	+	in Private Declaration text property		In one or more PL/SQL blocks in the package body	+	In one or more PL/SQL blocks in the package body	+
Migration effort	A move of PL/SQL blocks to private declaration property	-	A move of PL/SQL blocks to private declaration property	-	No migration effort	++	Capture in one or more files and subsequently uploaded	+

Note: The usage of the pragma autonomous transaction is very useful, for example for error handling. You can commit your log and error messages to a specific table independent of the state of your main transaction (failure or success).

As can be seen from the last row or dimension in the above table the migration effort will vary for each of the described storage methods for PL/SQL definition. There is almost no migration effort involved if your PL/SQL

definitions were stored as free format, while most effort is involved with the structured or semi structured format.

In previous Designer releases there was a strong tendency to use the structured or semi-structured format - the latter is still the default during design capture. With the introduction of the Dependency Manager there is less need for (semi)-structured format. The impact analysis based on files is almost as rich as the (semi)-structured format. Note however that the file format does not support the structured storage (and generation of grant scripts) of access rights against PL/SQL definitions .

Usage of the dependency manager to bring forward the summary table usages

The summary table usages are replaced by the output of the Dependency Manager as can also be seen from the table presented above - in the row dependencies. You therefore have to parse all dependencies for all database objects - including PL/SQL definitions - after a migration to Oracle9i Designer - to retrieve similar functionality. Note that you can parse dependencies for an entire container and its content or even for an entire workarea.

Chapter 4 Scenario 1: Migrate, Regenerate All, No Redesign

In this scenario, you will regenerate your database objects from Oracle9i Designer.

The goal of this scenario is to be able to generate your database objects out of Oracle9i Designer and achieve the same results you got when generating out of your previous Designer release. No attempt is made to redesign your existing application to make use of new features available in Oracle9i Designer.

This chapter assumes that you already have performed all the actions against the database objects described in Chapter 3 "General Migration Issues".

Migrating from 1.3.2

This section covers migrating from Designer Release 1.3.2. If you are migrating from a later release, you may skip this section.

There are no known specific migration issues arising from Designer 1.3.2. - provided that you have applied the steps described in the sections "Migrating from 1.3.2" and "Migrating from 2.1.2 or 6.0" in the General Migration Issues chapter of this part.

Migrating from Designer 2.1 or 6.0

First perform all the steps described in the section "Migrating from 2.1.2 or 6.0" in the General Migration Issues chapter of this part.

Secondly evaluate the following identified migration issues with respect to database objects coming from Designer 2.1 or 6.0.

Free format View DDL creation errors

The DDL of free format defined views - and materialized free format views - may result in syntax errors like a missing "from" clause. You can correct the view definition - and therefore ultimately the DDL syntax - by reevaluating the view select text property. Next to the “select” lines, add the “from” and the “where” clause.

Different handling of quotes in the column default property value

The behavior of the DDL generator with respect to default column values is changed when quotes are used around the default property value,e.g. ‘Y’, ‘N’. If you want to keep the literal string (including the quotes) in the generated DDL you have to set the next column property Default value type to “literal”. The DDL generator generates the following syntax:

… ,JOB VARCHAR2(9) DEFAULT "'P'" NOT NULL

…

If you do **not** want the quotes you have to set the next column property Default value type to “Database function call”.

The DDL generator generates then the following syntax:

…

,JOB VARCHAR2(9) DEFAULT 'P' NOT NULL

…

Note that migration wizard set this “Default value type” property default to literal.

Differences in names for valid values constraints

The database object generator shows a different behavior in generating valid values constraints. It generates for each valid value constraint a separate check constraint - this behavior was introduced in Designer 2.1.2. - and it generates a unique constraint name for each DDL session - introduced in Designer 6*i*. A valid value constraint for example can be enforced via a domain with allowable values.

This new behavior will most likely result in a recreation of the valid value constraints each time you generate DDL syntax against an existing schema that already contains all or part of the database objects (provided obviously that these database objects contain valid values). Note that this DDL behavior is persistent despite any changes in the valid values.

There are two ways to circumvent this behavior and as a result get more control over these kinds of constraints:

1. Create custom explicit check constraints in the repository with the same check syntax as the generated constraint. Note that you then have to revaluate these custom constraints each time the valid values are altered. Note also that you have to disable the following database object generation option: “Generate valid value constraints”.

2. Use the CDM Ruleframe framework to propagate these valid value constraints as separate CDM ruleframe business rules. You also have to disable the “Generate valid value constraints” database object generation option. You can find more information about the CDM ruleframe framework for enforcing business rules on OTN (www.otn.oracle.com)

Chapter 5 Migrate, Regenerate All with Redesign

In this scenario, you will regenerate your relational and object type database objects from Oracle9i Designer. You will use the new Oracle 9i database features provided by Oracle9i Designer. As you regenerate each database object, you will make use of new features as appropriate.

The goal of this scenario is to take advantage of the new database features available in Oracle9i Designer. As with Scenario 1, you want to be able to generate your application and get the same database layer you got from your previous release of Designer. However, many new features have been added to Designer to make more use of Oracle9i enhancements.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all the sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Note that the enhancement or redesign steps in this Chapter must be preceded or accompanied by the actions in Chapter 3 and Chapter 4.

You should also note that most of the new features (e.g. partition key entries, table partitions) may not be visible for existing table implementations - for a specific migrated database. These implementations are introduced in Oracle8 or later and your database 'Oracle Version' property is probably still set to **Oracle7**. However you can make these specific properties and/or entries visible by changing the 'Oracle Version' database property to **Oracle 8*i*** or **Oracle 9*i*.**

Migrating from 1.3.2

This section covers migration from Designer Release 1.3.2. If you are migrating from a later release, you may skip this section.

The following new Oracle9*i* subjects - already introduced in Oracle 8 - are discussed:

- Table and index partitioning
- Bitmap and reverse indexes
- Global and local indexes
- New database triggers properties
- New column properties
- New view database implementation dependent and independent properties
- New materialized view database implementation dependent and independent properties
- Deferrable constraints
- Scope properties
- New grant properties to roles or users

Note that almost all the above listed improvements do not require any structural change and can therefore be classified as cost effective.

Table Partitioning

Partitioning physically divides your table in horizontal pieces on different locations to boost performance, making use of multiple reads/writes. It is dependent on your frequently used access paths which columns are candidates for partition keys. For example suppose table EMP of Oracle (50,000 employees!) is frequently accessed on LAND_OF_ORIGINATION then that column is a fine partition key candidate. In general you should try to use columns that hold static values since the row is stored initially on a specific physical location. You should try to avoid a table reorganization as a result of an uneven distribution of rows among the partitions. You can find the following secondary table partitioning properties at the implementation level - provided that your database version is set to Oracle8 or higher:

- Table partitions. Tables can be decomposed into smaller pieces called partitions. Partitions are particularly useful where tables have grown so big that they have become difficult to manage. Each table partition definition that is recorded in the repository represents a partition into which a table is to be divided on a database. You can influence the physical storage of the partitions via the “Value Less Than” property. For example appropriate values for this property could be (A-F, G-M, etc.) for the starting alphanumeric character of the LAND_OF_ORIGINATION column in the Oracle EMP example.
- Partitioning key entries. A partition key entry defines the usage of a particular column in the partitioning key of a table or an index. Each partitioning key can be based on one column only. Partition key entries are stored against table implementations or index storages in the repository. They can be created for global index storage definitions, but not local index storage definitions.

The migration wizard does not add a Partition Key Entry nor does it define table partitions for a specific table implementation.

Bitmap indexes

Bitmap indexes are widely used in data warehousing applications, which have large amounts of data and ad hoc queries but a low level of concurrent transactions. For such applications, bitmap indexing provides:

- Reduced response time for large classes of ad hoc queries
- A substantial reduction of space usage compared to other indexing techniques
- Dramatic performance gains even on hardware with a relatively small number of CPUs or small amount of memory
- Very efficient maintenance during parallel DML and loads

Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of space because the indexes can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table. Bitmap indexes can substantially improve performance of queries with the following characteristics:

- The WHERE clause contains multiple predicates on low- or medium-cardinality columns.
- The individual predicates on these low- or medium-cardinality columns select a large number of rows.
- Bitmap indexes have been created on some or all of these low- or medium-cardinality columns.
- The tables being queried contain many rows.

You could transform any non-bitmap (and non-unique) index that fulfills the above criteria into a bitmap index by changing the index type into a bitmap.

User Object Index Storage property: Index Type - at database implementation level

Indicates whether the storage details are for a local or global index. An Oracle7 index is equivalent to an Oracle8 global index

A B-tree index on a partitioned table can be local or global. Global indexes must be fully rebuilt after a direct load, which can be very costly when loading a relatively small number of rows into a large table. For this reason, it is strongly recommended that indexes on partitioned tables should be defined as local indexes unless there is a well-justified performance requirement for a global index. Bitmap indexes on partitioned tables are always local. See also "Table Partitioning", above, for further details.

This property does not receive any value through the migration wizard.

User Object Index Storage property: Reverse or Nosort? - at database implementation level

Indicates whether Reverse or Nosort options are to be used for this index.

One cause of sorting is the creation of indexes. Creating an index for a table involves sorting all rows in the table based on the values of the indexed columns. Oracle also allows you to create indexes without sorting. If the rows in the table are loaded in ascending order, then you can create the index faster without sorting.

Presorting your data and loading it in order may not always be the fastest way to load a table. Specifically, if you have a single-CPU computer, you should if possible sort your data before loading. Then create the index with the NOSORT clause.

Creating a REVERSE key index, compared to a standard index, reverses the bytes of each column indexed (except the rowid) while keeping the column order. Such an arrangement can help avoid performance degradation in an Oracle Parallel Server environment where modifications to the index are concentrated on a small set of leaf blocks. By reversing the keys of the index, the insertions become distributed across all leaf keys in the index.

This property does not receive any value through the migration wizard.

Global Index Partitions as a secondary element of User Object Index Storage - at database implementation level

A global index partition represents a partition into which an index has been divided. The index partition can have different physical characteristics to the index; for example, it can be stored in a different tablespace and can have its own storage parameters.

The partitioning of a global index is completely independent of the partitioning of its associated table. A global index is partitioned based on its own specific partition key, which is not necessarily the same as the partition key of the table.

In a global partitioned index, the keys in a particular index partition may refer to rows stored in more than one underlying table partition or sub-partition. A global index can only be range-partitioned, but it can be defined on any type of partitioned table.

A global index is created by specifying the GLOBAL attribute.

A global partitioned index contains a single B-tree with entries for all rows in all partitions. Each index partition may contain keys that refer to many different partitions or sub-partitions in the table. See also Chapter 11 "Partitioned Tables and Indexes" in Oracle 8*i* Concepts.

The migration wizard does not add a Global Index Partition for each table implementation.

Local Index Partitions as a secondary element of User Object Index Storage - at database implementation level

A local index partition represents a partition into which an index has been divided. The index partition can have different physical characteristics to the index, for example, it can be stored in a different tablespace and can have its own storage parameters.

Each local index partition has exactly the same number of partitions (with exactly the same "VALUE LESS THAN" boundaries) as the number of partitions that have been defined for the table. The tablespace and the storage parameters for a local index partition do not have to be the same as those used for the table partitions.

In a **local** index, all keys in a particular index partition refer only to rows stored in a single underlying table partition. A local index is created by specifying the LOCAL attribute.

Oracle constructs the local index so that it is equipartitioned with the underlying table. Oracle partitions the index on the same columns as the underlying table, creates the same number of partitions or subpartitions, and gives them the same partition bounds as corresponding partitions of the underlying table.

Local indexes have the following advantages:

- Only one index partition needs to be rebuilt when a maintenance operation other than SPLIT PARTITION or ADD PARTITION is performed on an underlying table partition.
- The duration of a partition maintenance operation remains proportional to partition size if the partitioned table has only local indexes.
- Local indexes support partition independence.
- Local indexes support smooth roll-out of old data and roll-in of new data in historical tables.

See also Chapter 11 "Partitioned Tables and Indexes", in Oracle 8*i* Concepts.

The migration wizard does not add a Local Index Partition for each table implementation.

Database Trigger property: Fire When Propagated?

Indicates whether or not the trigger is to fire when the data has been propagated, for example, in a multi-master replication environment or with views (instead of triggers). Here the trigger code would perform the validation of the rows being modified and, if valid, propagate the changes to the underlying tables.

New column properties

Def Template/Library Object

The name of an object in a template or object library. This property will be populated within the context of a module - like a display property - each time it is referenced in a module and subsequently used by the Oracle Forms Generator. You could for example define a trigger on the double mouse-click behind this library object.

Enable this column property by referencing an item from your object library.

Server Defaulted?

Indicates whether a default value should be supplied by application logic residing on the database server if no value is supplied by a client side application. This tells the generator it should requery the row after insert/update to display the server derived data. The migration wizard sets the value of this property to 'N'.

Server Derived?

Indicates whether the column's value is derived on the database server, via a database trigger, and therefore should not be set or provided by the client side. The table API will generate server API code for an auto-generated column (e.g. Date Created, User Created) if this property is enabled. The migration wizard sets the value of this property to 'N'.

Derivation Expression Type

Indicates the type of derivation expression (e.g. null, function call, SQL expression) that is used to specify the column value. Obviously you have to provide a specific value for this property if you have defined a derivation expression. The migration wizard does not set a specific value.

Where/Validation type

Indicates whether the Where/Validation condition property defines a Function Call or a SQL Expression. Obviously you have to provide a specific value for this property if you have defined a where validation expression. The migration wizard does not set a specific value

View properties

Object type view property

Identifies the Oracle object type on which the object view is to be based. For an existing object view definition, this property can be changed to reference a different object type. However you cannot change an object view into a relational view by removing the object type reference from this property. You cannot reference an object type from an existing relational view. Use the utility 'Create Oracle Object Type' to migrate existing relational views definitions. See also the section "Oracle Object Types".

Non-free format View property: Optimizer Hint Clause - also applicable for Materialized Views

A hint clause to be used where the view is defined declaratively. The optimizer uses the hint to choose an execution plan for the SQL statement. You should add an optimizer hint clause if you specifically want to use the rule-based optimizer. The default behavior of a database is the usage of the cost-base optimizer based upon the computed statistics. The migration wizard does not set a specific value.

Base Table Locations: new database implementation secondary element for views

Create a base table location if the underlying table, view or materialized view is implemented in a different schema. This is because the Database Object Generator must be told which underlying objects it is to use when creating a view or materialized view on a database.

For example, if view B is based on table A, then you can use a base table location if the latter is to be implemented for a different user.

By default, views and materialized views are implemented in the same schema as the element on which they are based. Where this is the case, base table locations are not required

Materialized view implementation independent properties

Updateable

Indicates whether the materialized view can be updated, i.e., whether INSERT, UPDATE and DELETE statements can be performed on the materialized view. Only simple materialized views can be updated.

A materialized view is always synchronized with its source via a materialized view refresh (or a refresh of the surrounding refresh group). You may consider making or allowing these changes to be made to the data in a materialized view:

- intermediate (part) synchronization of the materialized view on specific DML operations on the source
- direct specific changes on the materialized view (for example in a replicated environment).

The following default migration value is applicable: 'No'.

Cluster

The name of the cluster in which the materialized view exists. It is used to place a materialized view on a cluster if the same grouping of columns are frequently selected from a table or tables.

Materialized view implementation dependent properties

The following materialized view implementation dependent properties must be reevaluated:

- Materialised view log properties. Materialized view log properties for a table implementation
- Cached. Specifies whether the block retrieved for this materialized view is placed at the most recently used end of the Last Recently Used (LRU) list. When set to Not Cached, retrieved blocks are placed at the least recently used end of the LRU. Placing the data at the most recently used end can improve performance. It is useful to set this property to Cached for small lookup tables
- Materialized View Group. The name of a set of materialized views on the same database (but not necessarily in the same schema on that database) that are to be refreshed at the same time and at the same interval. The definition specified for Materialized View Group is the definition that would apply to a Refresh Group. A Materialized View Group is the counterpart of a Master Replication Group.
- Build Type. Identifies whether the materialized view is to be populated immediately after executing the DDL, or whether it is to be deferred.
- Query Rewrite? Identifies whether the materialized view is eligible to be used for query rewrite. Valid values: Null, Disable, Enabled

Deferrable constraints

You can defer constraints for validity until the end of the transaction. A constraint is deferred if the system checks that it is satisfied only on commit. If a deferred constraint is violated, then commit causes the transaction to roll back.

If a constraint is immediate (not deferred), then it is checked at the end of each statement. If it is violated, the statement is rolled back immediately.

If a constraint causes an action (for example, delete cascade), that action is always taken as part of the statement that caused it, whether the constraint is deferred or immediate.

You can enable the defer status of a constraint by applying one of the following values for the constraint “Defer Status” property:

- Initially deferred. The constraint is deferrable and, by default, checked at the end of the transaction.
- Initially immediate. The constraint is deferrable and, by default, checked at the end of each DML statement.

The Migration Wizard brings the Not Deferred status value forward.

If you are using CDMrule frame or are planning to use it you should only use non-deferrable constraints. For more information about CDMrule frame see www.otn.oracle.com.

Scope properties (global synonym name and scope) for each database implementation object

Each implementation of a database object (e.g. table, view, sequence) has received the following additional scope properties:

- Global Synonym Name. The name to be used by the Database Object Generator when creating synonyms for this object. Default Migration value: null
- Scope. The scope of visibility of the database object (Database or World). This is used by the Database Object Generator when database links and synonyms need to be created. Default Migration value: Database

New or changed Granted to Users or Roles properties

Create Synonym?

Indicates whether a synonym is automatically created by the Database Object Generator when the database object is in a different schema to the user or role. This property receives by default a 'No' string from the migration wizard.

Execute?

Indicates that the EXECUTE privilege is granted to the user or role. Not applicable for tables. This property receives by default a 'No' string from the migration wizard.

Read?

Indicates that the READ privilege is granted to the user or role.

The Read privilege provides secured access to the files stored in the operating system directory to which the directory object serves as a pointer.

The directory object contains the full pathname of the operating system directory where the files reside. Because the files are actually stored outside the database, Oracle server processes also need to have appropriate file permissions on the file system server. Granting object privileges on the directory database object to individual database users, rather than on the operating system, allows Oracle to enforce security during file operations. This property receives by default a 'No' string from the migration wizard.

Enqueu?

Indicates that the ENQUEUE privilege is granted to the user or role. This privilege applies only to queue implementations. This property receives by default a 'No' string from the migration wizard.

Dequeu?

Indicates that the DEQUEUE privilege is granted to the user or role. This privilege applies only to queue implementations. This property receives by default a 'No' string from the migration wizard.

Migrating from Designer 2.1 or 6.0

This section covers some database migration issues when migrating from Designer Release 6.0.

The following new Oracle9i new database features - already introduced in Oracle 8*i* - will be discussed:

- Index tables only
- Function based indexes
- Compute statistics for indexes
- Domain key constraints
- Deterministic clauses for PL/SQL definitions
- Object types
- Java definitions

Note that the introduction of object types and Java definitions involves a substantial structural change and are therefore less cost effective than the other Oracle9i new features.

Index table only

Typically large tables with only a small number of columns like intersection tables are candidates for index-organized only.

You can change a “normal” table to an index organized table by enabling the table property Index-organized?. Note that the migration wizard does not enable this specific table property.

Pct Theshold - at database implementation level

Used in the case of an 'index only table' to specify a percentage of the block size. This property does not receive any value from the migration wizard.

Overflow Tablespace- at database implementation level

Used in the case of an 'index only table' to specify the name of the tablespace that is used for overflow purposes. This property does not receive any value from the migration wizard.

Function Based indexes

Function based indexes are introduced in Oracle8*i* and are a very cost effective means of boosting your application performance without changing the underlying database structure.

You can define function based indexes at the index entry level (e.g. column level) and define the function expression in the “Index Function” property like upper(ename). In addition you should set the “type” property to “function based” as opposed to “column based”.

Compute statistics for Indexes

Indicates whether statistics are to be collected on creation of the index. Typically you would set this value to 'yes' in the context of migration since all tables will contain data. Re-creation of the indexes will then automatically lead to the collection of statistics. Note that the migration wizard disables this property by default.

Domain Key Constraints

Oracle Designer 6i introduced a new SAC (secondary element) for tables called Domain Key Constraints.

Domain key constraints allow you to model domains using a table other than the predefined one normally used by Oracle Designer.

A domain key constraint allows a generated application to access a specific range of domain values, from a table containing multiple domains. The domain key constraint specifies that values entered in the domain key columns do not conflict with a predefined range of acceptable values.

You may want to do this for several reasons, e.g., you may already have a domain table that you would like to design-capture and continue to use, or you may want more control over how domain values are shown within a model.

You add a domain key constraint for a specific table as a new secondary constraint type next to the ‘normal’ constraints like foreign key or check constraints.

Primary Key, Unique Key or Foreign Key column property: Conversion Format Mask?

The format to be used when a date or number column is joined to a VARCHAR2 column in a domain table. If a format is not specified here, the application generators use a set of rules to define the format. Note that this property is not used by the Database Object Generator. It applies only to key components of domain key constraints

Foreign Key column property: Second Join Column?

The name of a column in the join table that is the derivation for all or part of this foreign key.

Usage of the Deterministic? clause for PL/SQL functions

Allows the system to use a saved copy of the function's return result - Functions only - if such a copy is available. The saved copy could come from a materialized view, a function based index, or a redundant call to the same function in the same SQL statement.

The Query optimizer can choose whether to use a saved copy or re-call the function. The function should reliably return the same result value whenever it is called with the same values for its arguments. Therefore do not define the function to use package variables or to access the database in any way that might effect the function's return result, because the results of doing so will not be captured if the system chooses not to call the function

You can simply make use of the deterministic feature by enabling the Deterministic? property for PL/SQL

functions.

Object Types

Oracle object types (aka user-defined types) provide a way of creating user-defined datatypes for the Oracle8*i* database server. These datatypes can be used in addition to the built-in datatypes provided by the server.

You could use Oracle object types:

- to define a datatype of a column in a relational table
- to create a table based on a specific Oracle object type, i.e. an object table
- as part of the definition of another Oracle object type, i.e. as an attribute of the new Oracle object type.

Oracle object types can be considered as templates for creating instances of objects. Typically, the objects have a set of common attributes and methods (operations), and may be structured in a hierarchy.

Use the Create Oracle Object Type and Create Object View utilities to migrate a server model based on a relational table design to an object design.

There are three key stages to migrating a relational design:

- Create Oracle object types from the relational tables/views
- Modify the default Oracle object type definitions
- Create object views for the Oracle object types

Stage 1 - Create Oracle object types from the relational tables/views

The first stage is to build Oracle object type definitions based on the structure of the relational table/views in your current server model.

The utility - first item on the Utility menu in the Design Editor - creates an Oracle object type for each selected table/view. It also creates an Oracle object type attribute for each non-foreign key column and a REF attribute for each foreign key. Note that for REF attributes to be correctly mapped to Oracle object types, both tables/views referenced by the foreign key need to be selected.

Note that the utility also creates a set of mapping elements to record the relationship between the relational table/columns and the Oracle object type/attributes. This mapping is only visible in the RON (see below). This mapping information is used when the Oracle object types are used to implement object views for the relational table/views (see Stage 3).

Stage 2 - Modify the default Oracle object types

Once you have created a set of Oracle object types based on your relational tables/views, you can review your design from an object-orientated perspective. In this stage you will typically introduce further Oracle8 database server features.

For example:

- Migrate a set of attributes to new embedded Oracle object type
- Change REF attributes to collections of VARRAY or nested table
- Change attributes to REF attributes
- Add/delete attributes (and column mappings).

Stage 3 - Create object views for the Oracle object types

Finally, you can build object view definitions based on the Oracle object type definitions and their mappings to relational table/views.

The utility - second item on the Utility menu in the Design Editor - creates an object view for each Oracle object type you select, together with the required SQL statements to construct the object rows from the underlying relational tables. It also creates OIDs from the underlying relational tables primary/unique key.

There are no new properties for the primary and secondary implementation independent properties of Oracle Object Types and there are no specific implementation dependent properties for Oracle Objects Types.

Collection types

Oracle collection types are used to define datatypes. They provide support for collections of similar items, and can be implemented as either of the following:

- Nested tables - These are useful where referential integrity is required and are suited to master-detail and one-to-many relationships. Nested tables can have attributes but no methods.
- VARRAYs - These are useful for providing quick access to small or uniform-sized collections of objects in a table. VARRAYs do not have methods or attributes.

An example of when you could use a collection type is for managing orders and their associated order lines. The orders would be stored in a top-level outer table, while the order lines could be stored in a nested table or VARRAY. The nested table or VARRAY is recorded as a column (on a relational table) or an attribute (on an object relational table) on the outer table. This column/attribute embeds the collection type within the outer table and is designed to hold the order lines for each order.

Typically nested tables are used when there is no limit on the potential number of items to be stored. Conversely, VARRAYs always have a maximum number of elements, that is defined by the user.

Nested tables and VARRAYs cannot be referenced directly, but they can be referenced indirectly via the outer table.

Object Tables

Object tables are based on Oracle object types. The objects that are stored in each row of the table are uniquely identified by a system-generated identifier, called an object identifier. This object identifier is maintained automatically by the database.

Object Views

Object views allow you to retrieve, update, insert and delete relational data as if they were stored as objects. This allows you to use the object oriented features of the Oracle 8*i* database server with existing relational data. If you want to build an object view that is based on an existing relational table or view, you can use the Create ObjectView utility (see above).

Transformation Mapping Sets (visible in RON only)

A transformation mapping set represents the overall 'collective' mapping for transformation from one level to another (for example, from the type model to the server model). It is made up of the individual mappings of pairs of specified elements.

There are three transformation types:

- Entitity Object to Relational BC4J mapping
- Entity Object to Relational Mapping
- Relational to Object Relational mapping

The last mapping type is created automatically by the utilities 'Create Oracle Object Type' and/or 'Create Object View'.

Java definitions

Oracle Designer 6i introduced the option to store Java Definitions in a structured way. It supports the following Java Definitions:

- Source Definitions
- Class Definitions
- Resource Definitions

If you use this structured method for Java Definitions in Oracle9i Designer you will not be able to use the JDeveloper 9*i* IE optimally. However you could also store the Java Definitions as text files in the Oracle9i Designer repository. Adopting the file approach you will use the Oracle9i Designer repository basically as a Source Control Tool to manage the Java files, its versions, its releases and its dependencies. In addition, the Oracle9i Designer repository will control the revision data and will keep track of a change history and the lock of a file when it is in use by a developer.

The following subjects for storing Java Definitions as files will be handled:

- Organization of Oracle9i Designer repository for storing Java files
- Interactions between the Oracle9i Designer / Repository 9i and JDeveloper 9i
- Dependency analysis for Java files

Organization of Oracle9i Designer repository for storing Java files

Oracle9i Designer will be used as the repository to manage all Java files and its versions. Within Oracle9i Designer, files will be stored in a folder structure that matches the Java package structure. Thus, a Java package - package names preferably in lowercase! - will also be visible in the Oracle9i Designer environment.

Interactions between the Oracle9i Designer / Repository 9i and JDeveloper 9i

A significant amount of effort is spent in the integration between the Repository 9i and JDeveloper 9i based on the assumption that all JDeveloper 9i components (e.g. Java files, XML files) are stored as files in the repository, rather than structured objects. This tight integration - together with the Oracle9i Designer capability of storing structured database objects - makes Oracle9i Designer/Repository 9i/JDeveloper 9i an effective and productive development and deployment tool stack.

Dependency analysis for Java files

You could enrich the repository with dependency information for Java files by analyzing your Java files via the Dependency Manager for dependencies with other files and structured objects (e.g. table usage). See also the section about dependency analysis.


Prev Next


Contents

Chapter 1 Introduction

This migration guide provides the information necessary for upgrading Forms Applications that were designed and generated using earlier releases of Designer to Oracle9i Designer.

The document discusses migration from the following earlier releases:

- 1.3.2
- 2.1.2
- 6.0

This document assumes that you have already installed Oracle9i Designer and migrated your repository. (See the instructions in Part 2 of this Migration Guide.) The document then explains steps that you have to take so that you can:

- Generate your application from Oracle9i Designer and achieve the same generated results you had from earlier releases, and
- Take advantage of new features that have been added to Designer since your previous release.

Throughout the document, special mention is made of any migration issues known at the time of publication of this document.

There are a number of migration scenarios that are possible in bringing your Designer generated applications forward into Oracle9i Designer.

Scenario 1. Migrate, Regenerate All, No Redesign

In this scenario, you will regenerate your entire application from Oracle9i Designer, including all forms, libraries, menus and reports. However, you will continue to use your existing template forms and object libraries.

The goal of this scenario is to be able to generate your application out of Oracle9i Designer and achieve the same results you got when generating out of your previous Designer release. No attempt is made to redesign your existing application to make use of new features available in Oracle9i Designer.

This scenario has the following characteristics:

- It is fast and requires minimal changes in the application definition in the Oracle Designer repository
- It uses your existing templates and libraries, upgraded to Forms 9*i*. Any customizations made to the templates and libraries are preserved.
- It does not take advantage of any new features in Oracle9i Designer. It is merely a 'technical' upgrade.

This scenario is appropriate when:

- Your application is already in production.
- Your application is stable, no major functional modifications are expected.
- Maintenance is limited to simple bug fixing.
- Your application is 100% generated, or post-generation modifications are minor.

Scenario 2. Migrate, Regenerate All, With Redesign

In this scenario, you will regenerate your entire application from Oracle9i Designer, including all forms, libraries, menus and reports. As you regenerate each module, you will make use of new features as appropriate.

The goal of this scenario is to take advantage of the new features available in Oracle9i Designer. As with Scenario 1, you want to be able to generate your application and get the same user interface you got from your previous release of Designer. However, many

new features have been added to Designer to make achieving the desired result easier. Many features that were difficult or impossible to generate with earlier releases of Designer are now supported. Thus, in one pass, you can eliminate post generation modifications and difficult constructs that were used only to work around limitations of earlier releases of Designer.

This scenario has the following characteristics:

- Any customizations made to the templates need to be applied to the Oracle9i Designer versions.
- It requires modifications in many module definitions, and is therefore more time-consuming
- It fully leverages the new features in Oracle9i Designer.

This scenario is appropriate when:

- Your application is still in development.
- Your application is in production, but major functional modifications are to be made, or are expected.
- Your application requires modifications that can only be implemented using the new functionality in Designer.
- Your application has been heavily modified post-generation, but the majority of the modifications are no longer needed due to the new functionality in Designer.

Scenario 3. Migrate, Regenerate Incrementally

This is the most complex scenario. In this scenario, you will migrate your application a little at a time, rather than all at once. You will begin by upgrading all of your forms, libraries, menus and reports to Forms 9i. You will then make the changes required to run forms generated from your previous release of Designer alongside forms generated from Oracle9i Designer. Finally, over some arbitrarily long period of time, you will regenerate all of your modules out of Oracle9i Designer.

The goal of this scenario is to allow you to regenerate your whole application, taking into account new features, but in such a way that you do not have to migrate your entire application in one go. This means you will be able to move the deployed application to the new tool stack before you have completely migrated every form. Thus, you can continue with bug fixes and new development in parallel with the continuing migration effort.

This scenario has the following characteristics:

- It requires you to do work to allow old and new generated forms to run side by side. This is extra work that is not required for Scenarios 1 and 2, and that will eventually be discarded. Thus, the total effort required is greater.
- It allows you to perform a phased migration. You can take advantage of the new Designer features right away, without the need to regenerate all your forms at once.

This scenario is appropriate when:

- Your application is in production, but major functional modifications are to be made, or are expected.
- Your application requires modifications which can only be implemented using new functionality in Designer
- Your application is too large to migrate in one ‘big bang'.

Scenario 4. Forms Migration Only

The first three scenarios all eventually require you to regenerate your application. Any post-generation modifications will be lost. If you heavily modified your application post-generation, and the characteristics of Scenario 1 apply to your situation, you might consider only upgrading the runtime environment to Forms 9*i*, and not upgrading to Oracle9i Designer. This implies that all future maintenance has to be done manually in Developer.

This part of the migration guide does not cover a 'Forms only' migration. For information on migrating Forms to 9i, see the Oracle Technology Network at http://otn.oracle.com/products/forms and select ‘Migration’ under ‘Oracle Forms Technical Information’.

Note that, even though you may choose not to use Oracle9i Designer for continued form generation, you may still use Oracle9i Designer to maintain your database definitions. You may also choose to use the Software Configuration Management features of Oracle9i Designer to manage your application source code (.fmb, .mmb, .pll and .rdf files).

For information on Software Configuration Management with Oracle9i Designer, see the Oracle Technology Network at http://otn.oracle.com/products/repository .

Scenario 5. Design Capture

If you heavily modified your application post-generation, and the characteristics of Scenario 2 apply, you may want to consider using the Design Capture features of Oracle9i Designer.

This part of the migration guide does not cover Design Capture. For information on Design Capture, see the Oracle Designer online help.

Chapter 2 Oracle9i Designer New Features

Depending on which Designer release you are coming from, many features of Oracle9i Designer may be new to you.

This chapter presents a brief overview of new features that are of particular interest when migrating a generated forms application. It is by no means an exhaustive list of all new features, and it does not try to explain each new feature in detail. Rather, it introduces the relevant features and points you to where you can find more information in the Oracle9i Designer online help.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all the sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from Designer 1.3.2

This section describes new features added in release 2.1.2. If you are already at release 2.1.2 or higher, skip this section.

Design Editor

The functionality provided by individual Release 1 design level tools is now incorporated into a single tool called the Design Editor. Each of the Release 1 tools maps onto a Design Editor component. It is essential that you understand how to use the Design Editor before beginning your migration.

Release 1.x tool	Release 2.x Design Editor components
RON	Design Editor Navigator The Design Editor Navigator is a key component within the Design Editor. It is similar to the RON, but contains only objects specific to the design phase. You can use drag and drop, instead of clicking on menu options or toolbar buttons, to perform a wide range of tasks. For example, you can create a module diagram from a module definition by dragging the module from the Design Editor Navigator component onto the worksurface. You can also drag existing Repository definitions from the Navigator onto the diagrams. Most of the tasks that can be performed using other Design Editor components can also be performed from the Design Editor Navigator.

RON Property Palette	Design Editor Property Palettes and Property Dialog boxes When you create and edit repository definitions in the Design Editor, you can use either Property Dialog boxes (new for Release 2) or Property Palettes. Property Dialog boxes are wizard-style elements that walk you through complex tasks. They are especially useful when creating whole new modules or module components as they walk you through all of the required tasks in order. They are also useful as a tool for learning Designer. Property Palettes provide you with a direct way of entering/editing information for all of the properties that exist for an object. Property palettes are the quickest way of setting properties for existing objects, because all properties are displayed in a single palette.
Module Diagrammer	Module Diagrams Module Diagrams are created by dragging a module from the Design Editor Navigator to the worksurface.
Preferences Navigator	Preferences Palette Preference values are set using the Preference Palette within the Design Editor. Some behaviors that used to be governed by preferences (for example Item Group Orientation) are now properties in the associated object's property palette. Many new preferences have been added.
Module Logic Navigator	Logic Editor In previous releases you could define server-side procedural logic by recording PL/SQL functions, packages and procedures. In this release you can also record 'application logic' which corresponds to Form Level, Block Level and Item Level Triggers in Forms.
Module Structure Diagrammer	Module Network/ PL/SQL Composition viewer A module network viewer enables you to display module networks horizontally in the Design Editor Navigator. This allows you to clearly see the relationships between the modules in a network.

Module Components

The structure of a module changed after Designer 1.3.2. Modules are now broken down into module components. Each module component consists of one base table usage and zero or more lookup table usages.

Module components can also contain so-called unbound items that are items that are not part of the base or lookup tables. This eliminates the need for secondary column usages.

Module components can also be designated as 're-usable'. This means that a single module component can be included in multiple modules.

Form Logic in Designer

One of the most important changes in Designer is the ability to record form level, block level and item level triggers in Designer. Each module, module component and item has a node for adding application logic. This can consist of one or more pl/sql code segments. Designer lets you intersperse your custom code segments before, between, after and instead of code segments that Designer itself will generate.

Preferences and the Object Library

Form Builder has introduced the new concept of the object library. An object library is the collection of forms objects into a file that uses the .olb suffix. Objects from the object library can be subclassed into forms. Thus, the object is a sort of super template. Designer allows you to use the object library for subclassing into generated forms. Designer comes with a pre-defined object library that contains a number of objects recognized and used by Form Generator. You can also extend this object library to add your own objects.

Designer still uses (and provides) a template form, but this template form now works very closely with the object library.

Designer also provides a utility for building an object library from a maintenance form. This form has a name like [designer home]\bin\fm2libxx.exe where xx is the major Designer release number. There is also a .txt document that explains how to use the utility.

See the Form Builder online help for information about how to use the object library.

The object library can now govern many of the settings and behaviors that used to be governed by preferences. Thus, many of the old preferences are now obsolete. See the Design Editor online help about preferences for information on which preferences are now obsolete.

TAPI (some logic can go in the server instead of in the form)

Designer now allows you to generate a table API that acts as a server-side wrapper for your tables to enforce business logic. Essentially, this boils down to one pl/sql package per table. The PL/SQL package is stored in the database. Each table has database triggers that invoke the TAPI.

Logic that used to be placed in your form can now be placed in the TAPI. For example:

- Populating a primary key from a sequence
- Populating audit columns
- Populating default values
- Validating dynamic domains
- Simple business rule validation

Generated forms use the new ‘Returning’ clause to refresh data generated by the TAPI into the form record buffer.

See the Designer online documentation for more details about the TAPI.

PL/SQL Libraries in Designer

Library generation is a major new feature in this release. Features of the new Library Generator are:

- Generation of library modules (.pll files) from repository library module definitions (either as a separate operation or during generation of a module to which it is attached).
- Design capture of existing library modules into the repository.

Reports

The ***module component***, a new Repository object, provides a container for the tables, columns, and derived items that Report Generator uses to generate a query. A module component contains one base table usage. It also may contain one or more lookup table usages and two new types of table usages:

- subquery table usages
- single-row SQL aggregate table usages.

In this release, groups are generated from the ***Item Groups*** that are defined in the module component. In addition, some of the properties that control group generation are now item group properties. For example, you can define a layout style for each break item group.

In this release, Report Generator provides a simpler interface for defining break groups in Group Left (Break) reports. To define the "break" column or columns, create a new break item group and add the break columns to it.

To define the column values that you want to display in a generated report, you define a new type of repository object, a ***bound item***.

You can now add Oracle Developer Report Builder ***report triggers*** and ***named routines*** directly to Oracle Designer repository objects.

This release includes a new type of repository object for defining navigation between report modules or between a report module and another type of module: the ***Navigation Action Item***.

You can now generate a matrix report from a single module component. In previous releases, you could only generate a matrix report from three base table usages (module components): one for each of the matrix groups.

There is now a distinction between two types of summaries:

- Computed summaries, which are calculated on the client
- SQL Aggregate summaries, which are calculated on the server

There is now a distinction between three types of derived items:

- SQL Expressions, which are evaluated on the server
- Client Side Functions, which are evaluated on the client
- Server Side Functions, which are evaluated on the server

Reports on the Web

In the previous release of Report Generator, you generated a report for Web output by attaching the CGWEBP.RDF (PDF) or CGWEBH.RDF (HTML) template to the report module. This release of Report Generator includes a new preference, DSPFMT, for specifying the type of report output.

In this release, you can generate HTMLCSS output to support cascading style sheets.

In the previous release of Report Generator, the CGWEB.pll library contained PL/SQL code for generating Web functionality such as hypertext links and tags. The library was attached to the default template report. In this release, Report Generator generates calls to Report Builder SRW built-in procedures instead of calls to procedures in the CGWEB.pll library. The CGWEB.pll library is no longer necessary and is therefore no longer attached to the default template report. However, CGWEB.pll is still supplied with this release of Report Generator to support generation of report modules that have been upgraded from the previous release. To attach CGWEB.pll to reports generated using this release of Report Generator either set MODLIB to CGWEB.pll or use Report Builder to attach CGWEB.pll to the template report.

Report Queries

In this release, Report Generator does not automatically generate a parameter that specifies the START WITH clause for treewalk links. To specify the START WITH clause and other treewalk link properties, use the table usage repository properties.

This release includes a new type of table usage: the subquery table usage. The subquery table usage enables you to explicitly define a subquery. In previous releases, you created subqueries by defining a table usage and hiding all the items (detailed column usages) in that table usage. In this release, you simply select the parent table usage and define a new table usage, specifying "Subquery" as the type. Use the WHERE clause property of the subquery table usage to define the restriction that you want the subquery to apply. This release also provides a more direct way of adding the NOT operator to reverse the logic of the query. Instead of setting a preference (ANNOTR), reverse the logic by setting the 'Not Exist?' Repository table usage property.

The new preference DETLNK controls how Report Generator creates the link between master and detail rows and enables you to use correlated subqueries to increase network efficiency.

You can now retrieve all rows from two or more tables by defining a SQL query set. When you define a SQL query set, Report Generator adds the UNION operator to the generated query to return all rows from all SELECT statements.

In this release, Report Generator adds all bound items to the generated SELECT list. In previous releases, if you specified that a

detailed column usage should not be displayed on the generated report, Report Generator removed the column from the SELECT list. You do not need to define bound items for foreign key columns unless you want to generate a field for the foreign key column. Report Generator adds the foreign key column to the SELECT list when needed to join module components or table usages.

Report Parameter Forms

If you create parameters in the report module definition, the parameter form is created whenever you run the generated report. Report Generator no longer supports parameter form generation, parameter form templates, or parameter form layout preferences.

Domain validation of parameters is now controlled on the server. For more information, refer to 'About reference code tables' in the Server Generator on-line help.

Template Definition Files

This release of Report Generator supports template definition files (TDFs), which are new in Oracle Developer Report Builder 3.0. If you upgrade report modules from previous releases of Oracle Designer, Report Generator automatically converts RDF templates to TDF format.

The close integration between Report Generator and Report Builder templates also means that some of the customization via preference settings and post-generation modifications is now consolidated. For example, spacing between labels, fields, and groups can now be set using Template properties.

In the previous release of Report Generator you were able to add default boilerplate object keywords to your layout model template and apply formatting styles to these objects. This was to allow generated objects in the report to inherit the format that you applied to the boilerplate object. In this release, the font and visual attributes can be specified in the template definition file.

Removed boilerplate object keywords	How to customize in Release 2
CG$M1 and CG$M2	Combined into one object CG$MT
CG$BUTTON	Use the repository item properties width and height
CG$SIZING	Use the Report Builder with the Layout Model open
CG$DCU_GRP_FRM	Use the Template properties in the Oracle Designer Item Group Style/Title sections
CG$HEADER	Use the Template properties in the Frames section of the Layout Model Body
CG$PARAMETER	This functionality is not supported in this release
CG$PROMPT	Use the Template properties in the Field/Labels Headings section of the Layout Model Body
CG$FIELD	Use the Template properties in the Fields section of the Layout Model Body
CG$US	Use the Report Builder Layout Editor to create a field in the Report Builder template, and set the source to be a column that is of datatype char (e.g., DESNAME). Add the following Report built-in function call to the Format Trigger: srw.set_field_char (0, user)

Migrating from Designer 2.1.2

This section describes new features added for release 6.0. If you are already at release 6.0 or higher, skip this section.

The biggest change for Form Generation in moving from Designer 2.1.2 to Designer 6.0 was the change from Forms 5 to Forms 6. There were a few minor changes in addition to this:

- generation of bean area items containing Java Beans
- generation of custom (java) user interface components (which includes support for the Oracle Look and Feel)
- full screen menu generation no longer supported.

Migrating from Designer 6.0

This section describes new features added for release 6i.

LOV components

The new repository LOV element simplifies the process of LOV generation by decoupling LOVs from lookup table usages. This not only makes it easier to define LOVs, it also enables you to:

- define multiple LOVs for the same block
- reuse the same LOV in multiple blocks and multiple forms
- define LOVs for unbound items.

New Layout Features

This release includes a number of new layout features. Some of these new features can cause changes to the layout of your migrated applications.

There are a number of new layout styles available:

- support for the splitting of blocks across multiple canvases by generating multi-region blocks, and also the design capture of multi-region blocks
- support for the placing of blocks beside each other on the same canvas
- generation of navigator style forms (to provide a user interface similar to that of the Design Editor) and navigator style items.

You can now generate spreadtables onto tab canvases.

This release introduced ***relative tab stops*** (as distinct from *absolute tab stops* in previous releases) as a means to position and align items and item groups. Your migrated applications will continue to use the old absolute tab stops unless you specifically change them to use the new relative tab stops.

Enhancements have been added to layout functionality, including:

- Specification of real units when setting the decoration preferences.
- Extra line above decoration options for the preferences BLKDEC, CONDEC, GRPDEC, OFADEC, RADDEC, STBDEC.
- Using BLKSBP, you can position a block scrollbar to the right or left of the multi-record area within a block instead of to the right or left edge of the block itself.
- In multi-record blocks in which the lines do not wrap, Form Generator now positions a summary item below the item it summarizes (instead of repeating the summary item for each displayed row).
- If a module component's Width property was not explicitly set, Form Generator reduces the width of the generated block to optimize side-by-side block layout by removing any unused space to the right of the rightmost item.
- The new LAYFRA preference gives you the option to generate frames instead of graphic objects/text as decoration.
- Increased coverage for the preferences ITMPPE and ITMMPW to allow expansion of other types of generated text as well as item prompts.
- You can now display values from a lookup table in a combo box, as well as in a text list or a poplist.

Support for New Oracle Forms Features

Enhancements have been added to support native Oracle Forms features, including:

- Native Oracle Forms tooltip support for generated items using ITMTIP.
- Generation of Oracle Forms display items is now supported, along with a corresponding new standard source object called CGSO$DISPLAY_ITEM.
- Generation of Oracle Forms hierarchical tree items is now supported, along with a corresponding new standard source object

called CGSO$HTREE.

- Generation of mirror items from unbound items.
- Placing all generated code into a generated module library attached to the generated form rather than into the generated form itself using the new PGULIB preference.

Support for New Oracle8 Features

Enhancements have been added to support new Oracle8 features available in Oracle Forms, including:

- Use of the Oracle8 RETURNING clause when inserting or updating records, enabling the form to populate a base table item with the value returned from a derived column without having to requery the database.

Chapter 3 General Migration Issues

There are a number of actions you must take regardless of which migration scenario you choose.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from 1.3.2

This section covers general migration issues when migrating from Designer 1.3.2. If you are migrating from a later release, you may skip this section.

Add System Folder to Oracle9i Designer Workarea

If your migrated modules have '*******' in the Language property, it means that they are referencing the language from another folder, usually the System Folder. If this happens, modify your workarea to include the System Folder.

Generating Context-Sensitive HTML Help

When application systems are upgraded to Oracle9i Designer, the Help Context Id Prefix is nullified for all Repository objects. The ID's are cleared because Oracle9i Designer uses a different format for the help IDs (see bug 656392).

If you have context-sensitive online help, **you must generate the Help file(s) before you start generating the forms!** By generating the Help file(s), new Context IDs are generated.

Module Implementation Name

In earlier releases, Form Generator derived the name of generated source and executable files from the module's Short Name property and ignored a module's Implementation Name property. MS Help Generator did use the module's Implementation Name property and required that its value was six characters or less.

It is therefore quite likely that a module developed using Oracle Designer 1.3.2 has its Short Name property and Implementation Name property set to two different values.

In Oracle9i Designer, Form Generator uses a module's Implementation Name property to derive the name of generated source and executable files. The (HTML) Help Generator continues to use a module's Implementation Name property, but the six-character limit has been removed.

This may be an issue where:

- You have added code in one module to call another module (i.e. if user code or template code in a generated form calls another generated executable).
- The called module's Implementation Name and Short Name properties are different.

If you generate the called module in Oracle9i Designer, the call will fail because the called executable will have a different name to

that expected in the code.

- Change the Implementation Name property of every form and menu module to the value of the module's Short Name property.

Where/Validation Condition on Lookup Table Usages

In version 1.3, the where/validation condition on Lookup Table Usages could be used to restrict the allowable foreign key values. The where validation condition entered was applied in three places:

- in the LOV record group query to restrict the records displayed in the LOV
- in the WHEN-VALIDATE-ITEM trigger on the foreign key column(s), to make sure the restriction was also applied when the value was entered manually without using the LOV
- in the POST-QUERY (or POST-CHANGE in case of a hidden FK column) trigger to apply the restriction to *existing* rows

However, using the same where clause for the POST-QUERY places a limitation on the use of foreign keys. With this schema, you cannot create a relationship where all lookup rows are valid on existing records, but only some subset of the lookups can be used when creating a new record.

To address this issue, many developers specified sophisticated conditions in the WHERE/Validation Condition property.

In Oracle9i Designer, the WHERE/Validation condition property has been replaced with two properties to handle the two different operations:

- The Lookup validation WHERE Clause property is used to restrict the value in the LOV and validate values entered directly into a field.
- The Where Clause of Query property is used to restrict the existing records returned by querying the block.

On upgrade, both the 'Lookup validation Where Clause' property and the 'Where Clause of Query' property are populated with the value of the Designer 1.3.2 'Where/Validation Condition' property. If you have included sophisticated conditions in the Where/Validation Condition property of lookup table usages to avoid inappropriate constraint violation messages, these are unlikely to be suitable for the new Where Clause of Query property.

You will probably need to clear the ‘Where Clause of Query’ property on lookup table usages after upgrading. You may also need to clear it on base table usages, though only if it is related to the old LOV (i.e. the base table usage may also have its own valid where clause that you don't want to delete).

In addition to splitting the 'Where/Validation Clause', the implementation of the 'Where Clause of Query' no longer uses the POST-QUERY or POST-CHANGE trigger. The 'Where Clause of Query' on the lookup table usage is now added as a sub-select to the 'Where Clause' of the base table block in the generated form. This is a much better implementation, as you no longer get an error message when querying an existing record that violates the restriction. A side effect of the implementation is that if you have used system variables in the old 'Where Clause' and you fail to clear the 'Where Clause of Query' property, your form will fail to compile as references to System variables are not allowed in the block 'Where Clause'.

Titling of First Block

Preference PAGTFB governs the titling of the first block on a page. This preference did not work when set to Yes in version 1.3. In version 9*i* the preference works fine, which means that if the value is set to Yes, and you generate the form again with version 9*i*, the layout will be different from the 1.3 generation as the block title will appear now.

Names of Lookup Items Changed

In Designer 1.3.2, the user could not control the name assigned to lookup items in the generated form. The generator typically determined the item name by prefixing the column name with DSP_. In Oracle9i Designer, the user is able to explicitly set the name of lookup items, and the default is set to L_<column name>. But, the migration from Designer 1.3.2 to Oracle9i Designer sets the lookup item names to the new style names, rather than the old style names. Thus, if you have any PL/SQL code in your libraries that refers to lookup items, the code will now be incorrect.

You will need to either:

- update the lookup names (where necessary) in Oracle9i Designer to the old names, or
- update the PL/SQL code to use the new names.

Display in LOV Property Lost

In migrating from Designer 1.3.2 to Designer 6.0, the Display in LOV property is lost for all items except the FK item. Since this is lost in 6.0, it is also not present in 9i.

You can either:

- Correct the lookup column usages in Designer 6.0 before migrating to Oracle9i Designer, or
- Correct the LOV components in Oracle9i Designer.

Long Item Names Truncated to 28 characters

If a column usage name was longer than 28 characters in Designer 1.3.2, it will be truncated in Oracle9i Designer. If you have written pl/sql code in your modules that uses the long column name, it will no longer compile.

You will need to either:

- update the column names (where necessary) in Oracle9i Designer to the old names, or
- update the PL/SQL code to use the new names.

LOV Tile lost during upgrade

The LOV title is lost during the upgrade from Designer 1.3.2 to Designer 6.0. Since this is lost in 6.0, it is also not present in 9i.

You will have to manually add the title back to each LOV.

Space Added below Spreadtable Horizontal Scrollbar

In Designer 1.3.2, if you set SPRSBH=1, there was no vertical space below the horizontal scrollbar for a spreadtable. In this release, there is always at least one line of vertical space below the scrollbar. There is no workaround. This has been logged as bug #1781417.

Review the setting of USEPKR

In earlier releases, if the AUTOQY preference was set to Yes:

- Form Generator generated a POST-FORM trigger to save the primary key of the current row into a global variable upon exit from a form.
- If the first block of a generated form is not insertable, Form Generator generated a WHEN-NEW-FORM-INSTANCE trigger that uses the value of the global variable to query the base table of the block.

In Designer 9i, the USEPKR preference replaces AUTOQY and the requirements for the first block have changed. If you set USEPKR to Query, the first block in the generated form must be query only (i.e. it must not be insertable **or updatable**). If you set USEPKR to All, the first block can be insertable and/or updatable.

So Form Generator will no longer generate auto-queried forms where the first block must not be insertable but can be updatable.

- If it is acceptable for the first block to be query-only (i.e. Not insertable or updatable), set USEPKR to Query.
- If the first block must be updatable, set USEPKR to All. Be aware that blocks that previously did not meet the conditions for AUTOQY (i.e. blocks that are insertable) will also be autoqueried.

Menu Separators

You generate menu separators by defining a module with the Language set to null. During the upgrade process, the module language is defaulted to 'Developer/2000 Forms'. Therefore, you must clear the Language property again for the menu separator modules before you generate the menu.

Name Resolution in Forms 9i

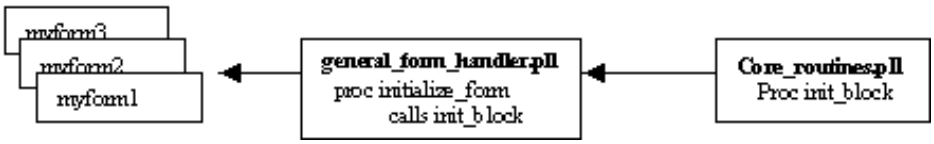
Strange behavior may occur whenever you have a routine in one .pll library that calls a routine in another .pll library. It does not occur when the routine in the .pll library is called directly from the form.

This is actually a Forms migration issue, rather than a Designer issue. However, since Designer 1.3.2 required extensive use of pl/sql libraries for application code, you are likely to run into this issue, so we will discuss it here.

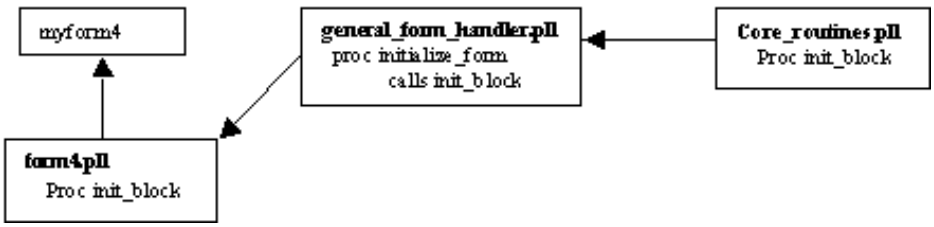
With Forms 4.5, you could define a program unit in one library, and then override it by creating a program unit of the same name in a second library and adding the second library before the first library in the list of attached libraries.

For example, suppose you created a library, general_form_handler.pll with a procedure initialize_form. This procedure calls a number of standard routines that are stored in core_routines.pll.

You attach core_routines.pll to general_form_handler.pll, and you attach general_form_handler.pll to your generater template form so that it is automatically attached to all generated forms.



Now, suppose in myform4, you want to customize the procedure init_block from core_routines.pll. With Forms 4.5, you could simply do the following.



When you called initialize_form from myform1, 2 or 3, it would call init_block from core_routines.pll. When you called initialize_form from myform4, it would call init_block from form4.pll.

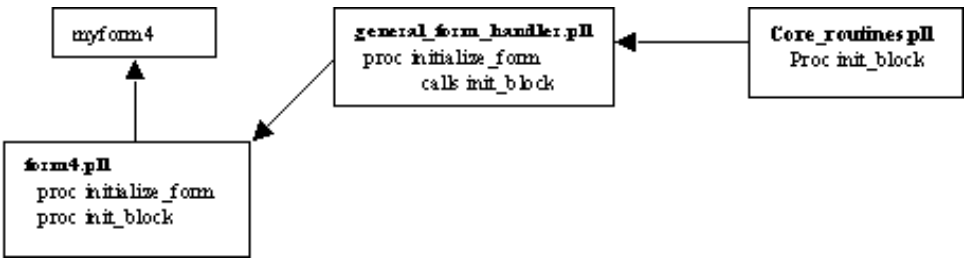
This worked because form4.pll is placed ‘on top of’ core_routines.pll in myform4’s list of attached libraries. Internally, this happens because, in Forms 4.5, every time a call is made to a procedure, a new ‘search’ for a procedure with that name is executed, finding the one closest to the form. This process is called 'Name Resolution'.

In Forms 9i (and all forms versions 5.0 and higher), the algorithm for name resolution has changed to improve performance. When a library procedure calls another library procedure for the first time, name resolution is performed, and the program unit that is called is cached in memory. The next time the same call is made, it uses the cached routine.

In our example, this results in seemingly strange behavior at runtime. If the user runs the application and opens myform4 first, and then later opens myform1, 2 or 3, then all of the forms will use the version of init_block from form4.pll. On the other hand, if the user opens myform1, 2 or 3 first, and then opens myform4, all of the forms will use the version of init_block from core_routines.pll. The behavior of all forms depends on which version of init_block was opened *first*.

There are two workarounds for this problem.

- Copy the chain of calling program unit(s) to the module specific library, even if you do not want to customize these program units.



Since name resolution is always performed for routines that are called directly from a form, myform1, 2 and 3 will call initialize_form from general_form_handler.pll and myform4 will call initialize_form from form4.pll. Then, since these two versions of initialize_form are not the same 'calling' routine, name resolution will be performed again, and each version of initialize_form will call its own init_block. Thus, each form will correctly call its own version of both initialize_form and init_block.

This is the preferred workaround, but not always feasible if the program unit being customized is called from various places.

- Copy the calling program unit (initialize_form) to the module specific library. Copy and *rename* the program unit you want to customize. Modify the calling program unit to call the renamed version of this program unit.

For migration scenarios 1 and 3, where the original libraries continue to be used in the migrated application, you are more likely to run into this behavior.

Generating Reports

The report generation process has changed dramatically. Report Generator now leverages the powerful new features in Reports 9i, including the new concept of a report template file (.tdf extension).

Report templates allow you to define standard layout settings, like font, font size, margins between frames, labels and fields, etc. Refer to the Report Builder online help for more information.

With the introduction of report templates, many layout preferences have become obsolete. These preferences have been removed from Report Generator. Refer to the Report Generator online help for a complete list of new, modified and removed report generator preferences.

As a consequence, you cannot use the old report generator template, since it contains a number of boilerplate objects that are no longer supported, and relies on a number of layout preferences that have been removed. You should create new report templates from scratch, or use the ones supplied with Oracle9i Designer.

Reports with Item Groups

The Layout Format of an item group used to be inherited from the module component in which it was contained. Now, the module component and item group can each have a Layout Format. This means that for all item groups that have the Layout Format set to 'Standard', you should change the item group's Layout Format to match the value of the Layout Style of its module component.

Money Items

The display datatype Money is no longer supported. You will have to set the Format Mask property to get the money format you like.

Reports with Across Style Layout

If the module component layout style is set to 'Across', and you have defined an item group within the module component, you should make the following changes to the definition of the item group.

- Set the item group's Layout Style to Across
- Set the item group's Break Style to Across

Review the setting of PARNME

This preference is used when naming parameters for generated reports. Set it to Null to ensure that generated parameter names are the same as the names defined in the repository.

Migrating from 1.3.2, 2.1.2 or 6.0

This section covers general migration issues when migrating from Designer 2.1.2 or Designer 6.0. These issues can also affect migrations from Designer 1.3.2. The upgrade from these two releases is the same.

Copy Where Clause of Lookup to LOV

The Designer migration fails to copy the where clause on the lookup table usage to the list of values. You will have to copy this manually wherever you have used such a where clause.

Where Clause of LOV

If you have a where clause in an LOV, and that where clause references more than one column, the generated record group query will be generated incorrectly. The first column references the LOV's table usage alias while subsequent columns reference the item in the form.

Example: the where clause… REL_TYPE='S' and NAME like 'F%'

This is generated into:

```
SELECT REL1.ID            ID,
       REL1.NAME          NAME,
       REL1.DEL_STREET    DEL_STREET,
       REL1.DEL_CITY      DEL_CITY,
       REL1.REL_TYPE      REL_TYPE
  FROM CLR_RELATIONS      REL1
 WHERE /*    CG$LOVI_WC_START REL 10 */
       (REL1.REL_TYPE='S'    and :REL.NAME like 'F%')
       /*    CG$LOVI_WC_END REL 10 */
```

Workaround: You must explicitly enter the table alias in front of the column names. In the example above, this would be :

```
REL1.REL_TYPE='S' and REL1.NAME like 'F%'
```

This has been recorded as bug #1422339.

Display Properties of LOV items

The Designer migration fails to copy the display properties from the lookup table usage to the list of values column usages. You will have to copy these manually.

Further, the migration includes the primary key column in the LOV column usages, and sets the Display Property = Yes. Unless you want the primary key visible as the first item in the LOV, you must change this display property to No.

LOVs using Filter Before Query

The old Forms bug which causes a runtime error when using aliases in the select clause of an LOV that has Filter Before Query set to Yes has returned in Forms 6*i*. This has been logged as Forms bug #1347776.

You must repair all LOVs that have Filter Before Display? = Yes.

- Determine which field will be the first displayed field in the generated LOV.
- In the LOV component in Oracle9*i* Designer, change the Name property of the column usage to match exactly its associated column name. (Usually, remove the L_<alias>_ from the beginning of the Name.)

New Possibility for LOV Buttons

Designer now lets you define a runtime webforms paramater in your registry.dat file which causes a field with an LOV to display a '…' in the field body when the user places the cursor in the field. Thus, you are no longer required to generate LOV buttons to give the users a visual clue that an LOV exists.

If you want to take advantage of this, you will need to change preference LOVBUT to No. In the registry.dat file, set parameter: app.ui.lovButtons=true

Data Source Type = Query

There is a problem using Datasource Type = Query with a module component which includes a Lookup. When you use this Datasource Type, Designer generates a post-query trigger to retrieve the lookup data. The problem shows up at runtime when you query more records than will display on the screen, go to the last record displayed on the screen, and then attempt to go to the next record.

At this point, running through the web-browser, the form locks. Running client server, the form closes.

When the post-query trigger is fired, Forms also fires the on-lock trigger to lock the record. It does this because, with this datasource type, it thinks that the lookup field is a base table field. For some reason, on all records that did not show on the first screenful of records, Forms is unable to obtain a lock on the record and returns FRM-40654. At this point, the form hangs or exits.

With this datasource type, it is not necessary to generate a post-query trigger. The lookup information is already being retrieved as part of the base table query.

Solution: Either do not use this datasource type or you will have to do a post-generation change to remove the post-query trigger generated by Designer.

This has been recorded as bug #1351904.

Updateable Views

Form Generator has changed how it handles views. Forms based on views can now retrieve server-generated values from the database without any extra coding. In order to support this, Form Generator now requires that if a view usage is updateable, the associated view must have a primary key defined in Designer.

To avoid this problem, add a primary key to all your views.

Views or Tables with No Primary Key

If you have a module component based on a view or table that does not have a primary key, you must set the NXTKEY preference to No. If you don't, you will get a compile error on the form. Alternatively, you can add a dummy primary key to the table or view.

Views with Derived Columns

You can create a column on a view from an expression (such as the concatenation of two columns). However, the view column will be generated with a default width of 4000 on the database. If you then use this column in an LOV, the generated record group will not compile. A column in a record group is limited to a width of 2000.

Check the width of generated buttons

In earlier releases, Form Generator used two different sizing algorithms when determining the dimensions of generated items and buttons.

In Oracle9i Designer, Form Generator uses the same algorithm to size both items and buttons. As a result, buttons are generated with slightly smaller dimensions.

The change in dimensions will be particularly noticable where the generated application contains large buttons, and the button label occupies the entire button face.

If changed button dimensions are unacceptable, increase the Width property of the repository item from which the button was generated.

Variety of Layout Differences

This issue cannot be assessed until after you have followed your chosen migration scenario and begun generating forms. However, it is included in the ‘General Migration Issues’ chapter since the same issues apply regardless of which migration path you choose.

Once you start generating your forms from Oracle9i Designer you will notice a variety of problems with layout. Depending on your forms, the following issues may or may not cause problems for you.

- Due to the addition of Relative Tab Stops and Side by Side canvases, Designer has changed its algorithm for determining the width and height of a generated canvas and the blocks it contains. In previous releases, if the Module Component's height and width were set to null, Form Generator set the width to the value in preference PAGCWD, and set the height just large enough to contain all the data (up to a maximum set in preference PAGCHT). Now, the algorithm is much more complex. (See the Designer online help for all the details.) If you leave the height and width null, you will not get the same layout you got from earlier releases. Chances are, the generated canvases and blocks will be larger. So, where we used to advise that you always leave the module component Height and Width properties blank, it is better now to fill them in. Designer no longer just uses the values you key in. It now calculates a good fit height and width from these values.
- In previous releases, if you had a module component that was narrower than the canvas and you set the block justification preference to CENTER, the data in the module component would be centered in the module component and the module component itself would be centered on the content canvas. In this release, Form Generator centers the data in the module component, but unfortunately it left-justifies the module component on the canvas. If you want a block centered, you will have to explicitly set the block width to the same value as the canvas width in order to get the block contents centered on the canvas.
- Designer has changed how it handles multi-line text display. If the field width is too large for the canvas, it used to expand the canvas. Now it shrinks the multi-line text item. You will have to adjust your item sizes accordingly.
- Tab handling has changed considerably with the introduction of relative tabs and changing the Horizontal vs. Vertical item group property from a preference to a property. You may find you have to revisit your tab definitions.
- A multi-record block with an overflow right may need to have the number of rows reduced so that the multi-row section is not taller than the overflow section.
- Oracle9i Designer has changed its functionality regarding the handling of scrollbars on a multi-record block whose data is placed on tabbed item groups. In the previous release, there was no scrollbar on these screens. Designer now displays a scrollbar on the context canvas next to the tabbed canvases.
- Designer has changed its algorithm for calculating item widths. In general, the widths of items being generated are slightly different.
- Designer has changed how it determines the width of action items. Be aware that action item button width may not generate correctly.

Chapter 4 Scenario 1: Migrate, Regenerate All, No Redesign

In this scenario, you will regenerate your entire application from Oracle9i Designer, including all forms, libraries, menus and reports. However, you will use your existing template form(s) and object libraries (if applicable).

The goal of this scenario is to be able to generate your application out of Oracle9i Designer and achieve the same results you got when generating out of your previous Designer release. No attempt is made to redesign your existing application to make use of new features available in Oracle9i Designer.

This chapter assumes that you have performed all the actions described in Chapter 3, General Migration Issues.

Migrating from 1.3.2

This section covers migrating from Designer 1.3.2. If you are migrating from a later release, you may skip this section.

Upgrading existing Forms 4.5 template forms

A number of changes to template forms created with Developer/2000 (Forms 4.5) are necessary before Form Generator 6i can use them:

- If a template visual attribute's Background Color property is set to Arrows, it must be set to None.
- Add comment CGAP$TES_SEQUENCE_BEFORE to each template trigger. This comment is needed because it is now possible to sequence template trigger code either before or after the code that will be generated into the trigger by Forms Generator.
- Add comment CGAP$TNR to each template program unit.
- The template form must be upgraded from Forms 4.5 to Forms 6i.

Oracle Designer supplies a utility to perform these changes.

- Read the PFRMxx.txt file in <oracle_home>\cgenfxx or <oracle_home>\bin.
- Run the PFRMxx utility to upgrade Forms 4.5 templates for use with Form Generator in Oracle9i Designer.

You must run this utility for all template forms as well as referenced forms. You can do this in batch by creating a .bat file containing the following line of code once for each form to be upgraded.

pfrmxx -i <input>.fmb -o <output>.fmb ChVisAt(arrows%none) AddTrg(CGAP$TES_SEQUENCE_BEFORE)

Review the Designer online help topics '…Release 1.x style…' to determine any changes you may need to make to your template form before generating.

Upgrade existing Forms 4.5 template menus

You cannot use the PFRMxx utility to upgrade template menus.

- Open and save each template menu module using Form Builder 6i

Upgrade Forms 4.5 Libraries

Upgrading the libraries is straightforward. To upgrade manually, you open the library in Form Builder 9i (or Report Builder 9i). Form Builder will automatically start converting PL/SQL version to PL/SQL version 2. During this PL/SQL conversion you will be asked to confirm certain conversion operations. You can safely press the 'Convert All' button if you are prompted for such a conversion operation.

You can also create a .bat file to perform this upgrade in batch mode. Include the following line of code once for each library to be upgraded.

ifcmpxx module=<library>.pll userid=<user>/<pass>@<connect> module_type=LIBRARY Logon=YES Batch=Yes upgrade=YES version=45 build=No window_state=Minimize

Attached OFG Libraries

Designer release 1.3.2 attached a number of OFG libraries to generated forms. You may also have attached some of those libraries to your custom PL/SQL libraries.

The Form Generator does not use these libraries anymore. Form Generator now uses a new, renamed version for each of these libraries.

Designer 1.3.2 Library	**Oracle9i Designer Library**
OFG4BSL	OFGBSL
OFG4CALL	OFGCALL
OFG4HPL	OFGHPL
OFG4MES	OFGMES
OFG4MNL	OFGMNL
OFG4TEL	OFGTEL

You must ensure the program units in the new libraries (which still have the same names as the old version) are used at runtime instead of the old versions.

When you regenerate your forms from Oracle9i Designer, they will automatically get the new libraries attached.

However, if you attached the OFG4… libraries to your pl/sql libraries you will have to correct each library manually.

- De-attach the obsolete OFG4… libraries from all libraries
- Attach the new OFG… libraries

Review use of the obsolete preferences and the setting of OLBOLD

This preference enables you to specify that Form Generator is to continue to use preferences made obsolete by the support for object libraries.

The obsolete preferences are:

Preference	Equivalent object library object property
AUTOHP	Display Hint Automatically
BLKVSB	Show Vertical Scroll Bar
COLSEC	Enforce Column Security
IMGBEV	Image Item – Bevel
IMGCMP	Image Item - Compression Quality
IMGDHT	Image Item – Height
IMGDWD	Image Item – Width
IMGHSB	Image Item - Show Horizontal Scroll Bar
IMGQLT	Image Item - Display Quality
IMGSZS	Image Item - Sizing Style
IMGVSB	Image Item - Show Vertical Scroll Bar
LOVNAV	Push Button - Keyboard Navigable
TXTBEV	Text Item – Bevel
WINCLO	Window - Close Allowed
WINDLG	Window - Window Style
WINFHT	Window - Height
WINFIX	Window – Resize Allowed
WINFWD	Window – Width
WINFXP	Window – X Position
WINFYP	Window – Y Position
WINHSB	Window – Show Horizontal Scrollbar
WINICN	Window - Icon Filename
WINICO	Window - Minimize Allowed
WINICT	Window - Minimized Title
WINMOV	Window - Move Allowed
WINVSB	Window - Show Vertical Scrollbar
WINZOO	Window - Maximize Allowed

Note that the default setting of OLBOLD is No, so you will have to change this preference if you want to continue using the obsolete

preferences. Also note that the OLBOLD preference is provided for backwards compatibility only. This preference and the obsolete preferences will be removed in future releases of Form Generator.

Review Template Window Properties

Check if your old templates have windows which have 'Primary Canvas', 'Horizontal Toolbar' or 'Vertical Toolbar' properties set explicitly to Null instead of inheriting null from the factory settings. (A small green box is displayed to the left of the property name.) If so, re-inherit these properties. (The small green box will change to a bullet.)

Review the setting of STOOLB

This preference enables you to specify the name and extension (in the fomat myobjlib.olb) of a default object library. The value of STOOLB is used as the default in the Object Library Name field on the Generate Form dialog box.

Since you have decided not to use an object library (by choosing this migration scenario), you must set STOOLB to null.

Review use of color palettes

When generating a form, Form Generator determines which color palette to give the generated form as follows:

- If a template form is being used, Form Generator gives the generated form the template form's color palette.
- If a template form is not being used, Form Generator gives the generated form the default Form Builder color palette.

In earlier releases, the template forms supplied with Form Generator did not use the default Form Builder template. Instead, they used a color palette similar to that used in Oracle Applications forms.

In this release, the shipped template forms supplied with Form Generator have been given the default Form Builder color palette. If you compare a form generated using the old templates and a form generated using the new templates you will notice significant color differences.

However, the earlier template forms are still supplied (as .fmb files with names beginning with OFG4) to enable you to continue using the non-default color palette.

If you have used one of the template forms supplied with Form Generator to generate forms, continue to use the same template form.

Note: If you intend to use an object library as well as the template form you used in earlier versions, be aware that the objects in the shipped object libraries supplied in this release were created in forms that had the default Form Builder color palette. To generate forms using both an object library and a template form, make sure objects in the object library were originally created in a form with the same color palette as the template form. For more information, refer to the online help topic 'Notes on color palettes and Form Generator'.

Review use of coordinate systems

Form Generator applies the coordinate system and the default character cell specified for the template form to the generated form.

Note: If you intend to use an object library during generation as well as the template form you used in an earlier version, there is an issue you must be aware of:

- If the object library contains objects that are subclassed or copied into the generated form, the generated objects might be sized incorrectly. This situation occurs where objects in the object library were originally created in forms with a different co-ordinate system to that of the template form.
- To ensure generated objects are sized correctly, make sure any object library objects that will be copied or subclassed into the generated form (e.g CGSO$LOVBUT) are created in a form with the same coordinate system as the template form you are using.

Review the setting of MSGSFT

In earlier releases, you could use MSGSFT to specify a form-level procedure for displaying soft-coded messages for use instead of the standard Forms message handling procedures.

In Oracle9i Designer, MSGSFT specifies the name of a package. Form Generator is supplied with a suitable message-handling package called CG$FORM_ERRORS in the ofgtel.pll library. If you specify a package other than CG$FORM_ERRORS, the

package you specify must contain the following three program units:

- a procedure called PUSH
- a function called MSGGETTEXT
- a procedure called RAISE_FAILURE.

There are also other naming conventions and parameter requirements you must follow if you decide to specify a package other than CG$FORM_ERRORS. For these reasons, we recommend you copy CG$FORM_ERRORS and then modify the copy. For more information, refer to the help topic 'About customizing the CG$FORM_ERRORS package'.

Refer to the help system to find out more about the requirements for a suitable message-handling package.

Review the setting of CANNTC

In earlier releases, you could use Form Generator to generate stacked item groups onto different stacked canvases displayed in the same area on a form. Only one of the stacked canvases is visible at any one time. Users of the generated form select which stacked canvas to display using a control poplist. This was an effective alternative implementation to Form Builder's native implementation of tab canvases.

In Oracle9i Designer, Form Generator enables you to generate a native Form Builder tab canvas with a number of tab pages. Each tab page contains a stacked item group.

Form Generator continues to support the alternative implementation (using stacked canvases and an item group), and uses the CANNTC preference to determine whether to generate native Form Builder tab canvases or the alternative implementation. However, the default setting of CANNTC (Yes) indicates that Form Generator is to generate native tab canvases.

Note that native tab canvases require more vertical space. If the increased vertical space required by tab canavases is not available, set CANNTC to No.

Regenerating the Application System

In the Preferences Editor, at the application level, change the preferences STFFMB and STMMMB to the names of your (upgraded) form and menu templates.

Use the Batch Generation option under the Tools menu in the Design Editor to generate your application.

You must generate the following object types.

- Libraries (if you have Design Captured them)
- Menus
- Forms
- Reports

Migrating from 2.1.2

This section covers general migration issues when migrating from Designer 2.1.2. If you are migrating from a later release, you may skip this section.

First perform all the steps described in the section 'Migrating from 2.1.2 or 6.0' in the Chapter 3 "General Migration Issues", earlier in this part.

Upgrade Forms 5.0 Libraries

Upgrading the libraries is straightforward. To upgrade manually, perform the following steps.

- Open the library in Form Builder 9i (or Report Builder 9i)
- Choose Program -> Compile -> All
- Save

You can also create a .bat file to perform this upgrade by including the following line of code once for each library.

ifcmpxx module=<library>.pll userid=<user>/<pass>@<connect> module_type=LIBRARY Logon=YES Batch=Yes build=No window_state=Minimize

Regenerating the Application System

In the Preferences Editor, at the application level, change the preferences STFFMB and STMMMB to the names of your old form and menu templates.

Use the Batch Generation option under the Tools menu in the Design Editor to generate your application.

You must generate the following object types.

- Libraries (if you have design captured them)
- Menus
- Forms
- Reports.

Migrating from 6.0

This section covers general migration issues when migrating from Designer Release 6.0. If you are migrating from a later release, you may skip this section.

First perform all the steps described in the section 'Migrating from 2.1.2 or 6.0' in Chapter 3 "General Migration Issues", earlier in this document.

Regenerating the Application System

In the Preferences Editor, at the application level, change the preferences STFFMB and STMMMB to the names of your old form and menu templates.

Use the Batch Generation option under the Tools menu in the Design Editor to generate your application.

You must generate the following object types:

- Libraries
- Menus
- Forms
- Reports.

Chapter 5 Scenario 2: Migrate, Regenerate All, with Redesign

In this scenario, you will regenerate your entire application from Oracle9i Designer, including all forms, libraries, menus and reports. You will use the new templates and object library provided by Oracle9i Designer. As you regenerate each module, you will make use of new features as appropriate.

The goal of this scenario is to take advantage of the new features available in Oracle9i Designer. As with Scenario 1, you want to be able to generate your application and get the same user interface you got from your previous release of Designer. However, many new features have been added to Designer to make achieving the desired result easier. Many features that were difficult or impossible to generate with earlier releases of Designer are now supported. Thus, in one pass, you can eliminate post generation modifications and difficult constructs that were used only to work around limitations of earlier releases of Designer.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all the sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from 1.3.2

This section covers migration from Designer 1.3.2. If you are migrating from a later release, you may skip this section.

Object libraries

To take full advantage of the the new features of Oracle9i Designer, you must use an Object Library. You can use the default object library provided with Designer, create an object library from your old reference form, or use a combination of the two.

Benefits

Object libraries are a new feature in Oracle Developer and are used by Form Generator in Oracle Designer. In earlier versions of Form Generator, much of the functionality provided by object libraries was available using preferences and objects defined in templates.

Since Oracle Designer Release 2, Form Generator has been able to use source objects in the object library to set the properties of generated objects. This method is more flexible since you can change object properties in the object library and propagate those changes to subclassed objects in generated forms simply by recompiling the forms (i.e. you do not have to generate the forms again). We strongly recommend you make use of the flexibility offered by object libraries to set and maintain standards in the applications you generate.

About object libraries

An Oracle Developer object library is a container for reusable objects. Object libraries enable you to:

- reuse an object in many forms
- apply standards to existing objects.

You can copy both objects and subclass objects from object libraries.

Object libraries are the recommended method for subclassing objects. Changes made to a source object in an object library can be propagated to any objects subclassed from that object by recompiling the appropriate forms.

About subclassed objects

A subclassed object is similar to a copied object, except that a subclassed object maintains a link to its source object.

Subclassing objects is an efficient way to reuse objects and enforce standards. A subclassed object automatically inherits any changes that have been made to the source object when you recompile a module containing it.

Object libraries are the recommended method for subclassing objects, since it is convenient to keep the source of subclassed objects separately. Changes made to properties of a source object in an object library can be propagated to any objects subclassed from that object by subsequently recompiling the appropriate forms.

Note: If you make changes to a subclassed object property that has been inherited from a source object, subsequently recompiling the form will not modify the changed property. Changing the subclassed property removes the link between that property and the source object. However, the link between unchanged subclassed properties and the source object remains in force.

Form Generator and object libraries

When you generate forms using Form Generator, you can specify that Form Generator creates generated objects by either copying or subclassing from properties of source objects in an object library specified by the STOOLB preference.

You will typically want generated objects to inherit their properties from source objects, although you can direct Form Generator to use preference settings or repository information instead. But because source objects have all the properties of generated objects, using source objects enables you to set generated object properties that cannot be set using preferences or repository information.

Object libraries used by Form Generator can contain two types of source object:

- standard source objects
- implementation source objects.

Standard source objects

A standard source object is an object in the object library that Form Generator is programmed to recognize and use when creating a particular type of generated object. Form Generator uses the names of standard source objects to determine which to use during generation. Standard source objects have names prefixed with CGSO$. The rest of the object's name indicates the generated objects for which it can be the source. For example, the CGSO$CHECK_BOX standard source object is the source object for generated check boxes. You can find a complete list of standard source objects in the Designer online help as well as in the default object library shipped with Designer.

Implementation source objects

An implementation source object is an object that can be used to override a standard source object. You can explicitly direct the Form Generator to use an implementation source object (instead of a standard source object) when generating a particular object. You use the repository object definition's Template/Library Object property to specify the implementation source object that Form Generator is to use.

You can specify two types of implementation source object:

- user-defined implementation source objects, which you have created and named yourself
- shipped implementation source objects, which provide default Oracle Developer functionality and which have names prefixed with CGAI$.

User-defined implementation source objects

You can add your own implementation source objects for any type of object. When creating a user-defined implementation source object, we recommend you do not give it a name that begins with CGSO$ or CGAI$.

If Form Generator copies or subclasses properties of a generated block from an object library block, any items belonging to the object library block are also included in the generated form. If such an item has a name that begins CGAI$, Form Generator places the item below other items in the generated block and beside buttons generated from action items defined for the module component. For more information, refer to the help topic 'Items copied or subclassed from items in object library blocks'.

Shipped implementation source objects

The shipped object library ofgstnd1.olb contains a number of implementation source objects that provide default Oracle Developer functionality. The names of these implementation source objects begin with CGAI$. You can specify one of these implementation source objects for the object Form Generator to use when generating:

- an action item
- an unbound item with a Display Type of Button.

Note: This same functionality was provided in earlier versions of Form Generator by CG$ button items in the template. Although the CG$ button item method continues to be supported in this release, we recommend you use CGAI$ implementation source objects instead.

For a list of the shipped implementation source objects and the equivalent CG$ template button items, refer to the help topic 'Alphabetical list of shipped implementation source objects'.

Creating an object library using the FM2LIBxx utility

Creating an Object Library the Hard Way

To create an object library using only Form Builder, you would have to perform the following steps:

- Create an empty object library (.olb file).
- Create a dummy form (.fmb file).
- Add objects to the dummy form.
- Drag and drop the objects into the object library.
- Discard the dummy form.

To maintain the object library, you would have to perform the following steps:

- Open the object library.
- Create a dummy form.

- Drag and drop the object you wish to modify from the object library to the dummy form.
- Make the desired changes.
- Drag and drop the object back to the object library.
- Discard the dummy form.

This method has a number of disadvantages. First, the process itself is cumbersome and error prone. Much worse though is the fact that if you have objects that are subclassed amongst themselves, then when you drag them out and back into the object library for maintenance, all the subclassing is lost.

Creating an Object Library the Easy Way

Designer supplies a utility called FM2LIBxx (where xx is the Designer version number) that will automatically create an object library from a form.

This utility enables you to create and maintain source objects in forms. Essentially, you create an 'object library maintenance' form which includes everything you want to have in your object library. This form is not discarded, but kept as part of your template set. Whenever you need to modify your object library, you make the changes in this maintenance form.

You use the FM2LIBxx utility to generate the object library .olb file from the maintenance form.

There are several advantages to this approach, including:

- Ease of maintenance - you do not have to drag objects into a form module for editing and then drag them back into the object library.
- Subclassing information is retained when editing objects - subclassing information is lost when you drag objects into an object library from a form module for editing.

For more information about using the FM2LIBxx utility, refer to the file fm2libxx.txt in the <ORACLE_HOME>\CGENFxx directory.

Object libraries and earlier versions of Form Generator

Object libraries, obsolete preferences, and OLBOLD

In previous versions, Form Generator used a number of preferences to set object properties. For example, you might have set the IMGBEV preference to determine the style of the border surrounding images.

The support for object libraries has resulted in a number of preferences becoming obsolete. For a full list of obsolete preferences, refer to the online help topic 'Alphabetical list of obsolete preferences and the equivalent object library properties'.

We strongly recommend you do not use the obsolete preferences. However, to ensure backwards-compatability, you can set the OLBOLD preference to specify that Form Generator continues to use these preferences instead of properties in the object library.

Note: The OLBOLD preference and the obsolete preferences will be removed in future releases of Form Generator.

Object libraries and CG$ template objects

In previous versions, Form Generator used a number of objects from the template form to control properties of generated objects.

In Oracle Designer Release 2, Form Generator continues to use CG$ template objects as in previous releases - with one exception. When creating an LOV indicator button, Form Generator first searches the object library for a standard source object called CGSO$LOV_BUT and uses that instead of a CG$LB template object if one has been defined.

Note: It is likely that CG$ template objects will be replaced by object library functionality in future releases of Form Generator.

Object libraries and CG$ template button items

In previous versions, Form Generator used a number of CG$ template button items from the template form to provide default Form Builder functionality.

In this release, Form Generator continues to use CG$ template button items. However, equivalent functionality is also available through the use of shipped implementation source objects (prefixed CGAI$) in the object library. For a list of the shipped implementation source objects and the equivalent CG$ template button items, refer to the help topic 'Alphabetical list of shipped

implementation source objects'.

Note that the behavior of Form Generator with regard to shipped implementation source objects and CG$ template button items is slightly different. If a template contains CG$ template buttons, every form generated with that template contains those CG$ template buttons. To include the same functionality in every generated form using CGAI$ objects, either explicitly specify implementation source objects for individual unbound items in every module (potentially difficult to maintain and not recommended), or follow the instructions below to generate a common control block:

- Create a reusable module component.
- Create an unbound item for each generated item you want in the control block.
- Set the Template/Library Object property for each unbound item to the appropriate CGAI$ implementation source object.
- Include the reusable module component as the first or last component in every module.

Object libraries and Generator named visual attributes

In previous versions, Form Generator applied a number of Generator named visual attributes (NVAs) from the template form to generated objects.

In this release, Form Generator can use the properties of standard source objects in the object library to set the properties of generated objects in much the same way as using Generator NVAs. This method is more flexible since you can change object properties in the object library and propagate those changes to subclassed objects in generated forms simply by recompiling the forms. We strongly recommend you make use of the flexibility offered by object libraries to set and maintain standards in the applications you generate.

However, to ensure backwards-compatability, Form Generator continues to use Generator NVAs on objects that have not been subclassed or copied from an object library as follows:

- If an implementation object in the object library has been specified for an object, use this if it exists.
- If an implementation object has been specified and it does not exist, or if no implementation object has been specified, use an appropriate standard source object from the object library.
- If no appropriate standard source object exists in the object library, apply the appropriate visual attribute from the template form.

Note: Generator NVAs will be removed in future releases of Form Generator.

Object libraries and generated items' widths, lengths, and formats

In previous versions, if information about an item's width, length, and format were not recorded in the repository (either against the column definition or the item definition), Form Generator obtained this information from a number of preferences (prefixed TXT).

In this release, if information about an item's width, length, and format are not recorded in the repository, Form Generator obtains this information from source objects in the object library. Form Generator uses the corresponding TXT preferences only if no appropriate source object exists in the object library, or if an appropriate source object exists but the relevent property has not been set.

Standard source objects and object libraries

During generation, Form Generator analyzes each object it creates to determine which standard source object it expects to use in the object library. Form Generator can use a standard source object from an object library during generation providing the standard source object:

- has one of the recognized names
- is of the correct object type.

Form Generator uses the names of standard source objects to determine which to use during generation. Standard source objects have names prefixed with CGSO$. The rest of the object's name indicates the generated objects for which it can be the source. For example, the CGSO$CHECK_BOX standard source object is the source object for generated check boxes. For a list of the recognized names of standard source objects, refer to the online help topic 'Alphabetical list of standard source objects and object types'.

The standard source objects are arranged in a predefined hierarchy. If Form Generator does not find the expected standard source object, it searches the object library for the parent of the standard source object and uses that instead. For a graphical representation of the standard source object hierarchy, refer to the online help topic 'Hierarchy of standard source objects'.

Standard source object suffixes

In some cases, you can further refine the standard source objects from which Form Generator obtains properties for generated objects by adding the suffixes below to the standard source object name:

Order	Suffix	Use if generated object will be a...
1	_MR	...multi-record block, or an item in a multi-record block
2	_CT	...control block (i.e. the module component contains only unbound items)
3	_DO	...display only item
4	_MD	...mandatory item

You can use the above suffixes individually or in combination. Form Generator will use the most appropriate standard source item, according to the order of precedence indicated.

Form Generator first searches the object library for a standard source object name containing all of the applicable suffixes. If no such standard source object exists, Form Generator searches the object library for the standard source object name that contains the greatest number of appropriate suffixes. If two or more standard source objects contain the same number of appropriate source objects, Form Generator uses the one containing the suffixes highest in the order of precedence (see table above).

Note that Form Generator ignores a standard source object if any of the suffixes are not applicable.

Examples

Example 1: Form Generator is going to create a display-only check box in a multi-record block.

Form Generator searches the object library for each of the following objects, in the order shown:

```
CGSO$CHECK_BOX_MR_DO
CGSO$CHECK_BOX_MR
CGSO$CHECK_BOX_DO
CGSO$CHECK_BOX
CGSO$DEFAULT_ITEM_MR_DO
CGSO$DEFAULT_ITEM_MR
CGSO$DEFAULT_ITEM_DO
CGSO$DEFAULT_ITEM
```

As soon as Form Generator finds one of these objects, it subclasses or copies the object into the generated form and abandons the search. If Form Generator fails to find any of these standard source objects, it creates a check box item using default item properties.

Example 2: The object library contains the following objects:

```
CGSO$CHAR_MD_DO
CGSO$CHAR_MR
CGSO$CHAR_DO
CGSO$CHAR
```

Form Generator is generating a form from a module that contains a display-only character item in a multi-record block. Form Generator subclasses or copies the CGSO$CHAR_MR standard source object into the generated form because:

- the _MD suffix in CGSO$CHAR_MD_DO is inappropriate
- the _DO suffix in CGSO$CHAR_DO is lower in the suffix order of precedence than _MR.

Example 3: The object library contains the following objects:

```
CGSO$DATE_MR
CGSO$CHAR_MR_MD
CGSO$CHAR_DO
```

CGSO$DEFAULT_ITEM

Form Generator is generating a form from a module that contains a mandatory data item in a single record block. Form Generator subclasses or copies the CGSO$DEFAULT_ITEM standard source object into the generated form because:

- the _MR suffix in CGSO$DATE_MR is inappropriate
- the only occurrences of CGSO$CHAR (i.e. the parent of CGSO$DATE) have inappropriate suffixes (_MR_MD and _DO).

If the object library had contained a standard source object called CGSO$CHAR, or failing that an object called CGSO$DEFAULT_ITEM_MD, Form Generator would have used these objects instead.

Current Record Indicator and the Object Library

If you want to use the Current Record Indicator in your forms, you must create a Visual Attribute called CG$CURRENT_RECORD in your object library. This is not included in the Designer default object library, so you will need to add it.

Application logic

Benefits

The ability to record client-side code or 'application logic' in the repository is a new feature in Oracle Designer since release 1.3.2.

In earlier releases, you could include your own code in generated applications only by entering code for template objects or in attached libraries. This method was cumbersome and prone to error.

In this release, not only can you enter the code in the repository but you can also:

- specify which events in the generated application cause the code to execute
- specify whether your code executes before, after, or instead of code created by Form Generator.

Storing the code within the repository also enables you to make use of Oracle Designer's dependency analysis features to assess the impact of proposed changes.

About application logic

'Application logic' is the name given both to code created by the Form Generator and to code you enter by hand. Application logic comprises event code (code that executes in response to particular events) and named routines (code that is called by event code).

There are different types of application logic:

- Generated application logic - event code and named routines created by Form Generator.
- User-modified generated application logic - event code and named routines created by Form Generator and subsequently modified by the user (and optionally captured into the repository).
- User application logic - event code and named routines defined by the user in the repository.
- Object library (OL) application logic - event code and named routines for particular objects that are copied into the generated application from a specified object library during generation.
- Template application logic - form-level event code and named routines that are copied into the generated application from a template during generation.

You can record application logic at the module level, the module component level and the item level.

In Form Builder, event code is implemented as trigger code. Event code segments are individual pieces of trigger code. A named routine becomes a program unit held at module level or in an attached library, or on the server.

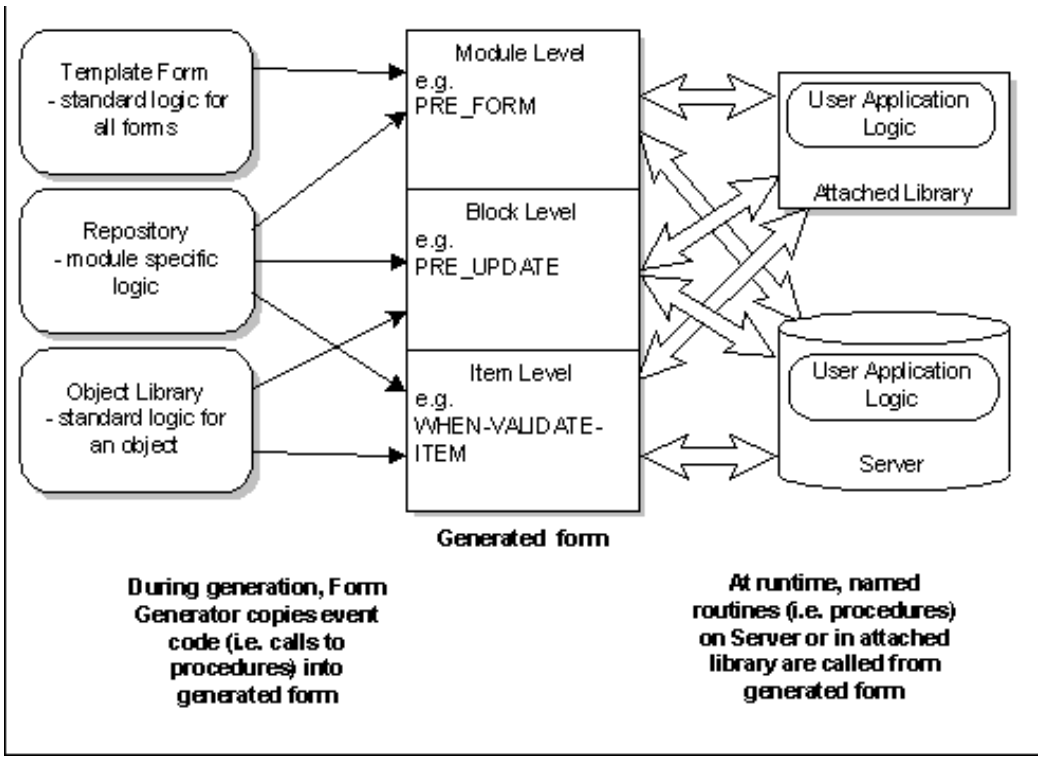
Where to enter application logic

The answer to this question depends on the complexity of the code involved. If the code is very simple (a few lines of code), then you can enter the code directly in the event logic in the form module. However, if the code is complex, we recommend you include as little code in the generated form as possible. In practice, this means including event code in the form that simply calls procedures on the server or in an attached library. This strategy offers several advantages:

- Maintaining generated applications is easier because code on the server or in a library can be modified without recompiling generated forms.

- In the case of server-side procedures, network traffic is minimized and performance improved.
- It promotes reusability, since the same code can be used by different forms.
- It reduces the size of .fmx.

The recommended scenario is shown in the diagram below.



We recommend that named routines that require database access (where the code includes SQL commands) be placed in stored procedures on the server and not in an attached library.

Where to enter event code

If the event code (or calls to code stored on the server or in an attached library) is to be included in every form:

- Enter block level and item level event code in an object library.
- Enter form level event code in the template form.

If event code is to be included in a single form or in some but not all forms, enter the event code in the repository against the required modules, module components and items.

Where to enter named routines

In a typical client server environment, we recommend you create named routines (e.g. procedures) either on the server or in a library attached to the generated form. These named routines are then called by the event code incorporated in the generated form from the repository, an object library or the template.

If the named routines are to be held on the server:

- Enter the named routine in the Oracle Designer repository using the Logic Editor, and then run the Server Generator to create the named routines on the server.
- Enter the named routines manually in a text file, and then run the text file using SQL*Plus.

If the named routines are to be held in an attached library:

- Enter the named routines in a library manually using Form Builder and attach the library to the template form.
- Enter the named routines as application logic in a repository module of type Library, link the repository module to one or more modules of language Oracle Developer Forms (or Oracle Developer Reports) in a module network, and use Library Generator to generate the library module.

Capturing application logic from Form Builder applications

You can capture (or 'reverse engineer') application logic into the repository from:

- Forms you did not generate using Form Generator
- Forms you did generate using Form Generator and subsequently modified using Form Builder
- PLL modules (which can now be stored in Designer).

During design capture, any user application logic or user modified generated application logic is captured into the repository. However, Form Generator uses comments in the code to identify and ignore:

- generated application logic
- template application logic
- object library application logic.

Where a trigger is captured, the execution style property of the corresponding event is set to the same value.

Note that BEGIN and END; statements must appear on separate lines.

About generated application logic

What is generated application logic?

Generated application logic is the code generated by Form Generator each time a form is generated. Generated application logic is not held in the repository.

If the form has already been generated, any existing generated application logic is overwritten.

Generated application logic comprises:

- generated code groups
- generated named routines.

Although generated application logic is not stored in the repository, generated code groups are represented in the Design Navigator window. This enables you to sequence user code groups and user-modified generated code groups before and after generated code groups.

Note that generated named routines are not represented in the Design Navigator window.

What are generated code groups?

Generated code groups are the standard sections of trigger code (and any called program units) generated by Form Generator.

Generated code groups are identified by a unique comment (e.g. /*CGRI$CHECK_ON_DELETE*/). If the generated form is captured into the repository, these comments will enable Form Generator to identify the code as generated application logic. Generated application logic is not captured into the repository during design capture.

For a list of generated code groups, refer to the following help topics:

- "Form Generator code groups (grouped alphabetically)"
- "Form Generator code groups (grouped by functionality)".

What are generated named routines?

Generated named routines are the standard program units created at module level by Form Generator.

About user-modified generated application logic

Using Form Builder, you can modify the application logic that Form Generator has generated.

To prevent user-modified generated application logic from being overwritten if the form is generated again, you must capture the application logic into the repository.

Capturing the user-modified generated application logic into the repository enables you to generate the form again, incorporating the changes you have made in Form Builder.

About user application logic

User application logic is both:

- code you enter into the repository using the Design Editor and the Logic Editor
- code you enter using Form Builder and then capture into the repository.

When you enter user application logic, you specify it as being one of the following:

- A named routine, which is generated as a Form Builder program unit and called from triggers or other program units
- An event code segment, which is generated as Form Builder trigger code

About OLB application logic

Object Library (OLB) application logic is code (event code and named routines) associated with objects in the object library that is added to blocks and items created during form generation.

When Form Generator includes an object from the object library in a generated form, any code associated with that object is also included in the generated form.

About template application logic

Template application logic is the name given to named routines or event code (program units or form level triggers) that you create in the template form. You might create template application logic to perform a common function that you want in all the forms you generate.

Form Generator copies program units from the template form straight into the generated form.

Form Generator copies form level trigger code from the template into the generated form. Form Generator uses comments in the template trigger code to determine whether to sequence it before or after generated code and user code for the same trigger.

Form Generator also copies form-level sub-classed triggers into the generated form. In this case, note that Form Generator is unable to modify the sub-classed trigger code. Therefore, any functionality Form Generator would have added to the trigger had it been non-sub-classed is not included in the generated form.

Unbound Items

In Designer 1.3.2, if you wanted to create a non-base table item you used a concept called Secondary Column Usages. This release of Designer allows you to create Unbound Items, a much more flexible implementation of non-base table items.

Unbound items are items that exist within a module component, but are not associated with any table usage. You can define the type, placement and display, as well as application logic, for unbound items. You can also define an LOV against an unbound item.

Your Secondary Column Usages will be automatically migrated to unbound items. However, they will retain the old (ugly) names from 1.3.2. Consider renaming the unbound items to more descriptive names. Be aware however, that any code referencing the old names will need to be updated to use the new names.

Action Items

Action items are a shortcut method of creating buttons in your application. There are two types of action items:

- Navigation Action Items – used to navigate to other blocks in the same form or to navigate to other forms
- Custom Action Items – used to generate all other button items.

Navigation action items are quick and easy to create and the code for the WHEN-BUTTON-PRESSED trigger is automatically generated.

However, action items are always displayed on the bottom of the current window. Designer does not provide a mechanism for specifying where you want the action item placed on the screen. If you want to control where the buttons are placed, you can:

- Write a PRE-FORM trigger that explicitly sets the x and y coordinates of the generated buttons.
- Use an Unbound Item with display type Button instead (and write your own WHEN-BUTTON-PRESSED trigger).

You should replace all buttons that were generated as secondary column usages with either unbound items or action items.

Again, you can rename the action items to more descriptive names. You will need to update any code that references the old names.

Reusable module components

Benefits

In previous releases, the only way to copy part of one module to a new module was to copy the complete module, delete those elements that you did not want, and then use the remainder as the basis for the new module.

In this release, you can simply specify that a module component be made reusable. Making a module component reusable encapsulates the properties, preferences and event code. You can then include this reusable component in any number of modules.

Reusing module components in this way increases developer productivity and encourages consistency. The modules that use the reusable component automatically inherit all changes that you make to the reusable component, its application logic, and its associated preferences.

Reusable module components are an ideal way of defining control blocks to help give generated forms a common look and feel. For example, placing standard Form Builder functionality in a common control block at the top of every form presents users with a consistent interface.

Native Form Builder tab canvases

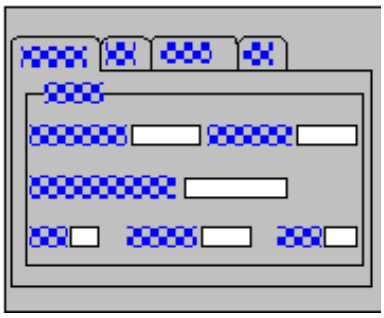
Benefits

Screen real estate on which to lay out the items that comprise an application is often severely limited, either by the physical size of the screen or by the number of fields that have to be displayed. Users frequently have to navigate through many screens to enter information.

In previous releases, a number of strategies were available to increase the virtual size of the screen:

- Multiple content canvases
- Multiple stacked canvases
- Spread tables
- Different stacked canvases displayed in the same area on a form and accessed via a poplist.

In this release, Form Generator can generate native Form Builder tab canvases, each containing two or more tab pages.



Native Form Builder tab canvases have standard Windows look-and-feel and an intuitive behavior. They can form an integral part of your screen layout strategy.

About tab canvases and tab pages

What is a tab canvas?

A tab canvas enables you to group and display a large amount of related information on a single, dynamic canvas object. Like stacked canvases, tab canvases are displayed on top of a content canvas, partly obscuring it.

A tab canvas is made up of one or more tab pages. Tab pages each display a subset of the information displayed on the entire tab canvas.

What are tab pages?

A tab canvas can have many tab pages, and must have at least one. Think of tab pages as the folders in a filing system. Each individual tab page (folder) has a labeled tab that developers and end users click to access the page. You click the labeled tab to display the page at the front of the tab canvas, thereby obscuring any other page(s).

Each tab page occupies an equal amount of space on the tab canvas, roughly equal to the size of the canvas itself. This layering of related canvas sections enables you to display more information on a single canvas.

Tab pages are sub-objects of a tab canvas. Like the canvas to which it is attached, each tab page has properties.

About generated tab canvases and tab pages

What are generated native Form Builder tab canvases?

Form Generator generates a native Form Builder tab canvas and tab pages on which to place:

- Generated blocks, if two or more repository module components have their Placement properties set to New tab canvas page
- Generated stacked item groups, if two or more repository item groups in a module component have their Stacked property set to Yes.

If the above conditions are met and CANNTC is set to Yes, Form Generator places each block generated from a module component and each stacked item group onto different tab pages.

On selection of a tab label, the corresponding tab page is displayed and the cursor navigates to the first enterable item on that tab page. When users navigate to an item on a tab page that is not currently displayed (by pressing the [Next Item] key), the appropriate tab page is displayed.

An alternative implementation in earlier releases

In earlier releases, Form Generator enabled you to generate similar functionality using stacked item groups placed onto different stacked canvases displayed in the same area on a form. The different stacked canvases could be accessed via a poplist.

You might have existing module designs that include this alternative implementation. The CANNTC preference enables you either to make use of the new Form Generator functionality to place the stacked item groups on the native Form Builder tab canvases, or to continue to place the groups on stacked canvases.

Notes on the use of tab canvases for displaying blocks

Note the following when generating tab canvases to display blocks:

- A tab canvas (and its tab pages) generated from module components will appear at the position of the first module component that has its Placement property set to New Tab Canvas Page. It is your responsibility to make sure that blocks in the resulting form do not appear to users to be out of sequence.
- A content canvas can contain only one tab canvas generated from module components.
- If a generated form contains multiple content canvases, each content canvas can contain one tab canvas generated from module components.
- All block layout preferences apply to blocks generated onto tab pages.
- If the block you are generating onto a tab canvas page contains stacked item groups that are themselves to be generated onto a native Form Builder tab canvas, the first item in the block must not be in one of the stacked item groups. The first item must be placed on the underlying tab canvas page.
- The label for a tab canvas page is derived from the title given to the first block displayed on that tab canvas page. If you do not want both the label and the block title to appear on the tab canvas page, set POPTFB to No to prevent the block title being displayed (if the first block on a tab canvas page is the first block on the content canvas, use PAGTFB).
- If you set the X Position and Y Position properties of the first module component to be placed on the first tab canvas page, Form Generator will use those properties to position the generated tab canvas on the underlying content canvas.
- If the first block in a content canvas is generated onto a tab canvas and you set the Width and Height properties of the module component from which that block was generated, Form Generator will use the dimensions you specified as dimensions for the underlying content canvas.

By default, Form Generator determines the dimensions of a generated tab canvas as follows:

- Form Generator determines the width of a generated tab canvas from the width of the underlying content canvas.

- Form Generator determines the height of a generated tab canvas from the height of the highest block placed on a tab canvas page.

However, you can override the default dimensions of a generated tab canvas using module component Width and Height properties. If you do use the Width and Height properties, Form Generator determines the dimensions of a generated tab canvas as follows:

- Form Generator determines the width of a generated tab canvas either from the largest Width property specified for any module component placed on a tab canvas page or from the width of the widest block to be displayed on a tab canvas page, whichever is the greater.
- Form Generator determines the height of a generated tab canvas either from the largest Height property specified for any module component placed on a tab canvas page or from the height of the highest block to be displayed on a tab canvas page, whichever is the greater.

Notes on the use of tab canvases for displaying stacked item groups

Note the following when generating tab canvases to display stacked item groups:

- A block cannot contain more than one tab canvas displaying stacked item groups.
- Tab canvases displaying stacked item groups can be laid out within blocks that are themselves displayed on tab canvases (for more information, refer to "Examples of blocks generated onto tab canvases/tab pages" in the Designer online help).
- Tab canvases displaying stacked item groups cannot appear on other stacked canvases (including spreadtables and overflow context areas).
- A tab canvas displaying stacked item groups cannot contain items from more than one block.
- To ensure items in stacked item groups on a tab canvas in a detail block are synchronized with a master block (on a different canvas), set the BSCSCP preference to Window or Form.
- If all the items in a detail block are in stacked item groups on tab pages on a tab canvas, Form Generator creates a block context area above the tab canvas for any context items from the master block (for more information, refer to "Block context area and block context item generation" in the Designer online help).

Server API

In this release, you can base a generated block on a procedure stored on the server and pass a PL/SQL table of records between the block and the procedure. It is the procedure that queries and performs DML on the base table. Basing a generated block containing lookup items on stored procedures is significantly more efficient in terms of network traffic than the other options outlined above. An Oracle Designer utility enables you to create suitable server-side procedures on which to base blocks by generating the Server API from table and module component definitions in the repository.

What is the Server API?

The Server API provides a powerful and easy-to-use PL/SQL interface that generated client applications can call to perform queries and DML operations on base tables in the Oracle Server.

The Oracle Designer Server Generator is the tool to use to generate the Server API.

The Server API is made up of:

- the Module Component API (comprising procedures to perform queries on the tables used by a generated application, and calls to Table API procedures to perform DML operations on those tables).
- the Table API (comprising procedures to perform inserts, updates, deletions and locks on the tables used by a generated application).

If you specify Procedure as the query datasource for a generated block, the block is populated by a procedure in the Module Component API (this procedure is referred to as a 'module handler') that passes a PL/SQL table of records to the generated form.

Similarly, if you specify Procedure as the DML datasource for a generated block, a table of records is passed from the block back to the module handler on the server, which in turn calls a procedure in the Table API (this procedure is referred to as a 'table handler') to update the base table.

If you intend to base a generated block on a server-side procedure generated as part of the Server API, you must generate the API before you attempt to run the generated form.

Library generation

A new addition to the Oracle Designer toolset is the Library Generator. Using Library Generator, you can:

- generate Oracle Developer library modules from module definitions recorded in the Oracle Designer repository
- capture existing Oracle Developer library modules into the Oracle Designer repository.

When generating library modules, you can run Library Generator in one of two ways:

- To generate specific library modules, you can run Library Generator standalone.
- When using Form Generator or Report Generator to generate an Oracle Developer module, you can run Library Generator automatically to generate any attached library modules.

All Oracle Developer modules can use a generated library module, unless the generated library includes calls to language-specific built-ins. If the generated library does include calls to such built-ins, Library Generator uses the appropriate Oracle Developer compiler to compile the library.

Note that it is the repository module's Language property that Library Generator uses to determine which Oracle Developer compiler to use. It is your responsibility to ensure that:

- the repository module's Language property is appropriate for the built-ins called in the module
- built-ins from different languages are not called by the same library module
- the compiler from the appropriate Oracle Developer component is available (for generating and capturing repository modules of Language Oracle Developer Common Library, both the Oracle Developer Form Builder and the Oracle Developer Report Builder compilers are suitable)

Module Specific Libraries

You will need to re-evaluate all of your module specific libraries that you created for forms generated from Designer 1.3.2. In many cases, the logic will no longer be necessary at all because of new features that have been added to Designer. In other cases, the logic may be so simple that you choose to record it as application logic in the form module itself, rather than having a separate library.

Where you do still require a separate library, you should capture the library into Designer. You can attach the library module to its related form module using the module network link, or you can continue to use the MODLIB preference if you like.

Templates cut down to size

In earlier releases, template forms were often the only mechanism for:

- controlling properties of generated objects
- including generator objects and user code in generated forms.

In this release, application logic and object libraries provide powerful alternatives to the existing template technology. In many cases, these alternatives are more efficient since:

- Using object library objects enables you to change the properties of objects in a generated form simply by changing the properties of source objects and recompiling the form. It is not necessary to generate the form again.
- Recording application logic in the repository enables you to specify when code is to execute, and to make use of Oracle Designer's dependency analysis features to assess the impact of proposed changes.

As you take advantage of other new features in Oracle Designer, you should also consider how those new features enable you to move functionality from the template and into object libraries and application logic.

However, continue to use template forms for:

- functionality requiring control blocks (e.g. toolbars, calendar widgets)
- form level code and objects (e.g. alerts, object groups, popup menus) that you want in multiple forms
- setting the character cell size and the coordinate system of generated forms
- defining the color palette used in generated forms
- attaching libraries to multiple generated forms (note that although this method is quicker than using repository module networks, you will not have access to Oracle Designer's impact analysis features)
- defining current record visual attributes

- property classes.

Migrating from 2.1.2 or 6.0

This section covers general migration issues when migrating from Designer 6.0.

Lookup Usages

In Designer 1.3.2, you used a lookup table usage to create both lookup items on the main canvas as well as to generate a Forms LOV. In this release of Designer, the concepts of lookup usages and LOVs have been separated.

The lookup table usage now has only one purpose - to display values from a lookup table on the canvas. These lookup items can be enterable, queriable and sortable.

You can attach a List of Values to base table column usage, a lookup column usage, or an unbound item. The only requirement is that the item must be explicitly defined as Insertable and/or Updateable.

See the next section for information on creating a list of values.

LOV generation

The new repository LOV element simplifies the process of LOV generation by decoupling LOVs from lookup table usages. This not only makes it easier to define LOVs, it also enables you to:

- define multiple LOVs for the same block
- reuse the same LOV in multiple blocks and multiple forms
- define LOVs for unbound items.

Native Oracle Forms LOVs

Using Form Generator you can generate Oracle Forms LOVs to populate two kinds of generated text items:

- Items generated from repository bound or unbound item definitions that are explicitly associated with repository LOV definitions.

  In this case, the generated LOV is always based on a query. The generated LOV will be available for those items explicitly associated with the repository LOV definition. You can specify which values are returned from the LOV to the form and which items in the form are populated with the returned values. In addition, you can specify different LOVs to use for querying information and for entering information.

  You can associate LOV definitions with both foreign key bound items and non-foreign key bound items. If you associate an LOV definition with a foreign key item in a base table usage, the lookup table at the other end of the foreign key might be the same as the LOV base table usage. If this is the case, note that:
  - You do not have to define a module component lookup table usage of the lookup table as well.
  - If you do define a module component lookup table usage of the lookup table as well, the module component lookup table usage's 'Where Clause of Query' property and 'Validation Where Clause' property will be used to restrict the values in the LOV when querying data and entering data respectively.

  The repository LOV definitions you create can be either specific to a single module or reusable by multiple modules.
- Items generated from repository bound or unbound item definitions for which allowable values have been recorded.

  In this case, the generated LOV can either be based on a static record group hard coded into the generated form or on a query record group populated dynamically by a query on a reference code table. For more information about reference code tables, refer to "Reference code table generation" in the Designer online help.

Lists of Values forms

By default, Form Generator generates native Oracle Forms LOV objects. However, you can also create a list of values form. This is a separate form that behaves like a list of values. It is called whenever the user presses the list of values button. It can return data to the calling form. The advantage of a list of values form is that the user can perform complex queries to find the requested records. It is also much better when a large number of records needs to be displayed.

To create a list of values form, you create a form module in Designer and set its Layout Format property to LOV, using the FKLOVT preference.

To call a list of values form, specify the form you want to call using the DVLOVF and DVHLVF preferences in the calling form.

Poplists

You can use repository LOV definitions to generate poplist, text list and combo box items by associating a repository LOV definition with an item that has its Display Type property set to Poplist, Text list or Combo box.

Reusable LOV Components

Oracle9i Designer also allows you to create reusable LOV components that can then be included in any form module.

Multi Region Blocks

Most blocks you generate will be single region blocks. Single region blocks are based on a single table and comprise one rectangular layout region that cannot be split across different canvases or windows. You generate a single region block from a single module component.

In some cases, you will want to generate multi-region blocks. As with single region blocks, a multi-region block is based on a single table. However, a multi-region block comprises a number of different rectangular layout regions. The layout regions in a multi-region block can be placed on:

- different places on the same canvas
- different canvases in the same window
- different canvases in different windows.

You generate a multi-region block by defining:

- a primary module component with one base table usage and (optionally) one or more lookup table usages
- one or more module sub-components, each with a usage of the same base table as the primary module component, and (optionally) each with one or more lookup table usages.

Form Generator creates a rectangular layout region for the primary module component and for each module sub-component. Although they contain items based on columns from the same table, the different layout regions can display different numbers of rows. The data in different regions in a multi-region block is always synchronized (i.e. the regions all display data for the same current record).

You can create bound items based on the same columns as bound items in the primary module component (or any other module sub-component). Note that Form Generator will generate second and subsequent items as mirror items. Also note the standard source objects for these mirror items will be the same as for the first bound item (i.e. not CGSO$CONTEXT, which is the usual standard source object for mirror items).

Side By Side Blocks

By default, Form Generator places the second and subsequent blocks on the same canvas below the previous blocks on that canvas. Use a module component's Placement property and Right of Component property to place a block beside a previous block instead of below it.

When you create a new module component, it is automatically placed in a new window and on a different canvas. To generate two blocks side-by-side on the same canvas, you must first include the corresponding module components in the same window and on the same canvas.

To place a block beside a previous block on the same canvas:

- Set the module component's Placement property to Right of.
- Set the module component's Right of Component property to the name of a previous module component.

Navigator Style Forms

A navigator style form enables users to quickly navigate through hierarchical data to locate a particular record and comprises:

- a hierarchical tree item in a navigator window
- one or more detail windows in which record details can be viewed and edited.

The look and feel of the expanding and collapsing tree is similar to the tree in the Design Editor's navigator window.

You can use Form Generator to generate a navigator style form by setting a module's Layout Format property to Navigator.

Form Generator generates the first block in the form on a new content canvas into a new window (the navigator window) and creates an item in the block of type Hierarchical Tree (note that the window, canvas, block and item are not modeled in the repository).

All blocks and records in the generated form are represented as nodes in the hierarchical tree item. When the form is opened, the navigator window displays all master blocks in the form as unexpanded 'block nodes'. If the user expands one of the block nodes, the records in the block appear as 'record nodes'. If the master block is linked to one or more detail blocks, each record node can be expanded to display a block node for the detail block(s). If the user expands the detail block node, the records in the detail block appear as record nodes.

At any point, the user can select a record node and view or edit that record in the corresponding block in a separate detail window.

You can also include buttons on vertical and horizontal toolbars in the navigator window to provide extra functionality. For example, you might want to include buttons to expand all nodes and collapse all nodes.

See the Oracle Designer online help for more information about generator Navigator Style Forms.

Relative Tab Stops

Form Generator enables you to specify the points at which Form Generator is to position the starts and ends of items and their prompts. These points are referred to as tab stops.

In this release of Designer, there are now two kinds of tab stop:

- Absolute tab stops (old style - tab stop positions are specified in character cells and items are placed at those positions)
- Relative tab stops (new style - items sharing the same tab stop are positioned relative to each other).

Note that the use of absolute tab stops and relative tab stops are mutually exclusive.

Relative tab stops are numeric values that enable you to position and align items and item groups relative to each other (i.e. next to each other or below each other). Using relative tab stops also enables you to explicitly specify that you want a particular item or item group to appear on a new line.

You set relative tab stops for both items and item groups using either or both of the following relative tab stop properties:

- Start Tab Stop property
- End Tab Stop property

If you specify relative tab stops, during generation Form Generator horizontally positions items/item groups so that any items/item groups with the same relative tab values are vertically aligned below one another. The Start Tab Stop property and the End Tab Stop property determine whether it is the left hand edge, the right hand edge, or both edges of the item/item group that are aligned.

In addition, an item's Align Prompt property enables you to vertically align the starts of prompts of items that are vertically aligned (note that Form Generator might ignore this property to optimize the use of space).

When using relative tab stops, you can set the Tab Stop Scope property of a container object (i.e. an item group or module component) to align:

- items within an item group with items outside the item group
- items within one module component with items in another module component.

For detailed illustrations of how to use relative tab stops, refer to the online help topic 'Examples of the use of relative tab stops'.

Notes on the use of relative tab stops

Note the following when using relative tab stops:

- In general, Form Generator uses relative tab stops only if relative tab stops have been defined and the Tab Stop Scope property of an item group or module component is set to Self or Parent. Form Generator also uses relative tab stops to position items in an item group with its Tab Stop Scope property set to None if the object that contains the item group (i.e. a module component or another item group) has its Tab Stop Scope property set to Self or Parent. In this case, Form Generator positions items as if the item group's Tab Stop Scope property was set to Self.
- Items are always sequenced (for navigation purposes) according to the value of their Usage Sequence property. Relative tab stops are sequenced in their numeric order. If the relative tab setting of the next item in the usage sequence is lower than the relative tab setting of the previous item, Form Generator places the next item on a new line and aligns it appropriately.
- The interaction between relative tab stops and the BLKVFL preference is complex. For more information, refer to "Notes on relative tab stops and BLKVFL" in the Designer online help.
- We recommend you increase the numbers for relative tab stops in increments to allow for the subsequent insertion of new tab stops between existing ones.
- Post prompts cannot be aligned using relative tab stops.
- If you have specified relative tab stops, Form Generator increases the width of the canvas sufficiently to display all the items and item groups. In other words, block overflow does not occur.

Chapter 6 Scenario 3: Migrate, Regenerate Incrementally

This is the most complex scenario. In this scenario, you will migrate your application a little at a time, rather than all at once.

- You will begin by upgrading all of your forms, libraries, menus and reports to Forms 9i.
- You will then make the changes required to run forms generated from your previous release of Designer alongside forms generated from Oracle9i Designer.
- Finally, over some arbitrarily long period of time, you will regenerate all of your modules out of Oracle9i Designer.

The goal of this scenario is to allow you to regenerate your whole application, taking into account new features, but in such a way that you do not have to migrate your entire application in one go. This means you will be able to move the deployed application to the new tool stack before you have completely migrated every form. Thus, you can continue with bug fixes and new development in parallel with the continuing migration effort.

Migrating from 1.3.2

This section covers general migration issues when migrating from Designer Release 1.3.2. If you are migrating from a later release, you may skip this section.

Review use of color palettes

When generating a form, Form Generator determines which color palette to give the generated form as follows:

- If a template form is being used, Form Generator gives the generated form the template form's color palette.
- If a template form is not being used, Form Generator gives the generated form the default Form Builder color palette.

In earlier releases, the template forms supplied with Form Generator did not use the default Form Builder template. Instead, they used a color palette similar to that used in Oracle Applications forms.

In this release, the shipped template forms supplied with Form Generator have been given the default Form Builder color palette.

The color palette consists of a grid, 38 colors wide by 6 colors high. The primary problems are caused in the first 14 columns of the grid. The remainder of the grid is very slightly different, but only noticable to the most discerning eye.

Designer 1.3.2, Custom Color Palette, first 14 columns

Black	White	r0g0b0	r40g0b0	r55g0b0	r70g0b0	r85g0b0	r100g0b0	r0g0b40	r40g0b40	r55g0b40	r70g0b40	r85g0b40	r100g0b40

Control	Blue	r0g40b0	r40g40b0	r55g40b0	r70g40b0	r85g40b0	r100g40b0	r0g40b40	r40g40b40	r55g40b40	r70g40b40	r85g40b40	r100g40b40
Button	Magenta	r0g55b0	r40g55b0	r55g55b0	r70g55b0	r85g55b0	r100g55b0	r0g55b40	r40g55b40	r55g55b40	r70g55b40	r85g55b40	r100g55b40
Canvas	Red	r0g70b0	r40g70b0	r55g70b0	r70g70b0	r85g70b0	r100g70b0	r0g70b40	r40g70b40	r55g70b40	r70g70b40	r85g70b40	r100g70b40
Gray	Cyan	r0g85b0	r40g85b0	r55g85b0	r70g85b0	r85g85b0	r100g85b0	r0g85b40	r40g85b40	r55g85b40	r70g85b40	r85g85b40	r100g85b40
Yellow	Green	r0g100b0	r40g100b0	r55g100b0	r70g100b0	r85g100b0	r100g100b0	r0g100b40	r40g100b40	r55g100b40	r70g100b40	r85g100b40	r100g100b40

Oracle9i Designer, Default Forms Color Palette, first 14 columns

Black	White	Green	Darkgreen	Gray96	Gray92	Gray88	Gray84	r0g0b0	r25g0b0	r50g0b0	r75g0b0	r88g0b0	r100g0b0
Gray	Darkgray	Cyan	Darkcyan	Gray80	Gray76	Gray72	Gray68	r0g25b0	r25g25b0	r50g25b0	r75g25b0	r88g25b0	r100g25b0
Red	Darkred	Blue	Darkblue	Gray64	Gray60	Gray56	Gray52	r0g50b0	r25g50b0	r50g50b0	r75g50b0	r88g50b0	r100g50b0
Yellow	Darkyellow	Magenta	Darkmagenta	Gray48	Gray44	Gray40	Gray36	r0g75b0	r25g75b0	r50g75b0	r75g75b0	r88g75b0	r100g75b0
Custom1	Custom2	Custom3	Custom4	Gray32	Gray28	Gray24	Gray20	r0g88b0	r25g88b0	r50g88b0	r75g88b0	r88g88b0	r100g88b0
Custom5	Custom6	Custom7	Custom8	Gray16	Gray12	Gray8	Gray4	r0g100b0	r25g100b0	r50g100b0	r75g100b0	r88g100b0	r100g100b0

Using both color palettes in the same application causes problems, because the color palette is loaded with the first form and then not reloaded. So, since your start form will be re-generated with the new color palette, **all** of your forms will use the new color palette.

Unfortunately, the colors are mapped by their position in the color palette, not by the actual color name. Thus, when one of your old forms uses a color, it is actually just using whichever color is in that position in the color palette.

So, given the two color palettes above:

If the old form asks for…	It actually gets…
Black	Black
White	White
Control	Gray
Button	Red
Red	Darkyellow
Cyan	Custom2

To further complicate matters, with Webforms, you don't have to specify colors in your forms at all. You can set all of your colors to 'undefined' or to the keyword 'automatic' in your object library. Then, you can pass a runtime parameter to the Forms60 Server that specifies the Look and Feel (Oracle or Generic) and the desired colorscheme. (There are 8 pre-defined color schemes.) At runtime, all items whose color is 'undefined' or 'automatic' will automatically be rendered with the proper colors for the selected colorscheme.

Unfortunately, the item prompts in the old forms that were generated by Designer 1.3.2 are generated as boilerplate and not using the prompt property. This means they will not be automatically changed at runtime.

Changing the Color Palette

You basically have two options to solve this problem:

- Modify the old forms to use the new color palette.
- Modify the new template form to use the old custom color palette.

To be able to change the color palette of an .fmb, Form Builder needs to be opened in a special ‘mode’. To do this, launch Form Builder, navigate to Tools > Preferences, and set ‘Color Mode’ to ‘editable’. Now close Form Builder, and open it again (this step is necessary).

Export the color palette you want to use to a file:

- Open a form which has the color palette that you want to use.
- Open the layout editor by double-clicking on a canvas (it doesn't matter which one)
- Choose File > Export > Color Palette. (If this option is grayed out, you have not successfully changed the “Color Mode” of Forms Builder.) Save the color palette to your file system. The file should have the extension '.pal'.

Import the color palette into the form(s) you want to change:

- Open the form whose color palette you want to change.
- Open the layout editor by double-clicking on a canvas (it doesn’t matter which one).
- Navigate to File > Import > Color Palette. (If this option is grayed out, you have not successfully changed the “Color Mode” of Forms Builder.)
- Choose “File” as Data Source, and click “Browse” to locate the color palette you exported.
- Click OK.

With your old forms, you will have to import the color palette into each form individually even if you used a reference form, because the color palette is not inherited from a reference form.

Unfortunately, we still have one more problem to overcome.

You have successfully installed the new color palette. If you look at the "Foreground Color" and "Background Color" properties for any given item, they appear to have the right values because the color names are correct. But, internally they are STILL pointing to the old position of that color in the color palette.

Even trickier is the fact that if you click on the LOV button next to the color value, the correct color in the color palette is highlighted, but internally it is STILL pointing to the wrong color.

The only way to correct this inconsistent state is to go to the property palette of the visual attribute used on the item, go to the "Background Color" and "Foreground Color" properties, first select any other color, and then change the color back to its original value. Now, not only the color name but also the underlying index value will point to the "true" color in the new color palette.

- The bad news is that you need to do this for ALL explicit color references. If you used a reference form to define visual attributes, and then used these visual attributes in your old template form, you only have to do this for each visual attribute in the reference form. However, if the visual attributes were directly in the template form, or if you set the colors directly in the template form, then you will have to open each form and 'fix' every color reference.
- Two exceptions are the colors "white" and "black". It is not necessary to 'fix' these colors, because they were not relocated between the two palettes.
- Another exception is the color "canvas". Since this color doesn't exist in the new color palette, all occurrences of "canvas" must be replaced with the color "gray".

What this all boils down to is this:

- Unless your old application is very small and you don't mind a lot of tedious work, you are going to have to continue to use the old custom color palette in your newly generated forms
- You probably won't be able to use the automatic color schemes with the new Oracle Look and Feel

Toolbar vs. Smartbar

The Oracle9i Designer menu template makes use of the Smartbar feature of Forms. The toolbar is now defined declaratively in the menu rather than in every form.

In your old forms, you will now see both the Smartbar and the old toolbar canvas.

You have two options:

- Remove the toolbar block and canvas from your old forms.

  If you used a reference form with Designer 1.3.2, you can easily remove the toolbar block and canvas from all your forms by removing it from the reference form and then recompiling new .fmx files.

  If your toolbar block and canvas were directly in the 1.3.2 template form, then you would have to open each generated form and remove them.
- Remove the Smartbar from the new menu template and add a toolbar block and canvas to the new form template (subclassed

from the object library of course).

To remove the Smartbar from the new menu template, you simply change the 'Visible in Horizontal Menu' property to 'No' for each item where it currently is set to 'Yes'.

You can copy your old toolbar block and canvas to your new template form.

Dummy LOV Objects

For modules that call an LOV module (not a native Forms LOV), the Generator creates a dummy LOV object named CGxx$DUMMY_LOV and a corresponding Record Group object with the same name.

In Forms generated from Designer 1.3.2, the record group will have a Column Specification with one dummy column, also named CGxx$DUMMY_LOV, of type character, length = 1, with a default value of (again) CGxx$DUMMY_LOV.

Notice that the default value is longer than the length of the column. This now causes a compile error in forms. To resolve this, you must either clear the default value in the Column Specification of the record group or increase its length to 14.

Unfortunately, this has to be done manually.

Upgrading forms to Forms 9i

You must upgrade all your existing forms to Forms 9i. You can use the Forms compiler to perform this upgrade.

ifcmpxx module= myform userid= scott/tiger upgrade=YES Logon=YES Batch=YES build=Yes Window_state=Minimize

You can run this in batch by creating a .bat file containing the above line of code once for each form to be upgraded.

Regenerate Your Menu and Start Form

You will need to re-generate your menu(s) and start form. This ensures that all your forms run in the context of your new application.

Upgrade Libraries to Forms or Reports 9i

Upgrading the libraries is straightforward. To upgrade manually, you open the library in Form Builder 9i (or Report Builder 9i). Form Builder will automatically start converting PL/SQL version to PL/SQL version 2. During this PL/SQL conversion it will ask you to confirm certain conversion operations. You can safely press the 'Convert All' button if you are prompted for such a conversion operation.

You can also create a .bat file to perform this upgrade in batch. Include the following line of code once for each library to be upgraded.

ifcmpxx module=<library>.pll userid=<user>/<pass>@<connect> module_type=LIBRARY Logon=YES Batch=Yes upgrade=YES version=45 build=No window_state=Minimize

Attached OFG Libraries

Designer release 1.3.2 attached a number of OFG libraries to generated forms. You may also have attached some of those libraries to your custom PL/SQL libraries.

Form Generator does not use these libraries anymore. Form Generator now uses a new, renamed version for each of these libraries.

Designer 1.3.2 Library	**Oracle9i Designer Library**
OFG4BSL	OFGBSL
OFG4CALL	OFGCALL
OFG4HPL	OFGHPL
OFG4MES	OFGMES

OFG4MNL	OFGMNL
OFG4TEL	OFGTEL
	OFGNAVL
	OFGTAB
	OFGTREEN

You must make both sets of libraries available in your runtime environment. The old forms will continue to run with the old libraries until they have been upgraded.

When you regenerate your forms from Oracle9i Designer, they will automatically get the new libraries attached.

However, if you attached the OFG4… libraries to your pl/sql libraries you will have to upgrade each library manually.

- De-attach the obsolete OFG4… libraries from all libraries
- Attach the new OFG… libraries

Name Resolution in 9i

Carefully read the section 'Name Resolution in 9i' in the Chapter 3 "General Migration Issues". It is highly likely you will run into these issues described there if your application has generic library routines that are overridden in module specific libraries.

Replace Designer 1.3.2 forms with Oracle9i Designer forms

This step can be performed on a form-by-form basis. The actions required depend on the type of module. Refer to the chapters on 'General Migration Issues' and 'Scenario 2' for detailed information on regenerating your forms.

If you have a form that calls another form, you should replace both forms at the same time.

Once all forms have been replaced, you can remove the old OFG4… libraries from your runtime environment as well as any other old libraries, reference forms, and templates.

Migrating from 2.1.2

This section covers general migration issues when migrating from Designer Release 2.1.2. If you are migrating from a later release, you may skip this section.

Toolbar vs. Smartbar

The Oracle9i Designer menu template makes use of the Smartbar feature of Forms. The toolbar is now defined declaratively in the menu rather than in every form.

In your old forms, you will now see both the Smartbar and the old toolbar canvas.

You have two options:

- Remove the toolbar block and canvas from your old forms.

  If you used an object library with Designer 2.1.2, you can easily remove the toolbar block and canvas from all your forms by removing it from the object library and then recompiling new .fmx files.

  If your toolbar block and canvas were directly in the 2.1.2 template form, then you would have to open each generated form and remove it.
- Remove the Smartbar from the new menu template and add a toolbar block and canvas to the new form template (subclassed from the object library of course).

To remove the Smartbar from the new menu template, you simply change the 'Visible in Horizontal Menu' property to 'No' for each item where it currently is set to 'Yes'.

You can copy your old toolbar block and canvas to your new template form.

Upgrading forms to Forms 9i

You must upgrade all your existing 5.0 forms to Forms 9i. You can use the Forms compiler to perform this upgrade.

ifcmpxx module= myform userid= scott/tiger upgrade=YES Logon=YES Batch=YES build=Yes Window_state=Minimize

You can run this in batch by creating a .bat file containing the above line of code once for each form to be upgraded.

Regenerate Your Menu and Start Form

You will need to re-generate your menu(s) and start form. This ensures that all your forms run in the context of your new application.

Upgrade Libraries to Forms or Reports 9i

Upgrading the libraries is straightforward. To upgrade manually, you open the library in Form Builder 9i (or Report Builder 9i). Form Builder will automatically start converting PL/SQL version to PL/SQL version 2. During this PL/SQL conversion it will ask you to confirm certain conversion operations. You can safely press the 'Convert All' button if you are prompted for such a conversion operation.

You can also create a .bat file to perform this upgrade in batch. Include the following line of code once for each library to be upgraded.

ifcmpxx module=<library>.pll userid=<user>/<pass>@<connect> module_type=LIBRARY Logon=YES Batch=Yes upgrade=YES version=50 build=No window_state=Minimize

Replace Designer 2.1.2 forms with Oracle9i Designer forms

This step can be performed on a form-by-form basis. The actions required depend on the type of module. Refer to the chapters on 'General Migration Issues' and 'Scenario 2' for detailed information on regenerating your forms.

If you have a form that calls another form, you should replace both forms at the same time.

Once all forms have been replaced, you can remove the old OFG4… libraries from your runtime environment as well as any other old libraries, reference forms, and templates.

Migrating from 6.0

This section covers general migration issues when migrating from Designer Release 6.0.

Toolbar vs. Smartbar

The Oracle9i Designer menu template makes use of the Smartbar feature of Forms. The toolbar is now defined declaratively in the menu rather than in every form.

In your old forms, you will now see both the Smartbar and the old toolbar canvas.

You have two options:

- Remove the toolbar block and canvas from your old forms.

  If you used an object library with Designer 6.0, you can easily remove the toolbar block and canvas from all your forms by removing it from the object library and then recompiling new .fmx files.

  If your toolbar block and canvas were directly in the 6.0 template form, then you would have to open each generated form and remove it.
- Remove the Smartbar from the new menu template and add a toolbar block and canvas to the new form template (subclassed from the object library of course).

To remove the Smartbar from the new menu template, you simply change the 'Visible in Horizontal Menu' property to 'No' for each item where it currently is set to 'Yes'.

You can copy your old toolbar block and canvas to your new template form.

Regenerate Your Menu and Start Form

You will need to regenerate your menu(s) and start form. This ensures that all your forms run in the context of your new application.

Replace Designer 6.0 forms with Oracle9i Designer forms

This step can be performed on a form-by-form basis. The actions required depend on the type of module. Refer to the chapters on 'General Migration Issues' and 'Scenario 2' for detailed information on regenerating your forms.

If you have a form that calls another form, you should replace both forms at the same time.

Once all forms have been replaced, you can remove the old OFG4… libraries from your runtime environment as well as any other old libraries, reference forms, and templates.

Oracle9i Designer Migration Guide
Part 5. Migrating generated Web/PLSQL applications to Oracle9i Designer


Contents

Chapter 1 Introduction

This migration guide provides the information necessary for upgrading Web/PLSQL Applications that were designed and generated using earlier releases of Designer to Oracle9i Designer.

The document discusses migration from the following earlier releases:

- 1.3.2
- 2.1.2
- 6.0

This document assumes that you have already installed Oracle9i Designer and migrated your repository. (See instructions in Part 2 of this Migration Guide.) The document then explains steps you have to take so that you can:

- generate your application from Oracle9i Designer and achieve the same generated results you had from earlier releases, and
- take advantage of new features that have been added to Designer since your previous release.

Throughout the document, special mention is made of any migration issues known at the time of publication of this document.

There are a number of migration scenarios that are possible in bringing your Designer-generated Web PL/SQL application forward into Oracle9i Designer.

Scenario 1. Migrate, Regenerate All, No Redesign

In this scenario, you will regenerate all of your Web/PLSQL application from Oracle9i Designer.

The goal of this scenario is to be able to generate your Web/PLSQL application out of Oracle9i Designer and achieve the same results you got when generating out of your previous Designer release. No attempt is made to redesign your existing Web/PLSQL application to make use of new features available in Oracle9i Designer.

This scenario has the following characteristics:

- If you have minimal post-generation changes, scenario 1 is fast and requires minimal changes in the Web/PLSQL application definition in the Oracle Designer repository.
- Scenario 1 uses your existing module definitions and any custom framesets or templates without changes.

- Scenario 1 does not take advantage of many new Web/PLSQL features in Oracle9i Designer. It is merely a 'technical' upgrade. Some new features will be implemented as part of this upgrade.

This scenario is appropriate when:

- Your application is already in production.
- Your application is stable, no major functional modifications are expected.
- Maintenance is limited to simple bug fixing.
- Your application is 100% generated, or post-generation modifications are minor.

Scenario 2. Migrate, Regenerate All, With Redesign

In this scenario, you will regenerate your entire Web/PLSQL application from Oracle9i Designer. As you regenerate each Web/PLSQL module, you will make use of new features as appropriate.

The goal of this scenario is to take advantage of the new Web/PLSQL features available in Oracle9i Designer. As with Scenario 1, you want to be able to generate your Web/PLSQL application and get the same user interface you got from your previous release of Designer. However, many new Web/PLSQL features have been added to Designer to make achieving the desired result easier. Many features that were difficult or impossible to generate with earlier releases of Designer are now supported. Thus, in one pass, you can eliminate post generation modifications and difficult constructs that were used only to work around limitations of earlier releases of Designer.

This scenario has the following characteristics:

- Scenario 2 requires a review of security implementation.
- Scenario 2 requires modifications in many module definitions, and is therefore more time-consuming.
- Scenario 2 fully leverages the new Web/PLSQL features in Oracle9i Designer.

This scenario is appropriate when:

- Your application is still in development.
- Your application is in production, but major functional modifications are to be made, or expected.
- Your application requires modifications (e.g. security) which can only be implemented using new functionality in Designer.
- Your application has been heavily modified post-generation, but the majority of the modifications are no longer needed due to the new Web/PLSQL functionality in Designer.

Scenario 3. Migrate, Regenerate Incrementally

This is the most complex scenario. In this scenario, you will migrate your application a little at a time, rather than all at once. You will begin by migrating your application to a 8i Database in a schema specific way, or create a database link and appropriate synonyms to your new database schema. You will then make the changes required to run the application generated from your previous release of Designer alongside Web/PLSQL applications generated from Oracle9i Designer. Finally, over some arbitrarily

long period of time, you will regenerate all of your modules out of Oracle9i Designer.

The goal of this scenario is to allow you to regenerate your whole application, taking into account new features, but in such a way that you do not have to migrate your entire application in one go. This means you will be able to move the deployed application to the new tool stack before you have completely migrated every module. Thus, you can continue with bug fixes and new development in parallel with the continuing migration effort.

This scenario has the following characteristics:

- Scenario 3 allows you to perform a phased migration. You can take advantage of the new Designer features right away, without the need to regenerate all your modules at once.

This scenario is appropriate when:

- Your application is in production, but major functional modifications are to be made, or expected.
- Your application requires modifications (e.g. security) which can only be implemented using new functionality in Designer.
- Your application is too large to migrate in one ‘big-bang’.

Scenario 4. Database Migration Only

The first three scenarios all eventually require you to regenerate your application. Any post-generation modifications will be lost. If you heavily modified your application post-generation, and the characteristics of Scenario 1 apply to your situation, you might consider only upgrading the database environment to 9*i*, and not upgrading to Oracle9i Designer. This implies that all future maintenance has to be done manually in PL/SQL.

This migration guide does not cover 9*i* database migration. It will, however, discuss modifications that will allow you to continue to run your generated Web/PLSQL modules alongside other applications. For information on migrating to a 9i database, see Part 3 of this migration guide.

Note that, even though you may choose not to use Oracle9i Designer for continued module generation, you may still use Oracle9i Designer to maintain your database definitions. You may also choose to use the Software Configuration Management features of Oracle9i Designer to manage your application source code. In that case you could bring in the previously generated and adapted packages as files in the repository and maintain these files for future enhancements and/or fixes. Note that in addition you can parse these packages - stored as files - for dependencies with the Dependency Manager for impact analysis.

For information on Software Configuration Management with Oracle9i Designer, see the Oracle Technology Network at http://otn.oracle.com/products/repository.

Note that all scenarios result in an actual usage of Oracle9i and Oracle9i Designer and therefore you will be optimally serviced by Oracle Support on your Oracle tool stack, i.e., you do not need to implement costly workarounds, you can simply profit from regularly released patches.

Chapter 2 Oracle9i Designer New Features

Depending on which Designer release you are coming from, many Web/PLSQL features of Oracle9i Designer may be new to you.

This chapter presents a brief overview of new features that are of particular interest when migrating a generated Web/PLSQL application. It is by no means an exhaustive list of all new features, and it does not try to explain each new feature in detail. Rather, it introduces the relevant features, and points you to where you can find more information in the Oracle9i Designer online help. These new features may be referenced in the appropriate migration steps of the following chapters.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all four sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from Designer 1.3.2

Design Editor

See Part 4, Chapter 3 of this migration guide for a general overview of these new features.

Module Components

See Part 4, Chapter 3 of this migration guide for a general overview of these new features.

Preferences

Oracle9i Designer has added many new preferences. These preferences allow for greater flexibility in layout and provide for new methods of security.

TAPI/Triggers

Designer 1.3.2 introduced the concept of a Table API (TAPI) specifically for the WebServer Generator. The WebSever Generator used the TAPI to do any inserts, updates or deletes (and some selects) against a table. The TAPI auto-generated primary keys associated with sequences and audit columns, and allowed you to place your own specific code within the TAPI. The TAPI was not used with any other Designer generator. Beginning with Designer 2.1.2, the TAPIs became an integral part with other generators and were associated with table triggers, causing the TAPI code to be run during any insert, update or delete. Oracle9i Designer has additional custom code placement options within the TAPI and auto-generates more functionality, including enforcing arcs, etc.

Migrating from Designer 2.1.2/6.0

This section describes migrating from Designer 2.1.2 or 6.0.

LOV components

Oracle9i Designer departs from the use of lookup table usages to drive LOVs. LOVs can now be created as specific, reusable components and attached to an element. This provides for more flexibility and the ability to drive the LOV from a custom query. This is particularly helpful in the WebServer Generator in that it allows you to better define the look of Poplist type LOVs.

All_domains Table

Before Oracle Designer 6i, the CG_REF_CODES table provided the only structured reusable table for multiple lookup types. This limited its usefulness to only lookups that fit within the column structure of CG_REF_CODES. The All Domains table feature extends this type of functionality to a table that can be custom designed to meet the needs of your application. Coupled with Reusable LOV components this becomes a very powerful new feature.

Reusable Modules

Reusable Modules allow you to define a module with base table usages, associated lookups, etc., and to reuse this definition in multiple generated modules. This provides enhanced standardization.

Preferences

Oracle9i Designer adds substantially to the preferences available for WebServer Generator modules. These preferences provide better control over layout and provide the ability to maintain “context” between modules. Because the web is stateless, information that may have been available in a previous module in the hierarchy may not be available in the current module. The Master Context section of the generator preferences provides a way to ensure context is available to subsequent modules. Please see the Oracle9i Designer online help for more information on new preferences.

Multi-Row Screens

Oracle9i Designer will generate multi-row screens to provide inserts, updates and deletes across multiple rows. This is determined at the module level.

Oracle Portal portlets generation

You are now able to generate more functional Oracle Portal portlets. Please see the Oracle9i Designer online help for more information on this feature.

Chapter 3 General Migration Issues

This section will review issues that may arise with migrating from any previous release of Designer to Oracle9i Designer. The following sections will deal with specifics of each release. For completeness, you may wish to read the sections regarding earlier releases than your own.

There are a number of actions you must take regardless of which migration scenario you choose.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from 1.3.2, 2.1.x, 6.0

This section covers general migration issues when migrating from Designer Release 1.3.2, 2.1.x and 6.0.

Perform a backup of your database schema that captures the Web-PL/SQL components

In all cases perform a complete backup of your database and export of any affected schemas.

Web PL/SQL Generator Libraries

You will need to upgrade the associated libraries for the Web PL/SQL generator (wsgl, wsglm, cg$errors, etc.). Because you will be regenerating all components, there is no need to run the existing version of these packages alongside the new versions. In other scenarios you will be required to maintain both sets of packages.

WSGLM is often modified to provide messages tuned to your applications. Any changes to your existing WSGLM will need to be re-applied to the Oracle9i Designer WSGLM package.

Occasionally changes might have been made to other library packages. Some of these changes may need to be reapplied, where others may not be needed due to new functionality within the product. For example, Designer 1.3.2 matched case on queries. The dynamic build_where_clause could be modified to overcome this. In Oracle9i Designer there is an option to perform case insensitive queries (the default is case insensitive).

Depending upon the number of existing applications, you might have multiple copies of the WSGL libraries. By default, Oracle9i Designer will install these libraries in one location (schema). If the schema has sufficient privileges, the installation will grant execute on these libraries to public and create public synonyms. In some cases you may prefer to have multiple copies of the libraries for use by different application systems. This can be done by installing the copies in separate schemas and pointing the application schemas to the appropriate library schema by use of private synonyms.

Security

Designer 1.3.2 and 2.1.2/6.0 were typically run using either the Oracle Application Server PL/SQL cartridge or the WebDB listener. In both cases you had two methods of configuring the database login to run the generated PL/SQL components:

1. single user login, and
2. database authentication (provide a basic authentication login screen and login to the database with the supplied username and password).

In Oracle9i Designer, there are new security features that should be considered. In Scenario 1 we assume that you will use your existing access and security methods. The new features are discussed in Chapter 5 "Migrate, Regenerate All with Redesign".

Regardless of which scenario you adopt, you will need a good understanding of your existing access and security mechanism. At a minimum you should determine your DAD settings and note any schemas and roles that are associated with securing your existing PL/SQL packages that comprise Web PL/SQL modules.

Web PL/SQL Generator uses a checksum routine to secure access to specific records. Regardless or the version of Designer, it is a good idea to change the function WSGL.CHECKSUM slightly so that your checksum is different than the default shipped. The easiest method is to change the mod number in the function, WSGL.CHECKSUM, from 4294967296 to some other large number. As you will be installing a new version of the libraries, specifically WSGL, previous changes to WSGL.CHECKSUM must be reapplied.

Application Server Choice

Designer 1.3.2, and 2.1.2/6.0 WebServer Generator applications were typically run with the OAS PL/SQL Cartridge or the WebDB Listener.

Although you can continue to use your existing Application Server, Oracle9i Designer applications are likely to be run with Oracle 9i Application Server using MODPLSQL. In general, the configuration of web pl/sql modules is very similar.

The DAD configuration information can be found in the file $ORACLE_HOME/Apache/modplsql/cfg/wdbsvr.app. Beginning with Oracle 9iAS 1.0.2.2 the DAD password is encrypted within this file. The browser interface to configure this file is found at http://yourmachine:port/pls/admin_/gateway.htm where your machine is the machine name where Oracle 9iAS is installed.

After installing Oracle 9iAS, make a copy of the file $ORACLE_HOME/Apache/Apache/conf/httpd.conf

Modify this file and change the line

defaultType text/ascii

to

defaultType text/html

The application server will then treat unknown content type as html. On rare occasions, with defaultType set to text/ascii, the html code generated by Designer WebServer Generator will be displayed as ascii text rather than an html page.

Chapter 4 Scenario 1: Migrate, Regenerate All, No Redesign

In this scenario, you regenerate your entire application from Oracle9i Designer, including all Web/PLSQL, libraries, menus and reports.

The goal of this scenario is to be able to generate your application out of Oracle9i Designer and achieve the same results you got when generating out of your previous Designer release. No attempt is made to redesign your existing application to make use of new features available in Oracle9i Designer.

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

This chapter assumes that you have performed all the actions described in Chapter 3, General Migration Scenarios.

Migrating from 1.3.2

In most cases, migrating your 1.3.2 application to 9i will go smoothly. There are four specific areas that may cause difficulty: module level security (z_chk), using calls to underlying values, return links and Table APIs.

Module Level Security

Designer WebServer Generator modules use “Query, List, Detail”. A user is first presented with a Query Screen, then a list of items that match the criteria, and ultimately a detail screen that typically allows update. In many cases, the list screen will always be limited by a fixed where clause in the module. Releases of Designer 2.1.2 onwards implement a security check to ensure that users do not access the detail screen directly by changing the URL. Designer uses a checksum value (Z_CHK) to enforce this. Every link on the list screen has Z_CHK=#####. The detail page checks to see that this checksum value is correct.

If you have defined your own method of calling a component based upon the URL, you will need to modify these calls to implement Z_CHK or turn off Z_CHK. You can turn off Z_CHK by setting the preference SECECS (Enforce URL Checksums) to No. Please see the Oracle9i Designer online help for specific information about how to call wsgl.checksum to implement Z_CHK from your own code.

Note: It is a good idea to change the function WSGL.CHECKSUM slightly so that your checksum differs from the default shipped. The easiest method is to change the mod number in the package (4294967296) to some other large number.

Using Calls to Underlying Values

In all releases of Designer you could reference values from routines placed in the user text area. This was not explicitly stated in release 1.3.2 and there were no APIs or standards given for doing so. Hence, calls made to these underlying values, by use of form_val.field_name, etc., will likely fail in the migrated application. Oracle9i Designer specifies the use of these values. Please see the Oracle9i Designer online help for more information on accessing these values.

Return Links

Designer 2.1.2 onwards provides return links to allow the user to select a link to return to previous modules in the hierarchy. Because this was not available in earlier releases, the existing application may have coded return links in the user text area. You can turn off return links with the preference MODBRL (Build Return Hyperlinks).

Table APIs

Designer 1.3.2 required Table APIs (TAPIs) only for WebServer Generator applications. With Oracle9i Designer the default is to have all transactions run the logic present within a TAPI. Hence, any code that is currently in the TAPIs will be run for all transactions regardless of the application. This is implemented through the use of table level triggers. You can disable this feature by deselecting the Generate Table API Triggers checkbox in the Generate Table API Definitions dialog box during generation. Other types of applications, however, may wish to make use of the TAPIs, hence it is a better practice to move any code specific to Web PL/SQL modules out of the TAPIs and into the modules themselves. Oracle9i Designer has many new trigger points within the module to accept this code.

Migrating from 2.1.2/6.0

In most cases, migrating your 2.1.2/6.0 application to Oracle9i Designer will go smoothly. There are two specific areas that may cause difficulty: using calls to underlying values, and table APIs.

Using Calls to Underlying Values

In all releases of Designer you could reference values from routines placed in the user text area. In many cases the format of calls or local access to these underlying values may have changed; calls made to these underlying values, by use of form_val.field_name, etc., will likely fail in the migrated application. Oracle9i Designer specifies the use of these values. Please see the Oracle9i Designer online help for more information about accessing these values.

Additionally, Oracle9i Designer provides a preference for bringing master level information into detail

blocks of a module. These preferences can be found in the Master Context area of the Web PL/SQL Generator preferences.

Table APIs

Designer 2.1.2/6.0 virtually insisted that you install triggers that implemented the TAPIs for all transactions. Oracle9i Designer allows you to bypass generation of these triggers. In general it will not affect your Web PL/SQL applications if you do not generate these triggers. It may, however, allow the data to become inconsistent with what is expected from your application. Before deselecting the Table API Triggers checkbox in the Generate Table API Definitions dialog box during generation, consider carefully the affect of not having the TAPIs fire for all transactions .

Chapter 5 Scenario 2: Migrate, Regenerate All, with Redesign

In this scenario, you regenerate your entire application from Oracle9i Designer, including all Web/PLSQL, libraries, menus and reports. As you regenerate each module, you make use of new features as appropriate.

The goal of this scenario is to take advantage of the new features available in Oracle9i Designer. As with Scenario 1, you want to be able to generate your application and get the same user interface you got from your previous release of Designer. However, many new features have been added to Designer to make achieving the desired result easier. Many features that were difficult or impossible to generate with earlier releases of Designer are now supported. Thus, in one pass, you can eliminate post generation modifications and difficult constructs that were used only to work around limitations of earlier releases of Designer.

This chapter assumes that you have already performed all the actions described in Chapter 3 "General Migration Issues" and that you have evaluated the steps in the previous chapter, "Scenario 1 Migrate, Regenerate All, No Redesign".

This chapter is organized by Designer release. You should begin reading at the section for your "from" Designer release, and then continue reading the sections for any later releases. For example, if you are migrating from Designer 1.3.2, you need to read all the sections below. If you are migrating from Designer 2.1.2, you may skip the section on Designer 1.3.2 and read the sections for Designer 2.1.2 and 6.0.

Migrating from 1.3.2

Please review Chapter 4 "Scenario 1: Migrate, Regenerate All, No Redesign", because this also applies to this section.

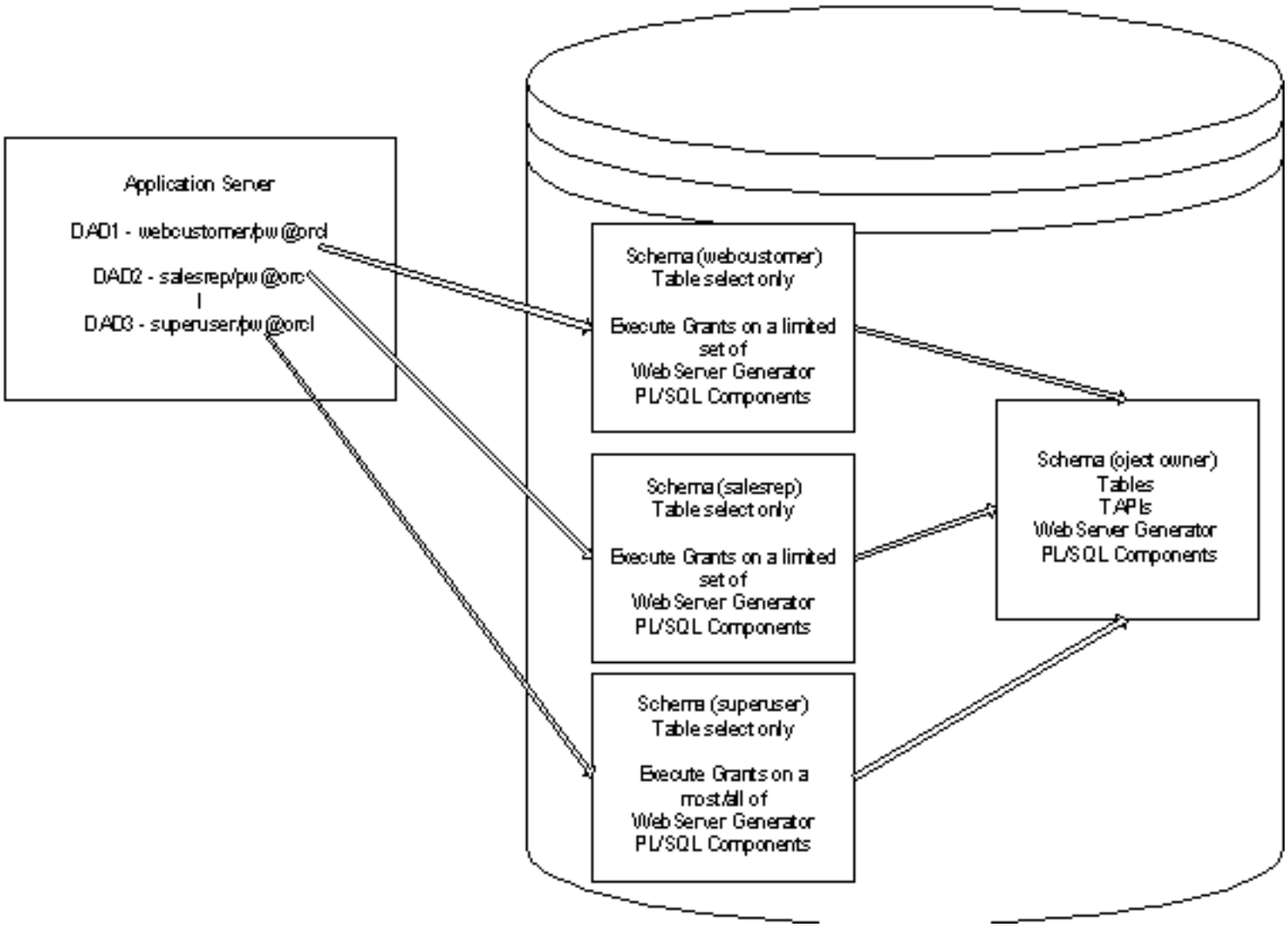
Security

This section is an extension of the security section in Chapter 3 "General migration issues".

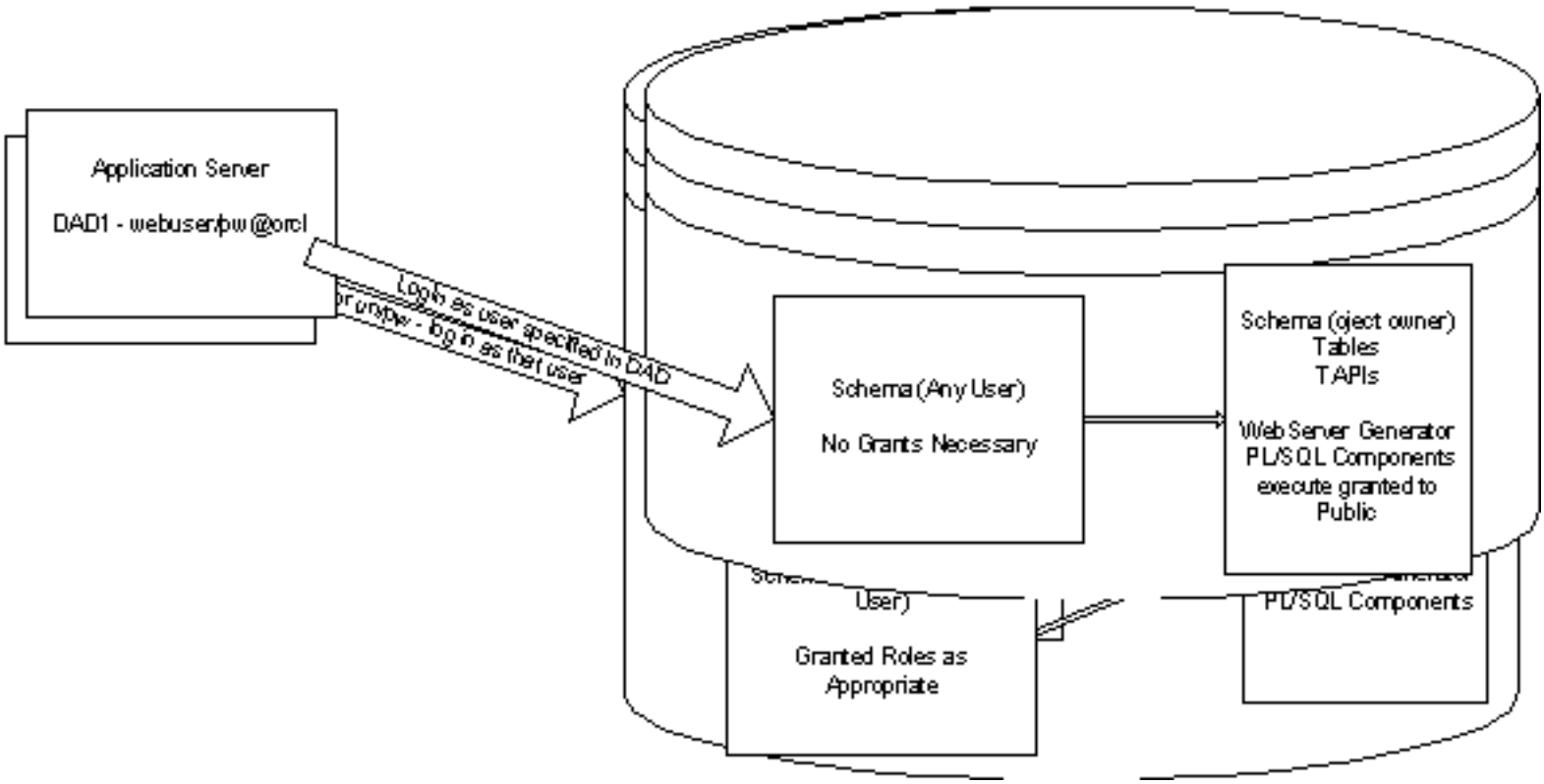
Designer 1.3.2 and 2.1.2/6.0 were typically run using either the Oracle Application Server PL/SQL cartridge or the WebDB listener. In both cases you had two methods of configuring the database login to run the generated PL/SQL components:

1. single user login, and
2. database authentication (provide a basic authentication login screen and login in to the database with the supplied username and password).

The first scenario, single user login, may have been extended to have multiple Database Access Descriptors (DADs) that pointed to specific database schemas.



The second scenario, database authentication, allowed the user to log in to the database as a specific database user. Grants to specific PL/SQL components could then be made to a role, which would in turn be granted to the database users.



In Oracle9i Designer, there is a new security feature that should be considered. The preference SECPKG (Security Package Name) allows you to specify a PL/SQL package that will determine the users' ability to use a component. (Please see the Oracle9i Designer online help for details of SECPKG.) By using SECPKG you can simplify your deployment architecture to that shown below.

You can now grant execute on all PL/SQL components to PUBLIC. Your custom developed SECPKG package will return a true if the user is allowed to run the component, based upon your own criteria.

Note: You can continue to run the generated application exactly as you did before, making use of the default SECPKG which always returns true.

If you are using WebServer Generator in the same database instance as Oracle Portal, you can use all of the Portal user and group features. You must run the Portal script PROVSYNS.SQL for the schema that contains your SECPKG package. You can then reference the Portal lightweight user name with a call to the Portal API wwctx_api.get_user. Additional Group and User API information is available within the Portal PDK (http://portalstudio.oracle.com).

In Scenario 1 we discussed four areas of concern: module level security (z_chk), using calls to underlying values, return links and table APIs. While these areas of concern remain, they also provide enhancements that may allow you to reduce the need for post-generation changes or custom code within the user text areas.

Module Level Security

Oracle Designer Web PL/SQL Generator modules use “Query, List, Detail.” A user is first presented with a Query Screen, then a list of items that match the criteria, and ultimately a detail screen that typically allows update. In many cases, the list screen will always be limited by a fixed where clause in the module. Designer 2.1.2 and forward implement a security check to ensure that users do not access

the detail screen directly by changing the URL. Designer uses a checksum value (Z_CHK) to enforce this. Every link on the list screen has Z_CHK=#####. The detail page checks to see that this checksum value is correct.

In the previous scenario we discussed how to turn off this feature to provide the same functionality as your existing application. In this scenario we will discuss how to make use of this feature.

In Designer 1.3.2 you may have "tricked" the generator into using a view instead of a base table, thereby not allowing the user access to the underlying data in the table. With Z_CHK this trick is no longer necessary. By setting the generator preference SECECS (Enforce URL Checksums) to Yes the generated module will ensure that users only have access to the detail screen via the record list. (Please see Scenario 1 for more details.) Additionally, if you wish to access any detail records from other modules you will need to make use of the wsglm.checksum routine. Please see the Oracle9i Designer online help for more information regarding wsglm.checksum.

Return Links

In Designer 1.3.2 navigation links were largely the responsibility of the developer. Designer 2.1.2 forwards provides links to components higher in the module hierarchy.

Table APIs

Table APIs were introduced in Designer 1.3.2 specifically for the Web PL/SQL generator. From 2.1.2 forward TAPIs are an integral part of the overall system. TAPIs have been extended to allow additional points of access to add custom code. TAPIs automatically generate code to provide denormalization and enforcement of constraints such as arcs, domain based keys (CG_REF_CODES and ALL_DOMAINS), etc.

Migrating from 2.1.2/6.0

Please review Scenario 1: Migrate, Regenerate All, No Redesign, because this also applies to this section.

Module Level Security

Designer 2.1.2 left security largely up to the developer. Security was generally implemented by creating a schema that either owned the pl/sql packages that make up the Web generator module or had specific grants to those packages. Another option was to create roles with grants to pl/sql packages and ultimately grant those roles to users. In either case, the security was left to the DBA to developed and was limited to granting access to pl/sql via database grants.

Oracle9i Designer provides a mechanism to define your own security policy. This policy is still implemented across the whole module, but it allows you to code the decision of access in any way you choose. Every procedure that is callable from a web browser first calls the package defined in the preference SECPKG, passing in the package name. The default SECPKG always returns true. By coding your own SECPKG routine you can base access on anything you like - it is not limited to

database grants. Perhaps some modules should only be run after normal business hours, simply return false when called between 8 and 5. Additionally, when SECPKG returns false, nothing is returned to the browser. This allows you to code your own access error screen. You also have the ability to log invalid (or valid) attempts to run a module. Each call to a module component initiates a call to your security routine, giving an access point to log information regarding the module.

When the generator installs the module component, it will call a procedure, add_package_resp, in the package defined in the preference SECPKG, passing in the value of SECRES (Default Responsibilities) and all PLSQL Packages that make up the module. Your procedure, add_package_resp, can then insert these values into a table that will later be used to determine whether a user can run the package.

Multi-Row Components

Designer Web PL/SQL components in version 1.3.2 were limited to insert, update and delete a single row at a time. Oracle9i Designer now provides the ability to insert, update and delete multiple rows in a single html screen. Forms that were of type List/Form should be reviewed to determine if the new multi-row format will improve the user interface.

Oracle Portal portlets generation

Oracle9i Designer provides new functionality to integrate with Oracle Portal. Oracle9i Designer will generate the wrappers required to present the screens as portlets within Oracle Portal. Because Oracle Portal and Designer Web PL/SQL generator modules both use Oracle 9i Application Server's modplsql, great synergy can exist between these technologies.

If you are using WebServer Generator in the same database instance as Oracle Portal, you can use all of the Portal user and group features. You must run the Portal script PROVSYNS.SQL for the schema that contains your SECPKG package. You can then reference the Portal lightweight user name with a call to the Portal API wwctx_api.get_user. Additional Group and User API information is available within the Portal PDK (http://portalstudio.oracle.com). By using the PDK you can determine what groups a user belongs to and map these groups to Designer Modules.

Oracle Portal provides great flexibility in navigation and presentation. By generating Portlet wrappers, you can place Designer modules on tabbed portal pages. Portal modules can be integrated on the same pages as Oracle Portal components such as menus, reports and charts.

Chapter 6 Scenario 3: Migrate, Regenerate INCREMENTALLY

This is the most complex scenario. In this scenario, you migrate your application a little at a time, rather than all at once. You begin by upgrading all of your Designer repository to an Oracle9i Designer repository. You then make the changes required to run Web/PLSQL generated from your previous release of Designer alongside Web/PLSQL generated from Oracle9i Designer. Finally, over some arbitrarily long period of time, you regenerate all of your modules out of Oracle9i Designer.

The goal of this scenario is to allow you to regenerate your whole application, taking into account new features, but in such a way that you do not have to migrate your entire application in one go. This means you will be able to move the deployed application to the new tool stack before you have completely migrated every form. Thus, you can continue with bug fixes and new development in parallel with the continuing migration effort.

The basic method that allows you to run multiple versions at the same time is the same regardless of the versions. You should read all of Chapter 3 and Chapter 4 before attempting this solution. You should back up any schemas that hold database objects, wsgl libraries and application components prior to attempting to reconfigure the system to allow multiple versions to run side by side.

The key to this method is segregating the different versions into their own schemas. Typically all of the database objects and the Web PL/SQL components might have resided in a single database schema.

In order to run your existing application alongside the new Oracle9i Designer application components you will need to first migrate your existing application and associated Web PL/SQL generator libraries to a separate schema. In all cases you will need to grant select, insert, update, and delete on all database tables (including CG_REF_CODES) and sequences that are used by your modules. These grants must be granted directly, not via a role. Depending upon the version that you are migrating from, you may need to create synonyms to point to the database objects (Designer 6i introduced the ability to define the database object schema within the generator preferences. Earlier releases require synonyms.)

After moving all existing application components to a new schema as described above, the system should work precisely as it had previously.

Next, you can install the Oracle9i Designer Web PL/SQL generator libraries. This can be done in the object owner schema or in a separate schema. For the purposes of this document we will assume that these libraries are in a separate schema.

Again, the original application should be tested at this point to ensure that it continues to run properly. The table APIs should be installed next. In Oracle9i Designer, TAPIs are intended to be usable by transactions other than Web PL/SQL modules. If you only intend to use TAPIs for Web modules, the TAPIs can be placed in the schema in the diagram above. We will assume that you will use TAPIs throughout and hence will place the TAPIs in the object owner schema. In so doing, you must grant execute on the TAPIs to the schema.

You can now migrate individual components to Oracle9i Designer and install them into the schema. Any logic placed in the TAPIs will be run for all transactions, including transactions from earlier Web PL/SQL modules.

Designer Web PL/SQL modules do not preface calls to the components with the schema name. Hence, depending upon your application server configuration, you will likely need to create public synonyms for all of the packages associated with your applications. This requires that your module names be unique across schemas.

While this configuration is the most complicated, it is based upon sound database techniques and can be accomplished. Many specific scenarios will require a more complex scheme than that described above.

Prev Next

Contents

Oracle9i Designer Migration Guide
Appendix A


Contents

PVCS/VM administration query

```
select distinct obj.vmproject  project
,      obj.workfile   workfile
,      obj_d.revision Develop
,      obj_t.revision Test
,      obj_a.revision Accept
,      obj_p.revision Prod
from   (select log1.vmproject
        ,      log1.workfile
        ,      grp1.revision
        from   solvctgrps grp1
        ,      solvctlogs log1
        where  grp1.archiveid  = log1.archiveid
        and    grp1.vmproject  = log1.vmproject
        and (grp1.groupname    = 'develop'
             or grp1.groupname = 'test'
             or grp1.groupname = 'accept'
             or grp1.groupname = 'prod'
            )
       ) obj
,      (select log2.workfile
        ,      grp2.revision
        from   solvctgrps grp2
        ,      solvctlogs log2
        where grp2.archiveid = log2.archiveid
        and   grp2.vmproject = log2.vmproject
        and   grp2.groupname = 'develop'
       ) obj_d
,      (select log3.workfile
```

```
        ,       grp3.revision
        from    solvctgrps grp3
        ,       solvctlogs log3
        where   grp3.archiveid = log3.archiveid
        and     grp3.vmproject = log3.vmproject
        and     grp3.groupname = 'test'
        ) obj_t
,       (select log4.workfile
        ,       grp4.revision
        from    solvctgrps grp4
        ,       solvctlogs log4
        where   grp4.archiveid = log4.archiveid
        and     grp4.vmproject = log4.vmproject
        and     grp4.groupname = 'accept'
        ) obj_a
,       (select log5.workfile
        ,       grp5.revision
        from    solvctgrps grp5
        ,       solvctlogs log5
        where   grp5.archiveid = log5.archiveid
        and     grp5.vmproject = log5.vmproject
        and     grp5.groupname = 'prod'
        ) obj_p
where   obj.workfile = obj_d.workfile(+)
AND     obj.workfile = obj_t.workfile(+)
AND     obj.workfile = obj_a.workfile(+)
AND     obj.workfile = obj_p.workfile(+)
order by obj.workfile
 n      /
```

Oracle9i Designer Migration Guide
Appendix B. Quick reference information


Contents

Repository Terminology quick reference information

Application (Container)

A type of Container that groups Structured Elements with a logical affinity to each other. This affinity may be data related (that is, all data elements for <App System Cluster>) or function related. An Application may container Folders to hold related File Elements and Documents (that is, ins and ddl).

Application Item

A type of Configuration Item that is the direct source for an application executable element (that is, Module Definition, Report Source File). These generate out as Developer, PL/SQL, etc. source or are extracted “as is”.

Application System Cluster

A group of application systems that are strongly interrelated. For example a set of entities and tables from application system CORE is referenced by the application systems SALES and STOCK. You would typically group these application systems in a single workarea.

Base line configuration or release

A base line configuration - or base line release - represents a full set of object definitions for a specific system. For example base line configuration <App System> 3 represents release 3 of system <App System> and contains all elements within <App System> that should be deployed for release 3.

Branch

Branches are sequences of object versions that have their starting point at a particular version in an existing branch, but evolve independently. All the versions together create a version tree. A version tree always has a MAIN branch, and may also have sub-branches. Branching enables you to develop multiple versions of a product simultaneously (parallel development) using the same set of source files.

When you check in an object for the first time, it is always added to the default MAIN branch. If the repository policy specifies an alternative branch to check in to for the first time, two versions are created, one on MAIN and the other on the specified alternative branch.

Checkin

The checkin option versions an uploaded file/folder or a saved structured element. For example, the file or folder is already stored in the repository and in addition it will be enhanced with version information and you will be able to view the file/folder/structured element in another workarea. Note that another version of an element can only be created via a checkin operation.

Checkout

Repository elements can be manipulated only when they are checked out (or unversioned). You therefore first have to check out a checked in element before you can apply changes. The checkout action starts with the creation of a duplicate of the element. This allows you to perform an undo checkout operation. As a result you will fall back on the original, checked in element version.

Configuration

A configuration is another means to isolate only one version of one or more repository elements. The content of a configuration is fully static, in contrast to workareas. Like any other element you can version configuration. This version capability of configurations can be used to associate configurations with a workarea or, to put it differently, to base a workarea on one or more configurations. Another application of the configuration version capability is the storage of an original element set in the “first” version of the configuration. Within the next version all corresponding derived elements (ddl files) are stored.

Configuration Item

The product of a task, or a deliverable for the project, which is placed under Configuration Management.

Container

A grouping of items for the purpose of usage, access, or extraction. Within the Oracle Repository containers can be Applications (for example, DMO, Headstart) or Folders (for example, ins, ddl). Containers should not be confused with Workareas (see below). Workareas are a means of presenting containers and their contents.

Data Content Item

A type of Configuration Item that is the direct source for the initiation of a database schema element (that is, seed data script). These are extracted “as is” for use.

Data Management Item

A type of Configuration Item that is the direct source for the restructuring, conversion, or initiation of a database schema element (that is, restructure table script). These are extracted “as is” for use.

Data Structure Item

A type of Configuration Item that is the direct source for a database schema element or a database schema logic element (that is, Table or View Definition). These generate out as DDL scripts.

Database Schema

A database schema is a user in the database. Each database element like a table or view (expect a role) is owned by a database schema.

Database Synchronization

Database Synchronization refers to the state of database definitions in the repository versus its mapped database objects in the database within a database schema. There can be a mismatch between these two, since there are (within the context of a workarea) two database representations, one in the repository and one in the database. The database synchronization status can be checked with the CheckRelease Form (see CheckRelease QRC).

Database User

See database schema. Note that each database user can access its own database objects - obviously. In addition a database user can access database elements of other users - or schema’s - via a mechanism of grants and synonyms.

Document (Item)

Any type of Configuration Item which is part of the supporting description an application system’s architecture, environment, usage, or its general definition.

Download

The download option allows you to publish files and/or folders from the repository to a specific folder path on the operating system - either based on the default folder mapping or session specific value. The download will prompt you if the file already exists on the file system.

Edit Workarea Rule Specification

The content of a workarea is primarily determined by workarea rules (see above). You need a specific workarea access right (Update Spec) to change the workarea rules for a workarea. This Update Spec policy is given only to a PCM. This Update Spec right must be distributed in combination with the Refresh or Compile right (see below).

Element

Any type of Configuration Item (see above), intermediate item (not under configuration control), executable item (for deployment), or other component which is associated with the development and deployment of the application system that is being constructed or maintained. An element can be either structured or unstructured (file).

Environment

A specific set of elements drawn from across each of the technology layers. An environment is identified as being for a particular promotion state or purpose, such as application development or application deployment. An environment will include a specific Baseline Configuration, a directory structure containing source and executable programs, a specific database schema containing data structures and data content, and an accessible toolset.

File Element

A type of Application Element which may or may not have been originally sourced from Designer, but whose current source is the file system representation, (for example, a report with post-generation changes, or data conversion scripts written by hand).

File Synchronization

File Synchronization (not to be confused with the synchronization option, see above) refers to the state of files in the repository versus its mapped files on the file system. There can be a mismatch between these two, since there are (within the context of a workarea) two file representations, one in the repository and one on the file system. The file synchronization status can be checked with the CheckRelease Form (see CheckRelease QRC).

Folder (Container)

A type of Container that groups File Elements with a logical affinity to each other. They may be related by usage, project phase focus, or other criteria. Top level Folders are usually created as children of Applications and may contain other Folders in turn. A standard set of Folders has been identified for general use in all Applications (that is, ins and ddl).

Folder Mapping

Folders - containers for files - in the repository can be mapped against directories on the file system via the folder mapping option (available for root containers only). This mapping - stored locally in the registry - is reused while downloading and uploading files (see below).

Item

See Configuration Item.

Item Type

The classification of an Item based on its definition, characteristics, or use (see below).

(CM) Repository

A storage system which contains a copy of all the items under Configuration Management control, and also contains the information that relates those items to each other and to project deliverables. This facility is provided by the Oracle Repository, which is implemented as a specific database schema, that underlies the Oracle Designer toolset. The CM repository contains for example all <App System> entities, tables, views and files and subsequent versions.

Root Container

A root container (application system or folder) is the starting point of a container structure like a root directory on the file system.

Partial configuration

A partial configuration contains only a limited set of object definitions for a specific release. A partial configuration can be associated with a single registered enhancement request (“Change Request”) or with a bundle of registered enhancement requests. For example partial configuration <App System>_20013011 represents enhancement request <App System>_20013011 for system <App System> and contains only objects that are associated with this enhancement request.

Refresh a Workarea

The refresh or compile option of a workarea (another specific workarea access right) reevaluates the content of a workarea based upon the workarea rules, without changing the workarea rules themselves. The end-result (pointers) is stored in the workarea table.

Requery a workarea

The requery does not reevaluate the workarea rules, but takes the persistent workarea table content as a given and requeries all primary and secondary elements, associations and properties.

Sandbox Workarea

A sandbox - or playground - workarea can be used for the preparation of releases or to experiment with workarea rules. The use of a separate workarea during the preparation of a release circumvents the interference with ongoing development and testing activities.

Specific Workarea Operations

The following operations can be executed against a workarea:

- Change the workarea rules or edit workarea rule specification.
- Refresh a workarea.
- Requery a workarea.

Structured Element

A type of Application Element that is defined wholly within the Designer toolset and is then generated out from there for use in the application system (that is, Entity, Table Definitions, View Definitions)

Synchronization

The synchronization option (files only) compares the repository content with the underlying file system, for a specific directory tree. Subsequently it will suggest uploads and downloads for file mismatches. This option will however be rarely used because files are downloaded or uploaded on a individual basis during checkin and checkout procedures (see Checkin and Checkout QRC).

Target or destination file system

The target or destination file system is the environment on the file system level that will receive the files for a specific system (DMO) captured in a specific release.

Target or destination database schema

The target or destination database schema is the schema in the database (in the specific target database instance) that captures the database objects of one or more systems (for example, <App System>, OHL) of a specific release (or configuration).

Target or destination workarea

The target or destination workarea is the environment in the repository that will receive the release, captured in a configuration. The target or destination workarea, given the (D)TAP model, could be Test (for example, <App System>_TST_REL<release nr>), Acceptance test (for example, <App System>_ACC) or Production workarea (<App System>_PRD).

Tip Version

The latest object version of a branch.

Upload

The upload option allows you to store files and/or folders in the repository from a specific file system path, either based on the default folder mapping or session specific value. Note that an upload does not check in the element. That is, no version properties are added and you cannot view a stored file/folder in another workarea.

Verification of the Synchronization between the Repository versus File and Database

The CheckRelease Form reports about the synchronization status of repository elements versus its file representatives on the file system and its database representatives in the database (see also CheckRelease QRC).

Version

The rendering of an item which incorporates all of its revised content starting from a given point (that is, from a previous version).

Workarea

A workarea serves as an access vehicle or environment to the repository content. Within a workarea you can see (and manipulate with appropriate access rights) only one version of one or more objects, grouped in a container structure. A workarea therefore provides a version resolved access mechanism to the repository elements. A workarea only contains pointers to the full definition of the element - it does not capture the full element definition.

The content of a workarea can change over time, for example via checkin and checkout actions. That is, a workarea content is dynamic. Almost all Oracle SCM and Oracle Designer tools (except the repository object navigator) - allow you to access the repository elements only in the context of a workarea. That is, you cannot access the Design Editor in the context of a configuration.

Note that there is, luckily, no need to version a workarea. You cannot therefore version a workarea.

Workarea rules

The content of a workarea is primarily determined by workarea rules. A workarea can contain multiple workarea rules (for example, LATEST(MAIN), INCLUDE_FOLDER(<App System>)) that are evaluated one by one (from top to bottom). The evaluation result (pointers to the full element definition) is subsequently stored in a persistent way (in a workarea table).

Designer/Repository Tools Quick Reference Information

Repository Object Navigator (RON)

The RON or Repository Object Navigator provides a tree-navigational interface to the repository elements (structured elements and files). Subsequently the RON hosts all configuration management functionality like the manipulation of workareas, configurations and branches, file manipulation, and merge utilities.

Design Editor (DE)

The DE or Design Editor also provides a tree-navigational interface and a graphical interface to the structured design repository elements like tables, views and sequences. In addition, it hosts the DDL generators. Note that these generators are no longer available in the RON.

Version History Viewer (VHV)

The Version History Viewer displays a graphical interface of all versions of a specific element. You can launch the VHV from the RON or DE.

Version Event Viewer (VEV)

The Version Event Viewer displays a character-based interface of all versions' history information (for example, checkin/checkout notes, the owner of the changes, when changes took place) of a specific element. You can launch the VEV from the RON or DE.

Element Compare Tool

The element compare tool displays a detailed overview of the differences between two element versions. You can launch the element compare tool either from the RON or DE.

Element set compare tool

The element set compare tool displays an overview of the element set differences, differences in version numbers, for example the differences in element set memberships of two workareas or configurations.

You can launch the element set compare tool only from the RON.

Element merge tool

The element merge tool merges differences between two object versions on different branches on a conflict by conflict basis. For example you can merge object version 1.2.1.1 into MAIN branch label 1.4 resulting in version 1.5.

You can launch the merge tool from the VHV.

Merge Wizard

The merge wizard allows you to perform a merge for a set of elements.

For example you can merge all elements checked in on a specific branch (for example, <App System>_REL1) into the corresponding elements in the workarea that captures the MAIN branch (for example, <App System Cluster>_DEV).

You can launch the merge wizard from the RON.

Naming Standards quick reference information

Release nr [x]

For example, 10.

The format of a release nr is indicated with x where x represents the major component. A release can contain all objects of an application system (baseline release) or it can contain only the changed and added objects introduced in a specific release (increment). Note that releases are implemented by configurations and that you could have multiple versions of configurations (see below).

Version nr [1.y]

For example, 1.1

The format of the version number of a single object is indicated with 1.y where y represents any major or minor change.

Branch Version nr [1.y.1.z]

For example, 1.1.1.1

The format of a branch version number of a single object is indicated with 1.y.1.z where ‘y’ and ‘z’ represent any major or minor change.

<App System Cluster>_DEV workarea

Workarea that captures the latest versions of all objects of one or more strongly interrelated application systems, for example MARKETING_DEV. The default checkin branch is equal to MAIN.

<App System Cluster>_SB<sequence nr>

Workarea to prepare a specific release or to try out different workarea rules with a stack of configurations. SB stands for sand box (play ground). There is no default checkin branch and there are no database schemas associated with this workarea.

<App System Cluster>_ACC

Workarea that captures one or more application systems of a specific release and/or application system patches that has reached the acceptance test status. There is no default checkin branch (by definition).

<App System Cluster>_PRD

Workarea that captures one or more application systems of a specific release (and/or application system patches) that has reached the production status. There is no default checkin branch (by definition).

<App System Cluster>_PRDFIX

Workarea that fixes incidents on one or more <App System> release that has reached the production status. The default checkin branch is equal to <App System>_REL_<release nr>.

MAIN Branch

Branch label for the MAIN branch

<App System>_REL<release nr> Branch

Branch label that captures the latest version of a specific <App System > release (for example <App System Cluster>_REL10).

<App System>_PRDFIX_REL<release nr> Branch

Branch label that captures the latest version of all production fixes of a specific <App System > release (for example <App System >_PRDFIX_REL10).

<App System>_REL<release nr> Configuration

Base line configuration for <App System> that contains a full set of all database elements or all changed and introduced elements (increment) (for example <App System>_REL10).

Note that you have to create a new baseline release if you remove elements from the <App System>.

<App System>_REL<release nr>FIX<sequence nr> Configuration

Patch configuration for <App System> that contains all elements that are solved for one or more fixes in

the context of a specific release (for example <App System >_REL10FIX1).

<App System>REL<release nr>.<ext> DDL script

Baseline DDL scripts belonging to a specific release of an application (for example <App System>REL10.sql, <App System>REL10.tab).

For a list of possible values of the <ext> see below.

<App System>REL<release nr>.sql DDL script

Overall DDL script, derived from the generated baseline scripts for a specific application system. The <release> part stands for the target release.

<App System><short name table>.ins Seed data script

Seed data script for a specific table within a specific application system.

<App System>REL<release nr>FIX<sequence nr>.sql DDL script

Overall DDL script, derived from the generated delta script(s), for a specific fix (or fixes) in the context of a specific release. <App System>REL10FIX1.sql

<App System>_OWNER Database Schema

Specific database schema that captures the database objects of a specific application system.

ins Folder

Sub-Folder that captures the database install (DDL) scripts.

ddl Folder

Sub-Folder that captures the packages stored as files (for example *.pck) and the overall release delta script that is transferred to Application Services.

<SID Name> Database Instance

System Identifier (SID) of the Oracle database.

Checkin and Checkout Quick Reference Information

Guidelines for Checkout and Checkin notes

1. Always fill in the checkout and checkin text box in English.
2. Provide references, when applicable, to the corresponding “Change Request” numbers.
3. Reuse the checkout text while checking in and subsequently update the checkin text with actual revision information.
4. **Never** perform an undo checkout against containers (application systems or folders) because you are not in control of the checkout context, other developers may have added or removed elements.
5. The context of a checkin should be equal to the change request context. That is, do not leave the element in a checked out status after you have finished your “Change Request” changes for a specific element.
6. Always add the revision keyword string to the structured design element of file (see Revision keyword guidelines).
7. Check in an element only if the element is complete and if the generated DDL syntax can be compiled or created without any errors. A table is complete if it contains all secondary elements (for example, columns, constraints, triggers, synonyms).

Checkout checklist for existing structured elements and files

1. Consult the members of the partial configuration - representing the “Change Request”.
2. Verify the checked-in status of the element - for structured elements and files only. Consult section Request Check-list for a checked-in status of a checked-out element if the element is already checked-out.
3. Verify the appropriate version level on the branch - via the VHV. In most cases you would like to check-out the tip version.
4. Request for a refresh or recompile of the context workarea - ask the PCM) if you do not view the tip (or latest) version on the branch..
5. Verify the correctness of the folder mapping against the root-container (files only). Note that folder mapping of a root container is unique within the context of a workarea. Your folder mapping for the root container should be mapped to a private workspace on the file system (for

example g:\ work\bck)

6. Check-out the specific element - either in the RON or DE - and provide check-out notes, see also section Guidelines for Check-out and Check-in notes - above. Note that you can only use the RON to check-out files
7. Check-out associated elements that also need to be modified, for example the entity at the other end of the relation.

Checkin checklist for folders, structured elements or files

1. Check out the owning container, if the owning container is not already checked out, for new elements only.
2. Add the revision keyword string, for new elements only (see Revision keyword guidelines).
3. Check the access rights, for new folders only. Note that only a PCM is allowed to create folders.
4. Create your new file with content (files only) on the file system before a checkin.
5. Check the completeness of the elements (for example, attributes, unique identifiers, columns, constraints).
6. Check in the element and provide checkin notes (see "Guidelines for checkout and checkin notes", above). Subsequently consult that information if you need to merge the structured element or file, for non-containers elements only . Or consult "Merge guidelines for containers" if you need to merge the container, for containers only.
   Note that you can only use the RON to check in files and/or containers.
7. Synchronize the corresponding database schema via the DDL generator, for tables, views and secondary database elements only (that is, triggers, indexes) (see also Database synchronization QRC).
8. Update and check in the partial configuration by adding specific versions of the new structured elements/files or changed versions of already populated structured elements. Note that you should always include the owning container of the element.
9. Analyze dependencies with the Dependency Manager. Note that the Dependency Manager is only applicable for design objects like tables, views and seed data scripts.

Request checklist for a checked in status of a checked out element

1. Determine the owner of the checked out element (for structured elements and files only) via the VEV.
2. Do ONE of the following:

- Postpone your changes until a checked in status is reached by the owner of the checkout.
- Request a checked in status, on the default checkin branch, of the element in the near future. This is likely to happen if the owner is nearly finished or can postpone his or her changes. Note that you

may deal (and therefore agree) with an undefined status of the element if the owner was not finished yet.

- Request for a checked in status on another branch if the owner agrees that his or her changes must be merged eventually on the default checked in branch.
- Ask permission of the PCM to undo the checkout of the element if the owner is not available. Note that with this operation all ongoing changes are lost.
- Enforce a checkin on the default checkin branch thereby accepting an undefined status of the element, if the owner is (still) not available.
- Enforce a checkin on another branch (after consulting the PCM) if the owner is (still) not available.

Note that the above operations cannot be enforced by Oracle9i Designer. Every developer is able to execute each of the listed actions, since every developer has the version right and they all share the same workarea.

Revision keyword guidelines

The repository supports the following revision keywords:

- $Re'||'vision:::xxxxxxxxxxx$ (aka revision)
- $Au'||'thor::xxxxxxxxxxxxxxxxx$
- $Da'||'te::xxxxxxxxxxxxxxxxxx$
- $Wo'||'rkfile::xxxxxxxxxxxxxxxxxxxxxx$

Note that you should not use the string '||' on the specific location. This string is added here to circumvent the expansion when this specific QRC document is checked in to the repository.

You should always add the "revision" keyword string, on a specific location/property, into new design elements and files. Use the following overview for the specific location:

- Table - comment property
- View - comment property
- File - in the header, prefixed with a comment tokens when used in a SQL script or Shell script

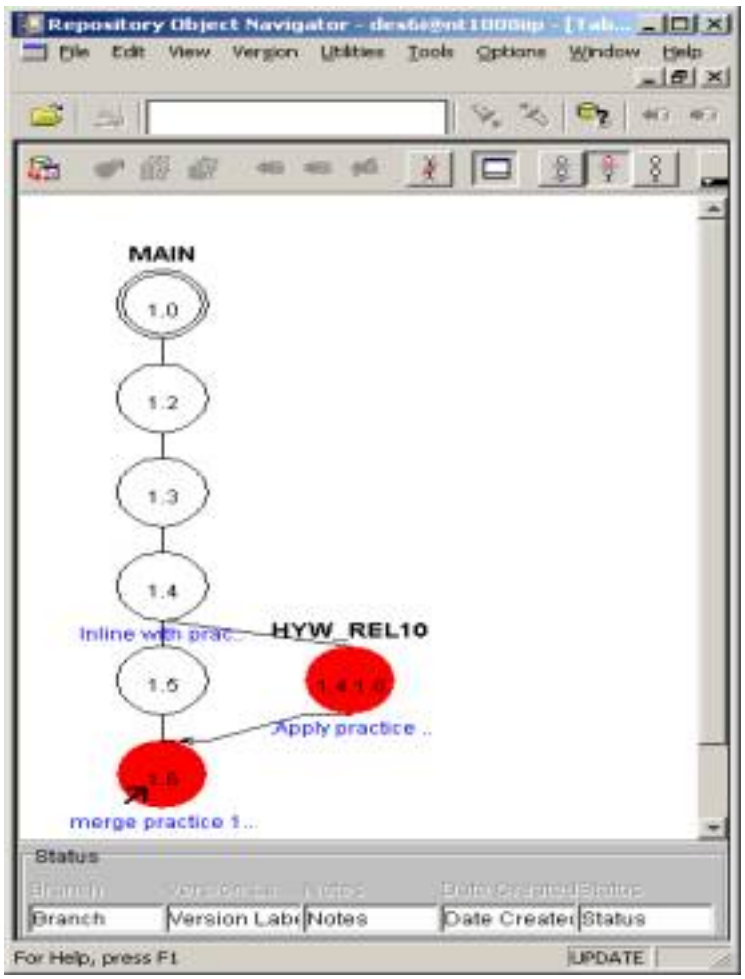
Note that you can copy/paste an existing revision keyword string for the exact syntax from existing tables/views/files. The version string is automatically updated after the checkin operation.

Removal checklist for folders, structured elements or files

1. Verify the checkin status of the structured element, file or folders. Only checked in elements may be removed.
2. Verify the dependencies, usages and inclusions (via the RON and the dependency analyzer) of the element.

3. Determine the appropriate delete order based on your dependency investigation.
4. Check out the owning the container with checkout notes, if not already checked out.
5. Remove the element(s).

Merge guidelines for structured elements and files



1. Perform a merge to the MAIN branch each time you are planning to cut a new release from MAIN.
2. Find the candidate merge elements, for a specific user, on a specific branch via the ROB search utility by providing a value for the following fields - on the several search ROB TABS for example:

   Workarea: [Basic]

   Branch Label : [Version]

   Checked In By: [Version]

   Check State: “Checked in” [Version].

   Note that you may find too many merge candidates, since you do not need to merge if the tip of the branch is already merged into the MAIN branch. The latter can be visualized via the ROB Version History Viewer. You could also try to narrow down the search result by translating the “merge of the tip version into the MAIN branch” condition into an “additional where clause” field on the

[Advanced] search TAB.

3. Highlight the TARGET workarea where you will execute the merge.
4. Highlight the candidate merge object, check it out and launch the VHV.
5. Abandon the merge whenever the source or the target tip version is already checked out, indicated by the blue color, and start a request for a checked in status (see "Request checklist for a checked in status of a checked out element") to enforce a checkin status.
6. Mark within the VHV the checked out version, move your cursor to the source object version and launch the option “merge to mark”. The Merge dialogue or wizard tries to suggest a default change on a specific branch for every conflict. This default is indicated with an enabling of a specific version. Subsequently you can perform an automatic merge if every conflict (or difference) is associated with a default. Therefore you may have to manually interfere whenever the wizard cannot make a specific choice for a particular difference.
7. Save the merge operation. The merge itself does not automatically create a new version.
8. Verify the merge result with a compare of the source object version with the target (merged) object version and make additional changes if necessary.
9. Check in the target (merged) object version.
10. Consider a merge to other branches of the object.

Merge guidelines for containers

The merge steps for containers are more or less equal to the merge guidelines of structured elements and files.

However, you will only be confronted with differences of folder memberships (see step 6 of the "Merge guidelines for structured elements and files").
