



Ministry of Education  
Information & Technology Management Branch

## **Software Configuration Management Standards and Services**

Repository Management & Configuration Management Unit  
Last Updated: April 28, 2003  
Version: 2.0

# General Background on SCM

## What is Configuration Management?

Configuration Management is the discipline of organizing and managing products and their individually existing component parts. The right components need to be assembled in the right way to produce the appropriate end product.

Components can be altered slightly or recombined to create a slightly different version of the end product (or a different product altogether). One of the goals of Configuration Management (CM) is to manage the processes by which new component versions are created and their variations documented. It also addresses the recording and creation of new end products, particularly which component versions are involved in assembling the product.

## What is Software Configuration Management?

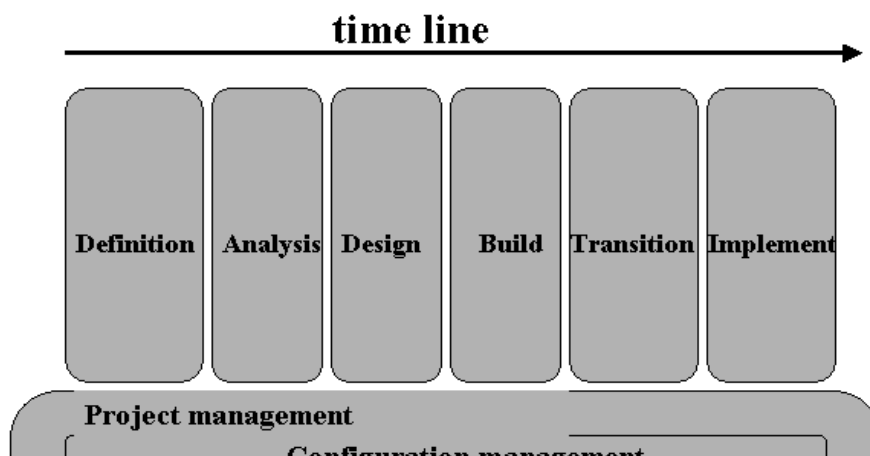
Configuration management applies to software engineering since software products are usually made of a selection of components.

An individual program, or a table in the database, are usually components of a specific application. An application requires a specific combination of versions of these components in order to function as a whole. In addition, a suite of applications or the applications that combined make up the enterprise environment also need to be 'version aware' of each other.

While historically the information systems industry has done reasonably well in managing elements at the file level and database level, it is only recently that the IT professionals have been faced with the need to manage metadata elements. With the use of CASE (Computer Aided Systems Engineering) tools, a new level of complexity was added to the CM discipline. The analysis and design definitions which originate an application's physical components also need to be configuration managed.

## CM positioning

According to Oracle's Custom Development Method, Configuration management is a discipline within project management. As with project management, it starts before, continues during, and ends after, the software development life cycle.



The scope of configuration management can go beyond individual projects. Much like project management needs to be aware of inter-project dependencies, CM also manages inter-project dependencies

configuration perspective.

from a

[< Next >](#) [< SCM Home >](#)

# Classification of Configuration Management processes

Within the discipline of configuration management, processes can be categorized by their requirements and objectives, as follows:

- **Developer CM** - developer CM supports software developers and analysts in making changes to product components. Aspects of such support include management of developer workspaces, propagation of software changes, version control and impact analysis.
- **Project CM** - project CM is related to the development and maintenance of a single product. Concepts such as change control, component packaging, promotion management and life-cycle process management are relevant to project CM.
- **Corporate CM** - corporate CM looks at software development from the global business perspective. It provides the context within which configuration management of a product spectrum may occur. Corporate CM is also concerned with the management of the environments for defining standards such as enterprise data models and code libraries, and their publication and use.

Other sections of this site are organized based on this classification.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

## Terminology related to SCM and the Ministry SCM toolset

Within the realm of SCM and the use of Oracle SCM as a configuration management tool, the following terminology is used:

- **CM repository** - The database schema that holds all managed elements of the enterprise. Within this repository, elements are tracked in terms of their ownership, version, state, type and usage.
- **Element** - At the lowest level of granularity, the repository is filled with elements (also referred to as objects; the two terms are interchangeable). All elements are classified as either Structured or Unstructured. In Oracle SCM and Designer, elements are also referred to as Primary Access Elements (PACs).
  - **structured element** - 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 Elements at this moment are the Oracle Designer element types (Entity, Business Function, Table Definition etc.)
  - **unstructured element** - An element that has a structure that is unknown to the Repository infrastructure is, somewhat strongly, labeled Unstructured Element. 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 the lowest level of granularity for versioning is the element in its entirety. For example, an entire entity definition is versionable, but you cannot version its attributes or unique keys and relationships independently.
- **Container** - an element grouping area within the repository. Containers can be of the following types:
  - **Folder** - a container that holds unstructured elements
  - **Application system** - a container that holds structured and possibly unstructured elements
  - **Package** - a general purpose mechanism for organizing elements into groups. It provides the context and namespace for the elements it owns. (used primarily in the context of java and Jdeveloper)
  - **Project** - a container that references elements owned by other packages. Projects do not own elements. (used primarily in the context of java and Jdeveloper)
    - Note: Containers have the version dimension built-in. In other words, containers hold all versions of every element they own. All versions of repository elements are always owned by the same container.
- **Work area** - a work area is an environment that provides a 'version-resolved' view of specific repository elements for a specific group of users / user roles. Access rights are established for work areas. Work areas are usually based on containers and / or configurations, which may have further access rights defined. Thus, the contents of a work area may vary from user to user, due to their varying access rights to the underlying components.
- **Configuration** - a fixed set of element versions. Configurations provide the basis for Release Management. They specify the element versions that together make up a product base release or patch or any otherwise significant grouping of elements.
  - Base release - a configuration that identifies the components of a major product release. Major product releases are the '.0' releases. For example, release 1.0, 2.0, etc.
  - Patch release - a configuration that identifies the components of a product patch or

enhancement release. These are the '.x' releases. For example, release 1.1, 1.2, 2.2, etc. Configurations are versionable elements as well.

- **Promotion model** - a specific set of environments and technology levels that elements go through during their development or maintenance process, as well as the allowed environment transitions.
  - Promotion level - (a.k.a. promotion state) the development levels that an element goes through. These levels generally follow the enterprise adopted systems development life cycle, with some additional levels. An example of a set of promotion levels is:
    - Development
    - Integration test
    - Acceptance test
    - Production
    - Post-implementation emergency fix
    - Training
- **Promotion environment** - a physical environment where elements reside during their existence in a promotion level. Each technology layer requires a promotion environment for each promotion level.
- **Promotion** - an allowed state transition between promotion levels
- **Demotion** - an allowed state transition 'rollback' between promotion levels
- **Version** - A frozen definition of an element.
- **Version tree** - the structure that holds the information about the evolution of a set of versions for a given element. A version tree holds the linkages between versions, such as parent-child relationships, root relationship, source relationship etc. Version trees must have one or more branches (see below).
- **Branch** - a linear linkage of element versions. A version tree has at least one branch (a.k.a. the MAIN branch). Branches will always originate from a particular element version, which can be in the MAIN branch or any other branch of the element's version tree.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

## Background for SCM at the ITMB

Software Configuration Management is one of the service areas within the Information Management Group (IMG) of the ITMB. The SCM service is one of the aspects of information resource management and application management.

Some of the goals of the Repository Management and Configuration Management Unit within the IMG are:

- To provide a reliable environment for definition and maintenance of the enterprise's information and application resources
- To provide adequate access to the configuration management repository
- To facilitate the enforcement of information management and application development standards across the enterprise

The IMG carries out configuration management using a toolset (see [SCM tools](#)). This toolset has been in operation since the summer of 2000.

The IMG supports SCM at the three process area levels identified in [Classification of Configuration Management processes](#). The implementation of SCM services, however, is following an evolutionary approach. To date, the IMG has implemented version control, change and delivery management, promotion management and release management. Future objectives include the implementation of build management, process automation and extraction of SCM metrics.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

## SCM Tools

The ITMB currently uses the following SCM tools:

- **ECHO:** this tool provides additional SCM capability to the Designer 6.0 environment. Use of ECHO is restricted to the Repository Management Unit. Element compares and copies are done by that unit on request.
  - **Note:** no further documentation is provided on this site about ECHO due to the near obsolescence of this toolset.
- **Oracle SCM:** this tool provides SCM capability to the Designer 6i, Developer 6i and JDeveloper environments and also is the SCM repository for management of files (source code, project deliverables, documentation, etc). Generally available from this tool are functions such as element compare, copy, merge. Impact analysis is also available for Designer design level elements and source code.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Contact information

For additional information on SCM at the ITMB please contact:

Bia Stratton Repository Management / Configuration Management Unit Information Management Group email: Beatriz.Stratton@gems1.gov.bc.ca phone: (250) 387-1163
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Promotion Models for application development

A **Promotion model** is a specific set of states, state transitions and technology environments that elements go through during their development or maintenance process.

A **promotion state** is a development stage that an element goes through.

A **promotion environment** is a physical environment where elements reside during their existence in a promotion state. Each technology layer within the ITMB's adopted technology stack requires a promotion environment for each promotion state.

The ITMB promotion model has a "work-in-progress" (WIP) environment, where application components are being designed and built. Elements in this environment are expected to be in unstable state. Once elements have reached a stable state they can be checked-in. When exactly a check-in happens is to some extent at the discretion of the developer. Usually a few intermediate versions will be created as a 'savepoint', or 'snapshot', of the element at a particular point in time, or to save interesting definitions prior to additional work.

Prior to delivery, elements are checked-in as well, and bundled in a configuration.

The "Development delivery" area is used by the repository management unit when creating configurations while development remains active in the WIP environment.

The ITMB promotion model has at least two stages of testing and quality assurance that elements go through before reaching the production environment. When defects are found, elements are checked-out and worked on in the work-in-progress environment, and their new (fixed) versions are included in the promotion configuration, replacing the previous (faulty) versions.

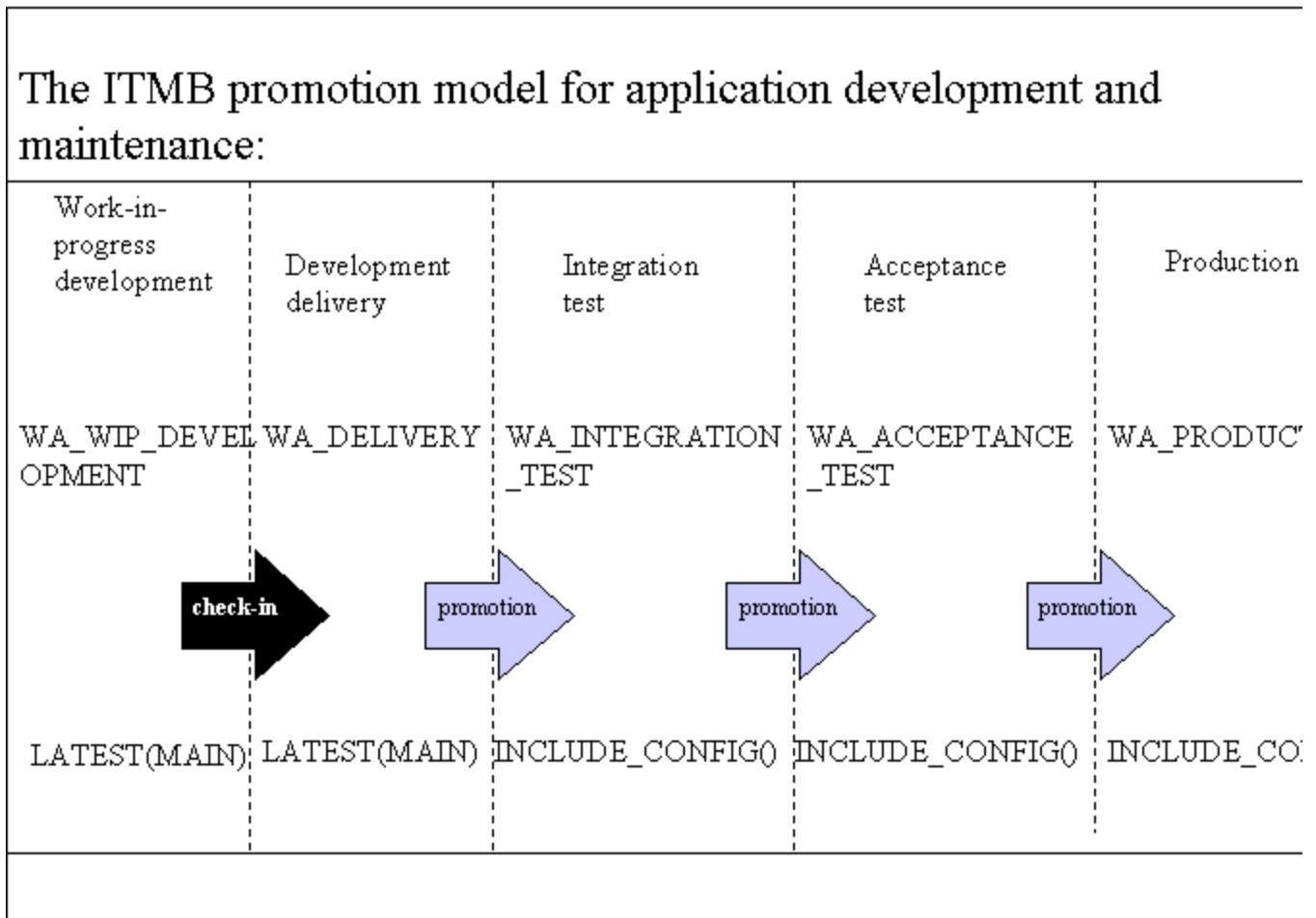
The integration test environment is geared towards developer testing of changed components in their application context. Unit testing is performed while in the WIP environment.

The acceptance test environment is used by users when testing application components.

IMG quality assurance is performed prior to user acceptance testing. QA may be carried out in the WIP workarea or in a separate QA workarea (not shown in the figure below), depending on the characteristics of each project. All QA staff have read access to the WIP workareas.

During the approval stages, configurations may be updated with newer versions of elements, or new elements altogether. Once a configuration reaches the production environment it is no longer updated. If defects are found after elements reach production, a new patch release configuration is created following the same promotion process.

Figure 2 depicts the ITMB promotion model.



Each promotion level in the promotion model is represented by a workarea in Designer 6i. Each ministry supported by the ITMB has its own set of promotion model workareas. (See section "Ministry Specific Environments" in the table of contents, for more information.)

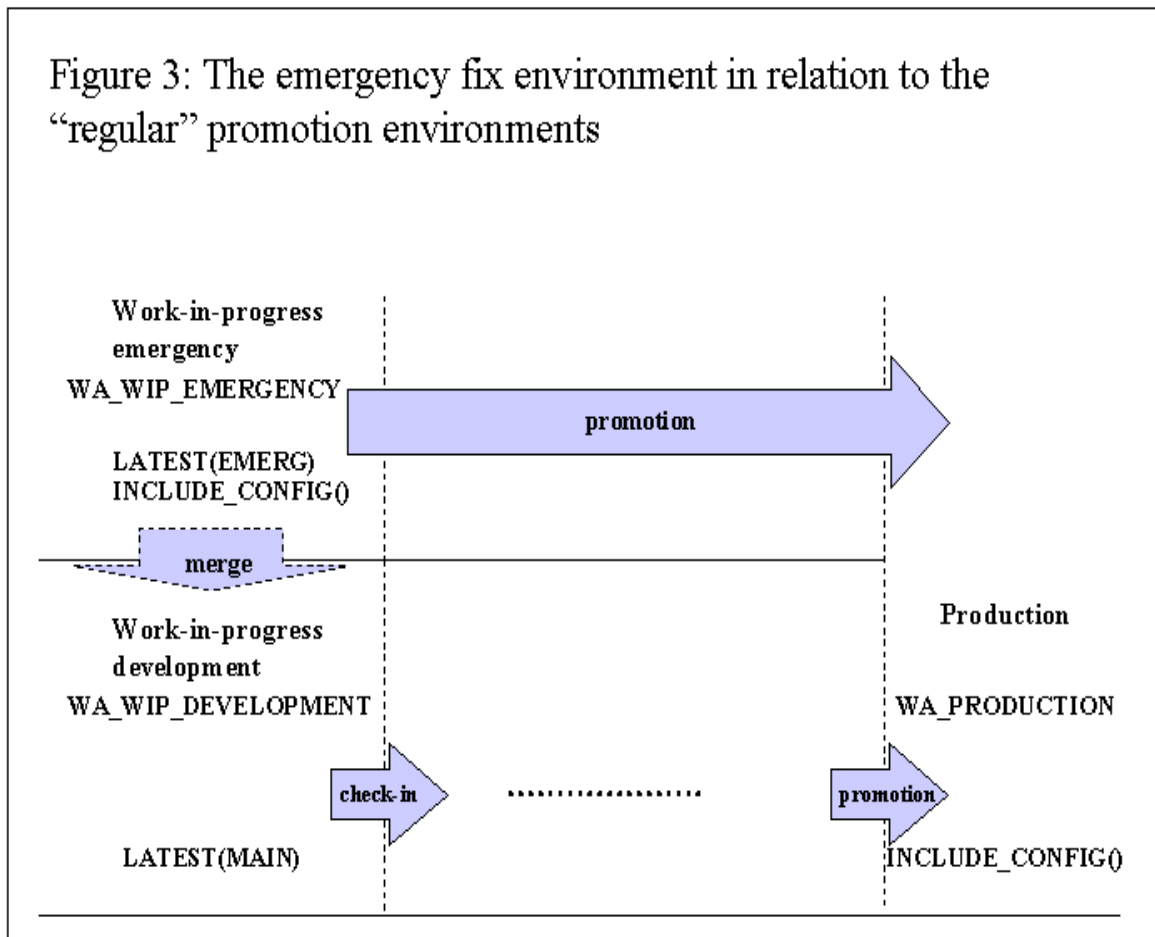
### Emergency fix environment

Emergency fixes to production applications may be required occasionally. When emergency fixes are required in the midst of the next release development project, the application components requiring the fix may be checked-out or may have subsequent versions other than the one that is in production.

In order to carry out an emergency fix, therefore, one must have a separate environment where changes can be made without affecting or being affected by ongoing development.

Figure 3 shows the interaction between the emergency fix environment and the promotion model.

Figure 3: The emergency fix environment in relation to the "regular" promotion environments



Elements requiring an emergency fix are worked on in the "work-in-progress" emergency workarea. Once the fix has been tested and approved, all fixed elements are bundled in a configuration. This configuration is then applied to the production environment as a patch, overwriting the previous element versions.

Once the patched elements have been promoted to the production environment, their changes may need to be merged back into the MAIN branch element versions. Whether or not a change is merged will depend on its usefulness to the new release under development. In some cases, a component may undergo major rework for a new release, and a change made to fix a production bug may no longer be applicable in the newly designed element version.

### Other variations of the standard promotion model

Some variation to the above presented promotion model may be applicable to projects at the ITMB. This variation may be warranted due to specific project characteristics such as the use of prototyping techniques or iterative development. The following sections detail the allowed tailoring in these situations:

- **Iterative development or prototyping** - In a software development project that uses prototyping techniques or iterative development the promotion model will include an additional promotion level. This level represents a 'prototype evaluation' exercise. Typically, elements will be checked

in and bundled in a configuration, which is made available in a 'prototype evaluation' work area, as many times as the number of iterations defined. After the iterations have been completed, elements follow the normal promotion stages as previously defined.

# Version control

Version control is the process of maintaining multiple versions of a given object (be it a document, program or element definition such as an entity definition or a class definition). It is the most basic process within SCM.

The most important consideration that must be made when versioning objects is the "granularity", or the level at which versioning is done. The remainder of this section details the level of granularity for versioning of all objects under version control at the ITMB. For information on the "mechanics" of versioning, please refer to the section [Developer SCM tasks](#).

## **Designer 6i objects:**

The level at which versioning is done, for all Designer elements, is the "Primary Access Control" level (a.k.a.PAC). PAC elements are owned by an application system, and can be of different types. Examples of PAC types are entities, functions, tables, modules, objectives, business units, etc.

As we know, most Designer PAC elements have sub-elements and associations to other elements. For example, entities have attributes, unique identifiers and relationships to other entities. Given that versioning is at the entity level, there is no such function as "versioning an attribute". The version of the attribute is the same as the entity version.

For PACs that are primarily grouping elements, such as **User Defined Sets** and **Diagrams**, the versioning of the PAC element represents a frozen view of the element's constituency, but does not go as far as identifying the specific version of the constituent objects.

## **UML objects:**

TBD

## **Application source files:**

All application source files are version controlled in Oracle SCM. These files are managed within a standard folder structure that exists within the application system container. The top level of the folder structure is called "CM\_non\_generated". Within this structure, files should be placed in the sub-folder appropriate for their file type prior to version control (e.g. SQL, Forms, Reports).

The level of granularity for any type of file is the file itself. This is appropriate most of the time. Each Oracle Forms source file is versioned by itself. Any referenced libraries are themselves source files and are thus managed separately.

For DDL scripts and pl/sql source, depending on the file content, the granularity of "file itself" may be inappropriate. For example if a file contains the package definition for all packages in an application, they are all versioned together because they are in the same file. The current ITMB guideline (and generally accepted best practice in the development community) is to have one file for each package specification and another set of files for each package body..

## **Project deliverables:**

Project deliverables are also version managed in Oracle SCM. Each deliverable file is managed individually (i.e. we do not manage a zip file containing a set of deliverables).

Project deliverable files are managed within a standard folder structure that exists within the application system / project container. The top level of the folder structure is called "CM\_project\_deliv". Within this structure, there is one folder for each CDM process, and deliverables for a given process should be placed in the corresponding folder prior to versioning.

The granularity for versioning is each file making up the specific deliverable. This generally translates to a Word document and possibly other documents such as Visio diagrams or pdf files.

**Other objects or files:**

Please contact the repository manager if your project is producing application components, deliverables or other files that you don't find a placement for.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Release Management

Application software is managed by the assembly and deployment of release configurations. No application component will be promoted beyond their work-in-progress environment without being in a configuration.

Configurations related to application and their components are:

- Component configuration
- Base release configuration (this can be made up of a set of component configurations)
- Patch or enhancement release configuration
- Emergency fix configuration

Configurations are also created for project life cycle deliverable management and delivery.

The mechanisms and procedure for creating release configurations are explained in the section [Project CM tasks](#).

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Impact analysis

## What is it?

Impact analysis can be defined as: keeping track of all dependencies, and their nature, of and between the elements in a system. Impact analysis is the process of storing and retrieving information about all links between elements.

We can make a distinction between *dependencies* for an object, meaning all objects needed by the object under scrutiny, and *impact* of an object, meaning all objects possibly affected by the object under investigation, or all objects using, in some way, that object.

## When to use impact analysis

Impact analysis can be very useful, important, or even necessary, in the following situations:

- to track down a bug
- to estimate the impact of a proposed change
- to define and allocate work (on an element grouping)
- to define sub-systems, plan development and tests
- to verify completeness of a deliverable
- to compile, generate or build executables
- to define access privileges to objects
- miscellaneous
- 

What is the impact analysis capability of the ITMB development environment and toolset?

### **Designer 6i:**

The Designer repository is composed of information about elements and their dependencies. Dependencies are an intrinsic part of the repository. It seems, at first glance, that for metadata (element definitions), the dependency information is completely covered by Designer.

On closer inspection, however, we can see that some dependency information is missing, and that we do require additional information, which at this point is provided by Oracle SCM.

An example of a dependency that cannot be found in the Designer repository are:

- a form that makes use of an Oracle sequence when inserting records in a table. This information is not directly available. By inspecting the module definition, one can determine that the module will insert records in a table, but not that it will issue a sequence number from a sequence. This dependency is, however, found by Oracle SCM, on inspection of the form source code generated by Designer

Beyond using the Repository Object Navigator to inspect dependencies, you can also use the Matrix Diagrammer. Some of the Repository Reports also provide dependency information.

### **Oracle SCM:**

Oracle SCM has the ability to determine dependencies between and across elements of the following types:

- the Designer repository metadata, for objects at the design and implementation levels
- Oracle Forms
- Oracle Reports
- SQL and PL/SQL
- C and ProC
- Java

Before Oracle SCM can establish dependencies, the source files for programs of the above types must be stored in the repository. You can find information about file upload and download in the section [File upload and download from the repository](#), available from the main index page.

Oracle SCM stores dependency information in a set of "dependency tables". It uses its knowledge of the structure of the Designer repository, plus parsers for source code files, to determine dependencies among element definitions and programs.

The calculation of dependencies is an automated process under the responsibility of the Repository Management Unit of the IMG.

"Unavailable" or "out-of-date" dependencies can be identified by blurred element names in the dependency manager object navigator pane.

Dependencies that cannot be found by the existing ITMB toolset:

At this time, we are unable to automate dependency tracking between files created by Microsoft Office tools such as Word, Excel, etc. Therefore, if a Word document makes references to table definitions stored in Designer, the Oracle SCM dependency analyzer cannot identify them.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Developer CM tasks

Note: "developer" in this context refers to an individual performing development or maintenance of any type of managed element, such as a model and its components, standards documents, templates, application system models, database objects and documentation.

As a developer, your primary concern is with version management of elements you're working on / responsible for. Occasionally, you will likely want to create a "freeze" of a particular element, before making additional modifications to it, or in preparation for delivery.

**To freeze a version of an element**, use the **Check-in tool**. This tool is available via right-click on an element or set of elements, and also via the Version menu.

**To "open" an element for change**, use the **Check-out tool**. This tool is available via right-click on an element or set of elements, and also via the Version menu.

**To find out all elements that you have checked-out** and/or elements that were never versioned, select the container you're working on, and use the **List Checkouts utility**. This utility will provide you with a list of the above elements, and give you the option of checking them in. You can find this utility via right-click on the container, and also via the Version menu.

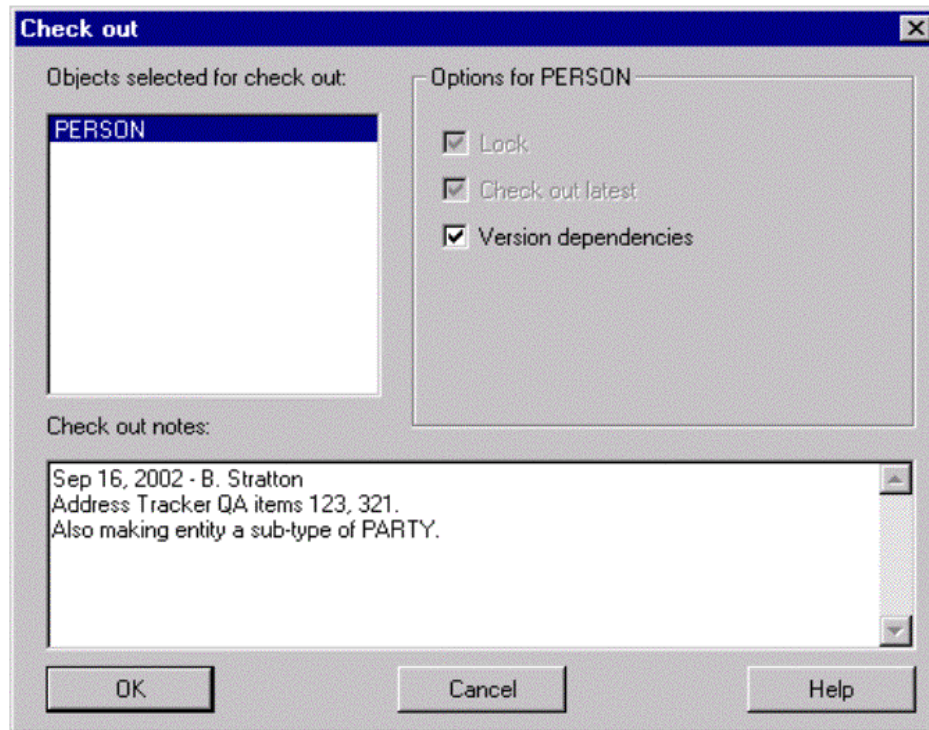
Before you can perform version control on a file, the file needs to be uploaded to the repository first. For information on both upload and download, please refer to the section [File upload and download from the repository](#).

## Check out standards

When checking out an element, you should always fill out the "Check out notes" text box with, at least:

- date of check-out
- your name
- a reference to a QA item that you are addressing, if any
- additional changes you're making to the object(s)

You can leave the "version dependencies" checkbox as is.



### Check-in standards

When checking in an element, you should always fill out the "Check in notes" text box with, at least:

- date of check-in
- your name
- a reference to a QA item that you have addressed, if any
- additional changes your've made to the object(s)
- **Note:** you can check the "Use check out notes" box to pre-populate the Notes with what you have specified at time of check-out (if this is a checked-out object, not the initial check-in)

If you're checking in the object to create a savepoint, but you want to continue working on it, you can check the "Check out after check in" box.

You should **never change** the default branch provided in the "Use Branch" property.

The screenshot shows a 'Check in' dialog box with the following components:

- Title Bar:** 'Check in' with a close button (X).
- Objects selected for check in:** A list box containing 'PERSON'.
- Options for PERSON:**
  - Current check in branch: MAIN
  - Use branch: A dropdown menu showing 'MAIN'.
  - Version label: A text box containing 'Automatic generation'.
  - Three checkboxes:
    - Use check out notes
    - Check out after check in
    - Check in even if unchanged
- Check in notes:** A text area containing:

Sep 16, 2002 - B. Stratton  
Address Tracker QA items 123, 321.  
Also making entity a sub-type of PARTY.  
Also added several attributes after approval of PARTY and sub-types enhancement request.
- Buttons:** 'OK', 'Cancel', and 'Help'.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Project CM tasks

Project level CM tasks are concerned with the packaging of sets of elements for the purposes of:

- identifying the constituency of relevant sets of elements, such as of an application component or business area within an enterprise model
- identifying an application release
- creating groups of elements in preparation for delivery

Once elements have been packaged in a release, the release needs to be tested for completeness.

Creating a release and its constituency list:

Grouping of release components is done using the User Defined Set (UDS) element type in Designer, which allows you to group any number of elements of any type (including files). User Defined Sets created by the project or application configuration manager are used by the repository management unit to populate configurations. Configurations further define the element set by specifying the particular element version that should be in the set, while the UDS only specifies the set's membership list.

**To create a User Defined Set:**

- if you can see the User Defined Set node in the Repository Object Navigator, select it and use the Create Object button to create a UDS occurrence.
- alternatively, you can also select the application system you're working on, and use the Create as Child button, selecting User Defined Set from the pick list of element types.

Once you have created the UDS element, you have to create its membership list. When you expand your UDS node, you will see the Set Members node. You can use the Create Reference button to add all required elements into the set.

Prior to delivery to the IMG, the UDS, and all elements it points to, must be checked-in.

A detailed description of the delivery process can be found in the [IMG Catalogue](#) site, in the document "Release Management and Delivery".

Testing the release:

The promotion workareas in Designer 6i play an important role in allowing release completeness/correctness testing.

To ensure that a release is complete ( it has all the objects it needs) and correct ( it has the right versions of each object), you must test the release by deploying it in a test environment that mirrors the production environment. Testing will include applying the release "on top" of the current production configuration. The release components can then be tested for compatibility with the production components.

Therefore, when you are ready to perform integration test, you must generate or extract all objects from the integration test workarea, which is where your release will be promoted to, once it is delivered.

Ministry application support staff who are deploying the release in the user acceptance test or production environments must also generate or extract all objects from the acceptance test and production workareas respectively.

- **Note:** the "one bucket" concept, where all source is placed and extracted from the same location (the development workarea) does not achieve release testing. It also creates problems when additional development is taking place in the development workarea, and subsequent object versions are created for members of the release.

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Ministry of Education environments

The environments specific to the Ministry of Education are a set of workareas that represent the promotion levels used for application development.

The workarea names are:

- WA\_EDUC\_DEVELOPMENT
- WA\_EDUC\_ACCEPTANCE\_TEST
- WA\_EDUC\_QA
- WA\_EDUC\_TRAINING
- WA\_EDUC\_PRODUCTION
- WA\_EDUC\_EMERGENCY
- 

Access to each workarea will vary according to each user's role and the promotion level in question.

For a description of the promotion model and it's intent, please refer to [Promotion Models for application development](#).

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Ministry of Advanced Education environments

The environments specific to the Ministry of Advanced Education are a set of workareas that represent the promotion levels used for application development.

The workarea names are:

- WA\_AVED\_DEVELOPMENT
- WA\_AVED\_ACCEPTANCE\_TEST
- WA\_AVED\_QA
- WA\_AVED\_TRAINING
- WA\_AVED\_PRODUCTION
- WA\_AVED\_EMERGENCY
- 

Access to each workarea will vary according to each user's role and the promotion level in question.

For a description of the promotion model and it's intent, please refer to [Promotion Models for application development](#).

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Ministry of Labour environments

The environments specific to the Ministry of Labour are a set of workareas that represent the promotion levels used for application development.

The workarea names are:

- WA\_LABR\_DEVELOPMENT
- WA\_LABR\_ACCEPTANCE\_TEST
- WA\_LABR\_QA
- WA\_LABRTRAINING
- WA\_LABR\_PRODUCTION
- WA\_LABR\_EMERGENCY
- 

Access to each workarea will vary according to each user's role and the promotion level in question.

For a description of the promotion model and it's intent, please refer to [Promotion Models for application development](#).

[< Previous >](#) [< Next >](#) [< SCM Home >](#)

# Enterprise Models and their use in application development

The IMG has developed enterprise models for each Ministry supported by the ITMB (namely, Ministries of Education, Advanced Education, and Labour) . These enterprise models provide overall context and "seed" components for application development.

## Systems development methodology modifications related to the use of enterprise models

The IMG's adopted systems development methodology (Oracle CDM) provides a development framework for systems development which has been extended to allow for the additional tasks and deliverables necessary for enterprise model integration. A detailed description of these tasks can be found in the document "IMG Deliverables by SDLC Phases for Classic CDM projects" available for download from the [IMG Catalogue](#).

## Enterprise model publication

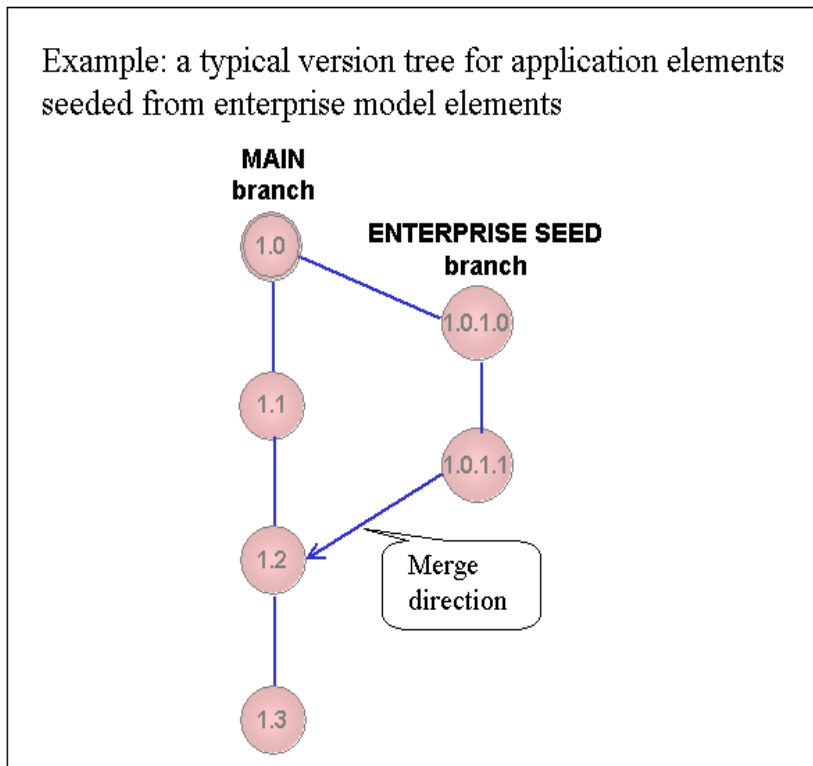
Ministry enterprise models are published to the ministry application development workareas, and are accessible by all development teams in read-only mode. New releases of the enterprise model are published to this environment when deemed ready for use in application development. Only the latest version is visible in the application development workareas.

A notice of publication is sent to all repository users when new model versions are published.

## Seeding of application development containers

In the beginning of an application development project, a container is created for the application. This container is populated with the portions of the enterprise model that are applicable to the application's business area (as determined by the IMG Data Administration unit). This "seeding" is performed by the repository management unit before delivery of the container to the development team.

The seeding process affects the version tree for those elements in the application system container which have originated from an enterprise definition. The figure below provides an example of a version tree for such an element.



The ENTERPRISE SEED branch is used to store the seed versions used by the application container. The contents of this branch are managed by the Repository Management unit. New seed versions are added to this branch as directed by the Data Administrators.

The application development team has "read-only" access to the ENTERPRISE SEED branch, and full access to the MAIN branch, which is where element versions created during application development are stored.

The first seeding creates versions 1.0.1.0 on the ENTERPRISE SEED branch and 1.0 on the MAIN branch. These are identical. Version 1.0.1.0 provides the frozen view of what the element first looked like, when it was seeded.

As application development progresses, application system elements then, evolve from their originating seed, and additional versions are created (such as version 1.1 in the figure).

During the course of a project, a new version of the enterprise model may be published to the development workarea (such as version 1.0.1.1 in the example). It is up to the project team and the enterprise data administrator to determine if new versions of enterprise elements should be incorporated into the existing application model. If so, the new seed elements are included in the application container, in the ENTERPRISE SEED branch. A selective merge of the desired element properties is performed by the application development team, to update the latest version of the element in the application development branch (MAIN). The merge direction is always from ENTERPRISE SEED to MAIN.

## Quality control

The evolution of application components which originate from an enterprise model component is a controlled evolution. Please refer to the Data Architecture Framework document for a description of the types of changes that are allowed (and expected) to be made to these elements.

Based on the set of allowable changes, the IMG will periodically validate the integrity of the seeded elements, using an automated process that compares the application element with its seed, identifies the changes, and validates the changes against an allowable set of change types. (Note: this is harder than it should be. must be done in the scope of a "GISP for enterprise models" project.).

# Metadata synchronization

When business applications are developed using the GAME infrastructure, the metadata that represents the application models is actually stored in two repositories, namely Designer 6i and the GAME repository.

In order to achieve configuration management of such applications, it is necessary that either the version dimension is added to the GAME repository model or that the Oracle SCM repository is able to see all metadata stored in GAME, for the purpose of packaging this metadata as an integral component of the application.

The solution adopted by the ITMB is to make GAME metadata visible to Oracle SCM by generating equivalent content in Designer 6i. Where possible, element properties are used. When no suitable properties are found, files are generated, containing GAME metadata in an XML format.

GAME offers synchronization tools that will perform round-trip metadata synchronizations. The detailed documentation for these tools is not in the scope of this documentation site. It can be obtained by ...

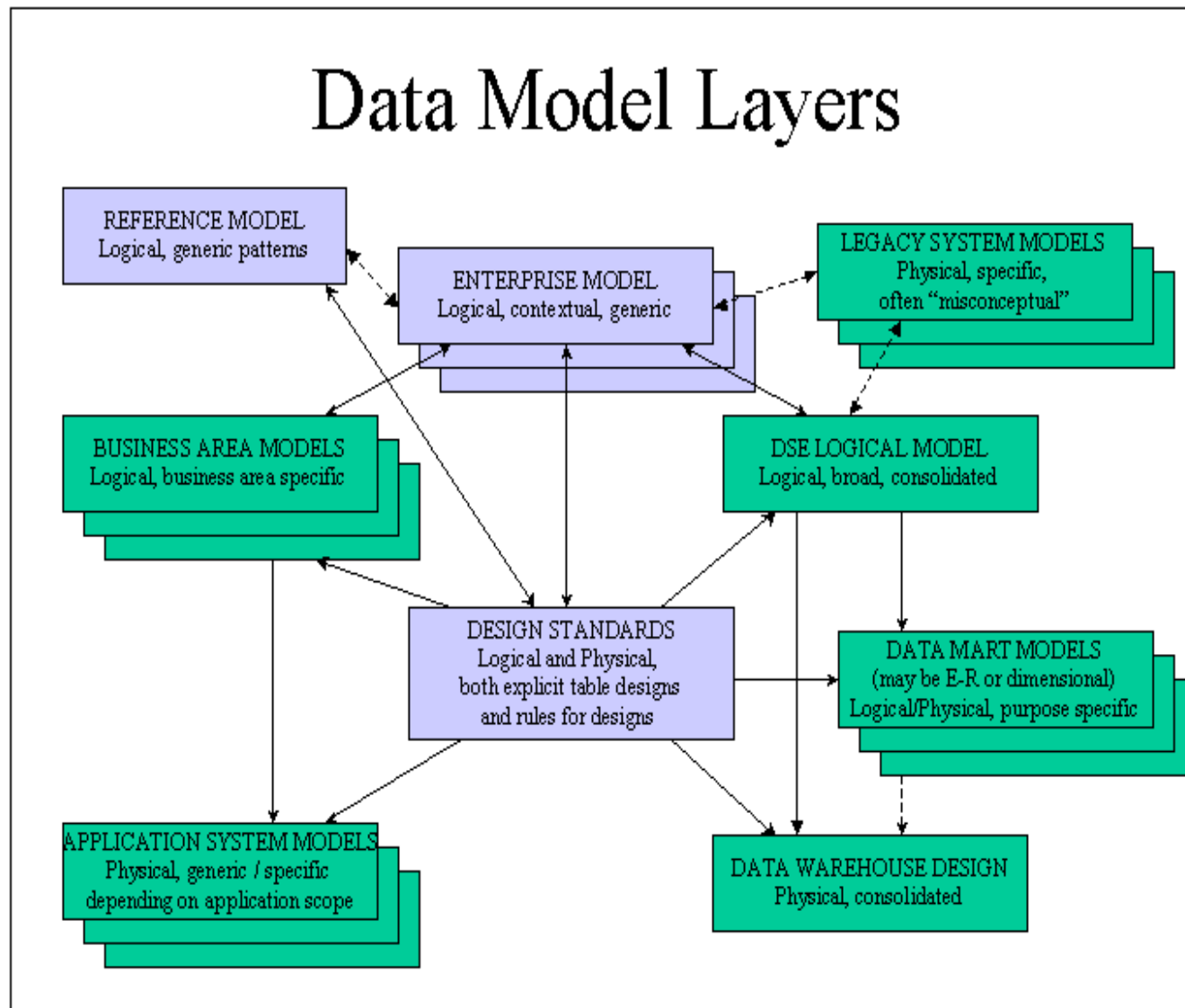
# Enterprise model development

This section describes the environments currently in place to facilitate the development of modeling standards.

There are currently three levels of modeling standards maintained by the IMG. These levels are known as the "reference model" level, the "enterprise model" level and the "design standards" level. A detailed description of these models, their objectives, scope and applicability can be found in the following documents:

- Data Architecture Framework - specify latest version
- other?

The figure below illustrates the relationships among models of all levels. The levels of interest in this section are identified in blue.



## Container organization

Each modeling standard (layer) is managed in its own container, with the exception that the design standards are actually managed within the same container as the model layer they refer to.

The container names are:

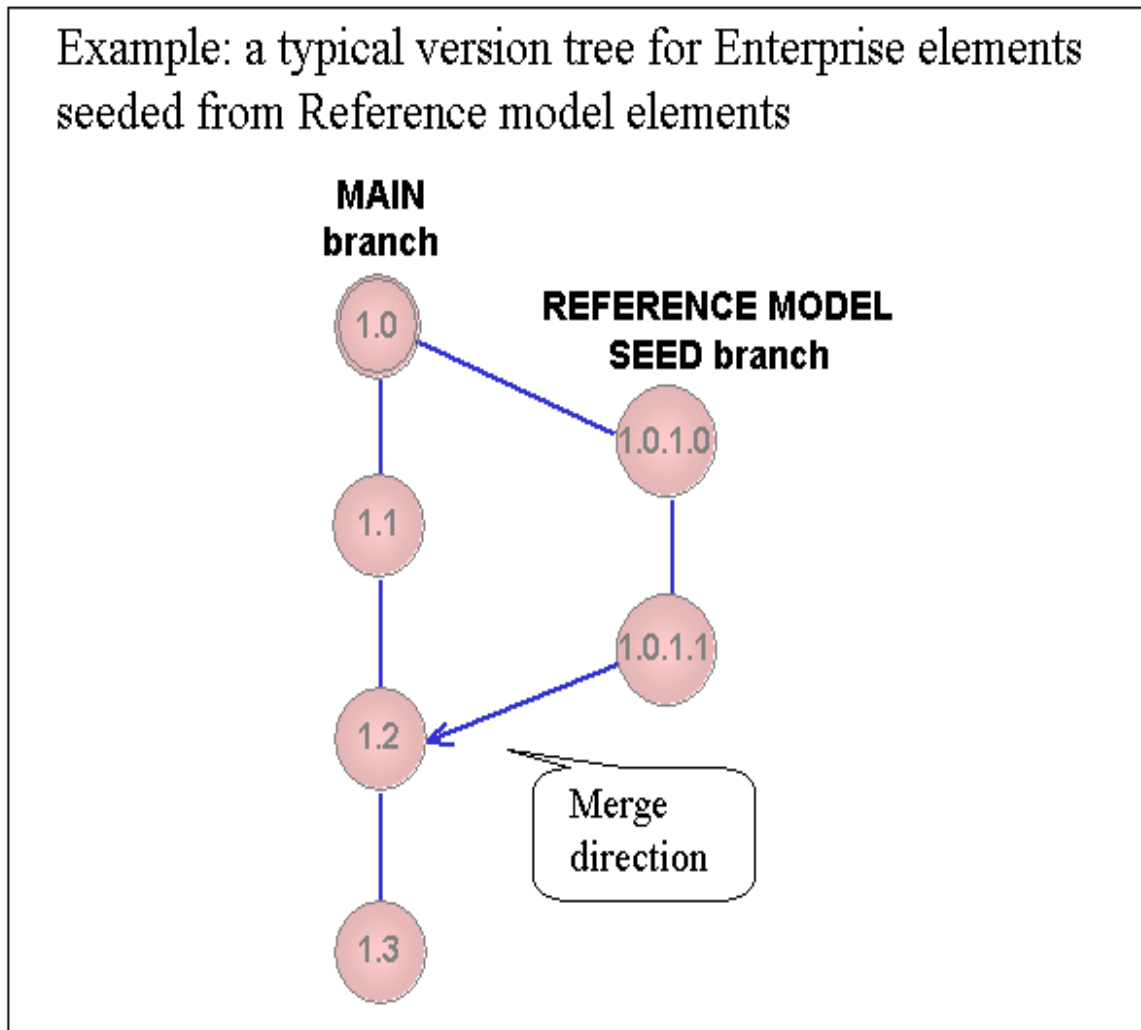
- REFERENCE MODEL
- EDUCATION ENTERPRISE
- ADVANCED EDUCATION ENTERPRISE
- LABOUR ENTERPRISE

## Development environment

The development environment within Designer is the WA\_CORPORATE\_STANDARDS.

## Traceability between model levels

The traceability between model layers, where applicable, is achieved by using versioning branches in the "receiving" model layer. For example, the enterprise model layer uses a branch called "REFERENCE MODEL SEED" to store the element versions originating from the Reference model. Element definitions from the reference model branch can be selectively merged into the MAIN development branch as desired. The Compare Tool can be used to find the differences between versions in the MAIN branch and the seed branch.



This mechanism is also used to ensure traceability between the enterprise model layers and the business area models.

## Publication of standards models

The publication of standard models to the application development environments is explained in [Enterprise Models and their use in application development](#).

In preparation for publication, the model custodian needs to:

- check-in all elements that should be published
- ensure that the set of elements to be published are all contained in a Designer "grouping element type" such as a User Defined Set (UDS) or a Diagram
- notify the SCM unit that a new release can be built and published

The SCM unit will then:

- create a configuration based on the diagram or UDS provided
- include the configuration in the application development workareas. This will replace any previously published versions of elements with the versions in the configuration
- create copies of the new elements into the business application's ENTERPRISE SEED versioning branch, so that it is available for merge into the application's own element definitions

# GAME infrastructure development

This section documents the characteristics of the GAME development environment which are unique to GAME. Links to the standard ITMB development environment documentation are provided where they are applicable to GAME as well.

[The GAME promotion model](#)

[GAME-related containers and database schemas](#)

[GAME Release management](#)

[Publication and use of GAME components](#)

[Impact analysis](#)

Development scenarios and process descriptions

- [Use case 1: Build new GAME component](#)
- [Use case 2: Develop GAME patch release](#)
- [Use case 3: Implement GAME patch release into business application environment](#)
- Use case 4: Implement GAME major release into business application environment
- Use case 5: Establish new GAME component constellation membership
- [Use case 6: Establish a new GAME parallel development environment](#)
- Use case 7: Merge enhancement or bug fix into other version of component
- Use case 8: Rollback a major GAME release from the business application environment

## The GAME promotion model

GAME infrastructure development follows the ITMB standard promotion model for application development. For information on this promotion model, please refer to the section [Promotion Models for application development](#). The actual promotion level workareas in Oracle SCM/Designer however, are GAME-specific, in order to seclude GAME development from general application development environments.

It is anticipated that the ITMB will have to maintain at least two releases of GAME in parallel for periods of time. To facilitate this parallel development, each release variant will be stored in a versioning branch specific to that release, while new development is managed on the MAIN branch. Bug fixes or enhancements can be merged as needed.

The workareas representing promotion states used by GAME developers are listed below:

WA\_GAME\_<release identifier>\_DEVELOPMENT

WA\_GAME\_<release identifier>\_INTEGRATION\_TEST

WA\_GAME\_<release identifier>\_ACCEPTANCE\_TEST

WA\_GAME\_<release identifier>\_PRODUCTION

WA\_GAME\_<release identifier>\_TRAINING

WA\_GAME\_<release identifier>\_EMERGENCY

# GAME-related containers and database schemas

## Designer containers:

Application system containers used by GAME developers are listed below. These containers go through promotions following the ministry promotion model:

**GAME:** contains all GAME components, but no business related metadata

**GAME\_OLTP:** contains a sample OLTP application system that is used to test GAME enhancements for OLTP applications.

**GAME\_OLAP:** contains a sample OLAP application system that is used to test GAME enhancements for OLAP applications.

At this point it is unknown whether or not the sample applications will need to "know about" GAME elements (metadata or source). If metadata is required in the application container, it needs to be copied into the container, not shared. Source code can be shared or simply visible in the same workarea.

## Database schemas:

In the database, the following schemas will be in use. These schemas will exist in each promotion level.

For GAME development:

- **GAME\_<release identifier>:** contains all GAME tables and views, but no business tables and views
- **GAME\_<release identifier>\_APP:** contains all GAME stored packages, procedures and functions
- **GAME\_<release identifier>\_AQ:** contains database queues. This schema is shared by all applications using the GAME infrastructure. Each application has its own set of queues.

For testing GAME in an application context:

- **GAME\_<release identifier>\_OLTP** and **GAME\_<release identifier>\_OLTP\_APP:** the test application for OLTP enhancements
- **GAME\_<release identifier>\_OLAP** and **GAME\_<release identifier>\_OLAP\_APP:** the test application for OLAP enhancements

# GAME Release management

## Criteria for determining types of releases of GAME

### GAME component releases:

Game component releases can be of the following types:

- **major release:** contains changes to GAME that include changes in its metadata structures. Major releases will likely require upgrade processes that include some degree of data conversion for transporting existing application metadata into a new metadata format.
- **minor release:** contains changes that do not involve structural metadata changes. These can generally be adopted by applications with a simple deployment process, usually source code deployment only, and possibly creation /update of metadata instances.
- **emergency fix release:** contains fixes to business-critical bugs. Emergency fixes do not involve change in metadata structures.

### GAME infrastructure releases:

GAME infrastructure releases identify a "constellation" of compatible component releases. The types of releases follow the same basic classification as GAME component releases (see above). Constellations can be thought of as "configurations of configurations", and like configurations, they are versionable elements.

## Release branches

Until such time as parallel development becomes a requirement, all element versions are stored in the MAIN branch. When it is deemed that a release of GAME must be maintained separately from new development, that release configuration will be made available in a workarea specific for that release. This new workarea will maintain new versions of elements in a specific release branch. But fixes or enhancements that are relevant across releases can be merged using the Merge tool in Oracle SCM.

A pre-requisite for parallel development is that releases are tested for completeness and correctness, as mentioned in [Project SCM tasks](#).

# Publication and use of GAME components

The publication of GAME components to environments where they can be used by business applications is explained in this section. The issue of publication needs to be addressed in all technology layers in use by the business application. Currently the technology layers that are serviced by the ITMB are:

- Metadata Repositories: Designer and GAME
- Oracle Database
- File system (LAN or Unix)
- Application server

The sections that follow identify, for each technology layer listed above, the environments and procedures that have been put in place for projects using the GAME infrastructure.

## **Metadata Repositories: the Designer repository**

In the Designer repository, GAME releases are published to promotion model workareas dedicated to development of applications using GAME. Each release of GAME that is supported by the ITMB will have its own set of promotion model workareas.

### **Publication procedure:**

The publication procedure involves:

1. creating a frozen set of element versions for those elements that make up the GAME component to be published. This set is also known as a configuration.
2. The component configuration is made available in the application development environment (workarea).
3. The GAME toolset provides a copy (or synchronization) tool that is used to copy GAME object definitions to the application container. (An actual copy is necessary because the types of relationships that are established between GAME objects and business objects need to be in the same container).
  - **Note:** The GAME copy tool creates new versions for all elements copied into the application container.

### **Maintaining GAME components intact:**

Metadata definitions originating from GAME should not be changed by application developers in the context of their application, as all GAME components use those definitions. However, Designer does not have the ability to cater for variations of access rights to objects within a container. Therefore a developer could unintentionally change GAME metadata. Given that we cannot prevent this situation from happening, we need to at least detect it soon after it occurs. To this end, a custom program will be developed to extend the repository functionality. This extension involves detecting any changes that have taken place in the application development containers, for any GAME element copied to that container. Any changes found by this process are identified in a report. This process is part of a set of nightly scheduled repository processes with email notification to the repository managers.

## **Metadata repositories: the GAME repository**

Unlike the Designer repository, the GAME repository is not shared by all applications. Each application making use of the GAME infrastructure has its mandatory schemas (see below) seeded with the GAME repository components at the beginning of the application development project.

### **Maintaining GAME components intact:**

Given that application developers will have full access to all application schemas during development and test, we need a way to ensure that developers will not make changes to the infrastructure components.

The promotion process that moves the application forward from developer test to user acceptance test (UAT) will not include promotion of the GAME structures themselves, but only of the metadata contained in the structures, which may be necessary in the destination promotion levels. Also, no promotion of GAME programs will occur by export from development or test. If the UAT environment needs an upgrade of its GAME programs, this upgrade will be done by extraction of the source code from the SCM repository.

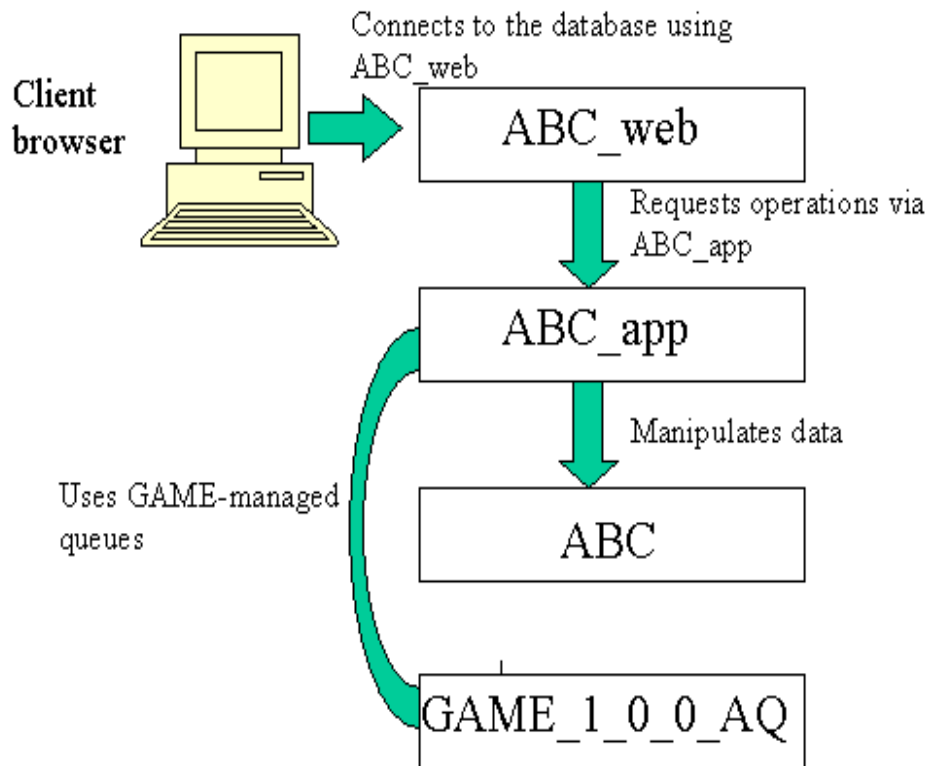
### **Oracle Database**

Business applications using the GAME infrastructure will require additional schemas as detailed in this section. There is also a difference in the content of some schemas depending on the promotion level because not all GAME-related information is required in promotion environments beyond the development and test environments.

For completeness this section lists all standard schemas currently implemented for applications.

Figure below illustrates the currently used schemas:

## Database schemas for a hypothetical application “ABC”



### Mandatory application schemas:

- **<schema\_owner>**: the schema that owns all database objects related to data (tables, sequences, table API), *including GAME objects*
- **<schema\_owner>\_APP**: the schema that owns all API tiers except the table API (including the view API)

Note: the table API consists of a set of triggers and packages for each table. The triggers are for before and after row, before and after statement, for insert, update and delete statements. The packages contain the procedures that validate table integrity and data rules specified in table and attribute properties in Designer.

### Optional application schemas:

- **<schema\_owner>\_HST**: (not shown in the above drawing) the schema that owns all Headstart database objects and API's

- **Note:** the current thought is that applications using the GAME infrastructure will not make use of the Headstart template package. The co-existence of both toolsets has not been tested by the IMG.
- **<schema\_owner>\_WEB:** the schema used for database connection for applications that opted to work within the "single Oracle account for all users" concept, as opposed to each user having his/her own Oracle account.

#### Shared GAME schemas:

Only one of these schemas exists per promotion level, and business applications may issue calls to these components in order to accomplish application tasks.

- **GAME\_<release\_identifier>\_AQ:** contains queues used by business application processes. Each application has its own set of queues.

## **File system (LAN or Unix)**

There are no GAME components currently residing on the LAN.

The GAME interface components (the "front-end") are a set of forms and reports available from the ministry's intranet site.

Each supported release of GAME has its interface components (forms, reports, libraries, graphics) deployed in their specific directory structures, for access through the intranet.

## **Application server**

The application server does not hold GAME components per se. It does, however, provide access to the available GAME environments, using Forms and Reports server technology.

Each supported release of GAME has its own URL.

## GAME Impact analysis

The scope of impact analysis for reusable components is not only the component interactions within GAME but also the use of the component by the business applications developed with the GAME infrastructure.

**For performing impact analysis within the GAME scope**, i.e. to find how a change in a specific GAME component will affect other GAME components, you can use the Oracle SCM impact analysis tool, called the Dependency Manager. Additional information about performing impact analysis within the scope of a container can be found in the [Impact Analysis](#) section of this site.

**For performing impact analysis of GAME changes on business application components**, a few additional steps are required.

**All GAME metadata** required by an application is actually copied into the application container using the GAME infrastructure, and thus the actual association of GAME elements with business applications per se does not exist.

In order to identify where a particular GAME metadata element is being used, the following questions need to be answered:

- What release configuration(s) does the element belong to?
- What are the workareas that are using the above configurations?
- What business application containers in the above workareas host elements with the same name as those that will require modification

The impact analysis should then be carried out in the workareas and containers identified by the above question, for the elements of the same name as the elements which are the subjects of change.

For GAME source code stored in files, it is possible to determine the applications using the files by looking at the "Referenced in Folders" association within the file node in the repository. Each application using GAME will have read access to the source files from its own container through a shortcut.

# Use case 1: Build new GAME component

Goal:	A component configuration exists and has passed approval stages. The component configuration is ready to be promoted to production environments.			
Pre-conditions:	Promotion environments exist and are ready to accomodate new development.			
Success end condition:	The component configuration is ready to be promoted to production environments.			
Failure end condition:	A working component configuration cannot be built.			
Primary actor:	Team Meta			
Trigger event:	A new component is required by a business application. The business application representative submits a new component request. <b>The new component request is approved.</b>			
Main success scenario:	Step#	Actor	Description	Tooling/mechanism
	1	Team Meta	User "opens" GAME container for change in the main GAME development area.	Check-out in workarea WA_GAME_<release>_DEVELOPMENT, versions stored on MAIN branch
	2	Team Meta	User(s), in SDLC fashion, develops the new component.	Any/all Designer tools, file editors, GAME, etc.
	3	Team Meta	Component configuration is created for submission to approval/testing stages.	User Defined Set element, Configuration element, GAME Set element
	4	IMG CM	Configuration is promoted according to promotion model	Workarea wizard
	5	Team Meta	Component is tested and changes are made. Iterations of steps 2-5 occur. Configuration is updated with fixes and versioned during iterations (i.e no new configurations are created, only new versions of the initial configuration)	Any/all Designer tools, file editors, User Defined Set element, Configuration element
		Team	No additional errors are found and component	

	6	Meta	configuration is ready for promotion to production
Scenario extensions:			
Scenario variations:			
Related information:			
Notes:	Assumption: approved new component requests will always be deemed usefull to all GAME releases, and thus will not be isolated to a particular release branch, but will be considered MAIN-stream development.		
Open issues:			

## Use case 2: Develop GAME patch release

Goal:	A patch release configuration exists and has passed approval stages. The configuration is ready to be promoted to production environments.			
Pre-conditions:	Promotion environments exist and are ready to accomodate new development.			
Success end condition:	The patch release configuration is ready to be promoted to production environments.			
Failure end condition:	A working patch release configuration cannot be built.			
Primary actor:	Team Meta			
Trigger event:	Number of outstanding non-emergency issues warrants the development of a new release.			
Main success scenario:	Step#	Actor	Description	Tooling/mechanism
	1	Team Meta	User "opens" GAME container for change in the main GAME development area.	Check-out in workarea WA_GAME_<release>_DEVELOPMENT
	2	Team Meta	User(s), in SDLC fashion, develops bug fixes or enhancements.	Any/all Designer tools, file editors, GAME, etc.
	3	Team Meta	Patch release configuration is created for submission to approval/testing stages.	User Defined Set element, Configuration element, GAME Set element
	4	IMG CM	Configuration is promoted according to promotion model	Workarea wizard
	5	Team Meta, IMG CM	Patch release is tested and changes are made. Iterations of steps 2-5 occur. Configuration is updated with fixes and versioned during iterations (i.e no new configurations are created, only new versions of the initial configuration)	Any/all Designer tools, file editors, User Defined Set element, Configuration element
	6	Team Meta	No additional errors are found and component configuration is ready for	

	promotion to production
Scenario extensions:	
Scenario variations:	
Related information:	If any improvements developed in this patch release are relevant to another currently maintained GAME release, these can be merged to other release elements using Use case 7: "Merge enhancement or bug fix into other version of component".
Notes:	
Open issues:	

## Use case 3: Implement GAME patch release into business application environment

Goal:	The components of a patch release configuration can be found in the business application environment.			
Pre-conditions:	The patch release has successfully passed through all approval stages. The business application environment is ready to receive the patch.			
Success end condition:	The components of a patch release configuration can be found in the business application environment.			
Failure end condition:	The components of the patch release cannot be found in the business application environment.			
Primary actor:	IMG Configuration manager			
Trigger event:	The patch release has obtained the last level of approval (UAT, QA).			
Main success scenario:	Step#	Actor	Description	Tooling/mechanism
	1			
	2			
	3			
	4			
	5			
	6			
Scenario extensions:				
Scenario variations:				
Related information:				
Notes:	This scenario is pending resolution of issue: whether or not GAME elements need to be visible to the application.			
Open issues:				

## Use case 6: Establish a new GAME parallel development environment

Goal:	Additional development environment is available for work on a non-MAIN stream release			
Pre-conditions:				
Success end condition:	The new development environment is accessible by the team and contains the appropriate seed elements.			
Failure end condition:	The new development environment cannot be provided to the team.			
Primary actor:	IMG CM			
Trigger event:	A new major release of GAME needs to be developed while the existing major release is still in use and requires a support environment.			
Main success scenario:	Step#	Actor	Description	Tooling/mechanism
	1	GAME CM	Determine the GAME configuration that needs to be made available in the parallel development environment.	RON, Configuration element
	2	IMG CM	Create a workarea to host the GAME configuration provided as seed. This workarea will allow versioning of its contents on a branch named after the release that is being maintained in the workarea	Workarea wizard
	3	IMG CM	Grant appropriate user access to the workarea	RON
	4			
	5			
	6			
Scenario extensions:				
Scenario variations:				
Related information:				
Notes:				
Open issues:				

# File upload and download from the repository

Before you can version files in the repository, you have to upload them into the appropriate sub-folder within the application you're working on.

Every application has **two sub-folders: "CM\_non\_generated" and CM\_project\_deliv"**, for holding non-generated application components and CDM project deliverables, respectively.

The "CM\_non\_generated" folder has a number of sub-folders to cater for the typical types of files that applications require. Additional sub-folders can be created by the repository manager if required.

The "CM\_project\_deliv" folders have a number of sub-folders which correspond to Oracle CDM processes. Project deliverables should be placed in the appropriate folder prior to versioning.

## **The "mechanics" of uploading and downloading:**

File upload and download are done in the Repository Object Navigator (RON).

**To upload a file**, navigate to the folder you want the file to be in, and select the file if it already exists. Then select "Upload Files and Folders" (either from the Utilities menu, or from the right-mouse-click menu).

- Note: if you're uploading a modified version of a file that is already under version control in the repository, the file should be in check-out state, and you should select the "overwrite changed files" option on upload.

**To download a file**, select the file. Then select "Download Files and Folders" (either from the Utilities menu, or from the right-mouse-click menu). If you intend to make changes to the file, you should check-out the file as well, to indicate to others that the file is being worked on.

It is possible to upload and download multiple files at a time, by applying the same functions to a group of selected files.

## Web browser access to the Designer repository

The ITMB provides access to Designer 6.0 and 6i information from a web browser, in addition to the usual client-server interface.

Access through the browser interface is read only. The tool that supplies this access is called ODWA (Oracle Designer Web Assistant).

For browsing Designer 6.0 applications, the ITMB supplies a generic account. Please request account information from the RM/CM Unit contact person (see Contact Information link).

For browsing Designer 6i applications, you can use your Designer 6i account if you have one. You will have access to the same workareas and containers as you do via the client-server front end.

Occasional users who require browser access to Designer 6i applications should request it from their Business Analyst.

To access the ODWA tools, go to the [I-Development Accelerators web site](#), and select the appropriate link.

# Frequently Asked Questions (FAQ)

## **How can I easily find out what objects I have checked out in a container?**

You can use the List Check-outs utility. To run this utility, follow these steps:

- first, select the container you're working in
- select List Checkouts either through the Version menu or via mouse right-click
- select "by me"
- make sure that both the "Checked out objects" and " Non versioned objects" checkboxes are set so that it picks up new elements that have never been checked-in too
- run the utility by selecting "Ok". It will return a pop up window with the list of findings

From this utility you can do bulk check-ins of all your checked-out objects.

## **Does the ITMB support private workareas?**

No. There has not been a true business need for private workareas. Generally, teams working in the ITMB repository work in groups, and benefit from immediate visibility to each individual's work. Also, teams are geographically close together, and communicate well as to what elements are being worked on by each team member.

## **Why can't I delete elements in 6i?**

Yes, sometimes it does appear that you can not delete elements in 6i. The delete function behaves differently once an element has been versioned. Prior to versioning, you should be able to use the Delete or Force Delete functions as you have in previous versions of the tool.

Once an element has been versioned (one or more times), other issues come into play. The actual action of a delete can have a few interpretations, and thus the delete function requires some additional information, for example:

- do you want to delete all versions of this element, or just the version you see in this workarea?
- do you want to remove this element from this version of its container (but leave it in the previous version(s) of the container?)

The Designer on-line help actually has good information on Delete functionality. You should read the pages for the topic "About deleting and purging repository objects" and related pages.

## **If I check-out an object, will other users be prevented from modifying it?**

No. It will prevent other team members from checking it out again, but they will be able to modify it and also check-in that object.

The repository has been set up to allow for this on-purpose. The dynamics of development and maintenance teams working in the ITMB repository are such that they generally communicate often, and also work as a team on a particular release. Therefore, if you have checked-out a set of entities, say, you and your team mates will be working together on those, and you both should be able to change it.