APPLICATION DELIVERY STANDARDS

Version 3.1.1 January, 2012

Table of Contents

VERSION CONTROL

This section of the document is to be used to control the various versions or releases of the document.

Version	Description	Distribution	Date	Author	Organization
3.0.0	Subversion Implementation	Full document	2010-03-31	CC	IMB Deliveries
3.1.0	VMAD Implementation and repository consolidation	Full document	2010-10-15	CC	IMB Deliveries
3.1.1	Deployment and Configuration file changes; typos	6.2, 6.3, 6.4, 6.10, APPENDIX A, B, C Typos	2012-01-16	FV-KM	IMB Deliveries

1. INTRODUCTION

This section outlines the Audience and Purpose of this standards document.

1.1 Audience

This document is directed at those who will be delivering corporate applications to the Information Management Branch of the Ministry of Environment and/or Ministry of Agriculture (hereafter collectively referred to as “the Ministry“) corporate infrastructure. Typically, these are ***Vendor***s, i.e. private sector application developers. In addition, this document may be of interest to Business Portfolio Managers and Ministry Application Administrators.

1.2 Purpose

This document outlines the standards which must be followed when delivering new and updated corporate applications to the Ministry. This includes all applications that will be deployed in the Ministry's UNIX/Oracle, Terminal Server or Windows Server corporate infrastructure.

"Delivery" is in fact two operations: "Deposit" and "Deployment". Deposit is the task of adding software to the Ministry Code Archive and Deployment is the act of configuring and deploying that software in an operational environment.

1.3 References to Other Documentation

The Ministry has a number of standards pertaining to application development and the system development lifecycle (SDLC).

2. APPLICATION DELIVERY ROLES

The delivery of an application rarely requires the talents of a single resource, although an individual may perform many roles. The following is an explanation of the roles played by team members in the application delivery process.

2.1 Application Administrator — *responsibility: Ministry*

- responsible for the ongoing operations of the application
- co-ordinates with the ***IMB Business Portfolio Manager*** to schedule delivery of new versions or releases
- responsible for the User Acceptance Test (UAT) process
- verifies that the application functionality adheres to Ministry standards

2.2 Vendor Delivery Personnel — *responsibility: Vendor*

- Vendor staff members responsible for delivering applications to the Ministry
- must designate one individual as the primary contact and Vendor delivery specialist and this person must be trained in or familiar with the standards and procedures
- responsible for running all data conversion scripts against all instances
- primarily responsible for deploying onto the DLVR environment

2.3 IMB Deliveries — *responsibility: Ministry*

- provides for communication to Infrastructure and Database and Middleware Services for administrative support for IMB Deliveries responsible applications
- responsible for ensuring that a stable application environment is available for the deposition and for DLVR deployment of IMB Deliveries responsible applications
- performs or dispatches to Database and Middleware Services tasks associated with an application delivery
- responsible for performing all aspects of the deployments to test, production and training (except for data conversion)
- ensures that Ministry standards and guidelines are followed for the application delivery

2.4 External Project Manager — *responsibility: Vendor*

- primary Vendor contact for the application
- accountable for the delivery process
- accountable for ensuring that the application delivery adheres to Ministry standards

2.5 IMB Business Portfolio Manager — *responsibility: Ministry*

- coordinates the Ministry and the ***Vendor***
- directs that Ministry standards and guidelines are followed for the application delivery
- will schedule IMB resources required for the delivery process

3. SOFTWARE CONSIDERATIONS

3.1 Oracle

Please refer to the Systems and Application Technology Standards document for the currently supported development environment for applications that connect to an Oracle database.

3.2 Web Browsers

All HTML documents are to conform to the BC Government Internet Standards. Government web browsers have certain restrictions to features that can be used in an application. An example of a restriction is the blocking of Active-X controls. The standard web browser used by the Ministry is Microsoft Internet Explorer.

3.3 Java Components

Please refer to the Systems and Application Technology Standards document for supported component versions.

- All applications must adhere to the BC Government Web Development Standards.
- All applications must adhere to the Ministry's Java Application Development Standards

4. FILE MANAGEMENT and CODE ARCHIVE

This section describes the directory structures on Ministry application servers. These specified patterns are to be used both for code archive structuring and for deployment locations.

4.1 File Naming Conventions

All file and directory names must be UNIX compatible and must conform to Ministry restrictions. Long UNIX-compatible names may be used. However, as UNIX is case sensitive, all filenames must be in lowercase and must not contain blanks or '$', '[,]', '{,}', '(,)', '&','!','>,<' Following are exceptions to this convention:

- *.Z extension for files compressed with the UNIX ***compress*** utility;
- Java components can be mixed case where required;
- certain Java deployment directory names; and
- certain Java deployment files (e.g.: MANIFEST.MF)

The following conventions for filename extensions must be used:

File type	Description
.c	C source code file
.cab	Microsoft cabinet file
.ccs	Designer generated code-control sequences
.class	JAVA class library
.con	Designer generated constraints
.css	Cascading Style Sheet
.dbl	Database Link specifications
.dll	Windows Dynamic Link Library

.doc	Microsoft Word document
.dmp	Oracle*CASE (Designer) dump file
.ear	Enterprise Archive file
.fmb	Oracle Forms form source file
.fmx	Oracle Forms compiled form
.fnc	Designer generated function
.gif	GIF file; WebForms icon file
.grt	Designer generated object grants
.gz	GNU zip format
.h	C header file
.hlp	Microsoft Help file
.html	HTML document
.ico	Icon file
.ind	Designer generated indexes
.jar	Java Archive file
.java	JAVA source file
.jav	Designer generated JAVA code
.ksh	UNIX Korn Shell script
.jpg	JPEG file
.js	Java Script file
.jsp	Java Server Page file
.log	Output from UNIX application software
.lst	Output from SQL scripts

.MF	Java Deployment Manifest file
.mmb	Oracle Forms menu source file
.mmx	Oracle Forms compiled menu
.pc	Pro*C source code
.pck	Designer generated releases
.pdf	Adobe portable document format
.pkb	Designer generated release body
.pks	Designer generated release spec
.pl	PERL script
.plb	PL/SQL release body
.pll	Oracle Forms library source file
.pls	PL/SQL release specification
.plw	PL/SQL release body that has been wrapped
.plx	Oracle Forms library compiled file
.prc	Designer generated procedures
.rdf	Oracle Reports source file
.rep	Oracle Reports compiled report
.rgp	Refresh Group definitions
.rgr	Designer generated roles and role grants
.rle	Designer generated roles
.rol	Designer generated roles
.seq	Designer generated sequences
.sh	Unix Bourne Shell script

.snl	Designer generated materialized view logs
.snp	Designer generated materialized views
.sql	SQL script
.sqs	Designer generated sequences
.syn	Designer generated synonyms
.tab	Designer generated DDL for tables
.tld	Java tab file
.trg	Designer generated triggers
.tsp	Designer generated tablespaces
.txt	ASCII text file
.tyb	Designer generated object types (with methods)
.typ	Designer generated object types or collection types
.ugr	Designer generated users and grants
.usr	Designer generated users
.vw	Designer generated views
.xml	Extended Markup Language file
.zip	Windows WinZip file
.Z	UNIX compressed file format

If a required file type is not listed in the table above, please contact the ***IMB Business Portfolio Manager*** for resolution. Modules with a non-supported file type extension will not be accepted as part of the application delivery.

Note that there are a number of different possible extensions for some of the types; this is to accommodate changes in the different versions of Designer. When generating with Designer, always use the default file types it creates.

4.2 Ministry Naming Conventions

The <application name> is the application acronym. See Naming and Describing standards Section 4.3 for definition of an acronym.

4.2.1 Release Labels (Numbering)

Release labels are of the format ##.##.## where the first '##' represents a major release, the second '##' represents a minor release, and the third '##' represents a patch release.

The ***External Project Manager*** and the ***Application Administrator*** should discuss with ***IMB Deliveries*** if this version of the application is a major, minor or patch release. IMB Deliveries will select the release label (numbers) in order to provide for optimum release organization through the delivery process.

Major Numbers are to be used when the application significantly changes architecture, business functionality or end user organization.

Minor Numbers are to be used when the application changes business functionality.

Patch Numbers are to be used when the application requires fixes or configuration changes, but no change to business functionality is intended.

4.3 UNIX (Deployment) Directory Structure

An application specific account (e.g. coors, ems, ers etc.) will be created on the UNIX delivery server(s) for each new application. This must be coordinated through the ***IMB Deliveries***. Application home directories are named /apps_ux/<appName>. ***Vendor***s can create other directories under $HOME/ and the following directories will typically be created by the ***IMB Deliveries*** prior to the application delivery:

Directory Name	Description in Use
$HOME/stage	a **temporary** location for building deployable objects, such as a compiled form, a compiled report, a configured CRON job or a application.ear file
$HOME/admin	deployment location for CRON executables and administration scripts or programs
$HOME/bin	Compiled and deployed server-side executables such as server-side reports, run time scripts, and 'C' programs
$HOME/logs	logs for manual processing or CRON

4.3.1 Code Archive (Subversion) Directory Structure

Directory	Description
<appName>/trunk/	Master media image of DLVR level source code. This directory must contain only configuration items (source code, readmes, Windows deployer releases, etc.)

<appName>/tags/	Subversion Tags will be created only by the IMB Deliveries. Tags are created as part of the change control process, at the end of completed deployment to production

There will be a series of sub-directories under trunk/ in the code archive and under /apps_ux/<appName> in deployments. The structure of these sub-directories is described below:

Directory	Description of Files
/admin	Scripts that will be used on a regular basis
/docs	Readme file deployment instructions. Text files only
/documentation	All documentation regarding all software, excluding readme files. .doc and .xls are acceptable as well as .txt
/bin	Client-side executables
/help	Microsoft Help files or HTML help and image files for Client/NTTS-side help
/html	HTML files and images for server or web-based presentation
/icons	Icon files for client server Oracle Forms **(restricted use)**
/misc	Miscellaneous client-side files including Word and Excel templates
/reports_server	All source files for reports-server reporting
/scripts	Scripts that will be used (typically only once) on delivery i.e. database objects, patch scripts, and DML scripts
/web-src	Java application web source
/webforms	All source files related to Oracle WebForms. ie: .fmb, .pll, .mmb files
/webicons	Icon files for Oracle WebForms
/bin	Location of all executable files.
/dbsrc	Location of all Pro*C pre-compiled files.
/h	Location of all header files
/obj	Location of all object files.

/src	Location of all source files (Pro*C and C)

4.3.2 Code Repository Source Directory Structure (File System)

/apps_source/source/<appName>	Master media image of DLVR level source code. (In the near future this image, and the TEST and PROD source images, will be maintained in Subversion.) This directory must contain only source code.

The sub-directory structure for the File System source archive is identical to the Subversion structure under "trunk".

The source for the latest DLVR level version of the application will reside in the source directory structure.

4.3.3 Code Repository Release Directory Structure (File System)

Subversion managed Code Repositories do not use release packages. Section 4.4.3 applies to file system managed Code Repositories only.

/apps_source/release/<appName>	Delivery package archive.

The release directory contains a series of packages of delivered software related to the application. Release package files get copied into the corresponding /apps_source/source/<appName> directory as part of a delivery.

Release packages are compressed tar files containing a single directory (with subdirectories). The directory must be named ##.##.## identical to the release label. The directory has subdirectories exactly as structured for source. (see above)

NOTE: Migrations (including patches) must not be executed from the Release or Source directories; all of source must be copied to the stage directory, compiled for deployment and scripts executed from there.

4.4 Text Files

All configuration items that are ASCII text files delivered to the Ministry must be in UNIX compatible format. UNIX line endings must be used and each text file must end with a final line ending control character. It is important that text files be converted from Windows text format to UNIX ASCII format. Consider using dos2unix, a standard UNIX utility, or fixcr, a Ministry utility. Do not depend on FTP to reformat files. Consider using findDOS.ksh, a Ministry utility that tests for format.

4.5 XMI Export Files

Any time a UML Modeling Tool (e.g. Enterprise Architect, Rational Rose, Visio) is used, an XMI export of the model must be provided. This XMI Export File must have a root release named <application name> Model, as in "ABC_Model", with the following sub-releases:

- Business Process Model
- Use Case Model
- Domain Model
- Class Model
- Logical Persistence Model
- Physical Persistence Model
- Component Model
- Deployment Model

XMI Export files must be delivered as part of a release, and must be added to a "xmi" directory under /apps_source/source/<appName> in the file system or under the Subversion trunk. XMI Export files should also be emailed directly to the IMB DA on request from the Ministry during the QA or review process. In addition, the tool-specific schema descriptor (e.g. Document Type Definition, or XML Schema in the future) file must be included in the /xmi folder. Typically, it has a name such as `uml.dtd`, or `uml_ea.dtd`.

XMI Exports must be in XMI Version 2.1.

5. PRE-DELIVERY TASKS

The following pre-quality assurance and delivery tasks should be followed in order to ensure a smooth delivery.

5.1 Scheduling QA and DLVR Deployment

Prior to any delivery, scheduling of the quality assurance and the deployment should be discussed between the IMB Business Portfolio Manager, Application Administrator and the External Project Manager. Once target dates are set, the IMB Business Portfolio Manager is responsible for submitting a booking to VMAD which must include bookings for Quality Assurance, DLVR deployment, TEST deployment and PROD deployment.

Delivery of changes to applications or new applications proceeds from commit of new code in Subversion (or application of a release package to source in the File System archive) through QA through deployment to DLVR. Deposit of the delivery release (Subversion commit) provides the ***IMB Deliveries*** with the release readme file, scripts and other files prior to DLVR deployment. Please allow 3 - 5 business days for the QA task prior to the DLVR deployment task.

The IMB Business Portfolio Manager will edit QA and DLVR deployment time slots. Normally, time slots are 1.5 hours; in the event of a large or complex changes, additional time must be booked if there is a reasonable expectation that QA and DLVR deployment cannot be completed in two 1.5 hour periods.

The expectation is that the QA and deployments will be ready to start at the beginning of the time slots, and be completed by the end. It is important that the commit to Subversion be completed before the beginning of the QA time slot and notice of QA readiness be sent from the ***Vendor*** or IMB Business Portfolio Manager to IMB Deliveries. If there is a requirement for deployment involvement from the ***IMB Deliveries***, it is important that ***Vendor***s keep within the DLVR deployment allotted time frame.

If, due to last minute circumstances, a delivery will be unable to meet its time slots, the ***External Project Manager*** must notify the ***IMB Business Portfolio Manager*** and ***IMB Deliveries*** as soon as possible of this fact and to arrange for rescheduling; arbitrarily deciding to deliver later that day or the next is not acceptable.

5.2 VPN Access

In order to access the BC Government network, the ***Vendor*** must install and configure the Virtual Private Network (VPN) software from Cisco. The VPN software can be obtained from SPAN BC.

In addition to the VPN software, a SPANDial account is required to access the BC Government network. SPANDial accounts can be requested through the ***IMB Business Portfolio Manager***. SPANDial accounts expire on March 31st of each year; requests for renewal of SPANDial accounts should be directed to the ***IMB Business Portfolio Manager*** responsible for the project.

5.3 Planning Data Conversions

Data loading and conversion, such as when a new system is implemented to replace an older system, is the responsibility of the ***Vendor***. Scheduling becomes especially important when the application is migrated to the testing, production and training environments as the ***IMB Deliveries*** will be the one actually building the application, but not converting the data; therefore coordination between the ***External Project Manager*** and the ***IMB Deliveries*** is vital to ensure that the different processes are completed in their proper order and at appropriate times.

5.4 Warehouse Data Loading

All data to be loaded into the Ministry's data warehouse must be modeled and approved by both the Ministry's and ILMB/GeoBC's Data Administration sections. After approval the data must be provided to ***IMB Deliveries*** to be loaded into the warehouse following the normal delivery process.

Data must be submitted in SQL 'insert' scripts unless prior approval is given from the ***IMB Deliveries***.

6. DEPOSIT AND DEPLOYMENT

Deployment to the delivery environment of an application involves a number of important tasks. These tasks are detailed in a release specific "readme" text file. It is the responsibility of the ***External Project Manager*** to ensure that the delivery instructions prepared by the ***Vendor Delivery Personnel*** are accurate. Accurate delivery instructions are required to ensure that no problems arise when the ***IMB Deliveries*** deploys the application to the test or production environment.

It is vitally important that the ***Vendor Delivery Personnel*** follow all instructions in the readme file created with the delivery, to ensure accurate and complete deployment.

6.1.1 How to deposit new or modified code into Subversion

1. Check out the application code to a temporary working directory, usually, /apps_ux/<appName>/stage. The UNIX svn co command is one way of doing this.
2. Work within your stage folder to affect your changes (unit test, system test)
3. Commit the changes to Subversion. The UNIX svn commit command is one way of doing this.
4. Always provide messages to explain why you are depositing. The UNIX svn command accepts -m "place message here" options.
5. It is good practice at this point (after commit) to clear the ../stage folder
6. Notify IMB Deliveries that QA may commence by entering a VMAD NOTIFY version status record.

Please note: Tags will not be created at this time. Tags are created by IMB Deliveries when an application completes PROD deployment.

Caution: It is important that code that is not part of this release cycle not be included in the commit. In the development site, keep development code from each release separate from the next to ensure it is not committed to the code archive prematurely.

Regarding deletion of code items: Subversion does not actually ever delete anything. To remove a file from the code archive ***Vendor***s must record each deletion in the first section of the readme (e.g. "rm script123.sql") and also must use the Subversion delete function on the file. The file will then not appear in the Subversion head version and IMB Deliveries will remove it from deployed infrastructure.

Every Subversion operation, including commits, must have a message attached. There are no IMB Deliveries imposed restrictions on the message contents. They are not used in automation but are for human understanding only.

Please note: Subversion delete does not actually delete files, it simply masks it from the head version. If a real deletion is required, in the case of a large accidental data dump being included or for legislative reasons, for examples. IMB Deliveries will effect this by rolling the archive back to a previous archive backup.

6.1.2 How to deposit new or modified code into File System Code Archives

6.1.2.1 Creation of the Release package

The first step in any delivery is to create a release package. Every release package should be structured as a single file in UNIX ***tar*** format. The format of the tar file name should be <appName>.<version>.tar. The <version> portion of the delivery file is of the format #.#.# where the first # represents a major release, the second # represents a minor release, and the third # represents a patch release. Use the UNIX ***tar*** command to create a ***tar*** file that includes all the files you wish to deliver, but do not include the directory path above the version directory. A **.tar* extension must be included in the naming of the ***tar*** file.

e.g. coors.4.0.1.tar
app_name=coors, major_release=4, minor_release=0, patch_release=1

Each ***tar*** file must be compressed using the ***compress*** or ***gzip*** utilities. These utilities will add the (***tar***) .Z or (***gzip***) .gz extension automatically for you.

Examples:

tar -cvf <appName>.<version>.tar <version>
compress <appName>.<version>.tar
or
gzip <appName>.<version>.tar

What is tarred is a version directory (named <version>) that includes all the to-be-delivered material into subdirectories of that version directory. For example, for release 1.0.0, you would need to create a 1.0.0 directory structure following the conventions outlined in the section entitled UNIX Directory Structure.

6.1.2.2 Copying the Release package

Next, the release package should be copied to the UNIX application delivery server (tanto) using ssh based file copy clients such as sftp on Solaris or PuTTY psftp (or use FTP). The Release package must be placed in the /apps_source/release/<appName> directory. Note that VPN client must be running in order to connect to the Ministry's UNIX application delivery server -- **tanto.env.gov.bc.ca**.

6.1.2.3 Extracting the Release package to update Source

Once the release package has been copied to the UNIX application delivery server, you will need to uncompress and extract the files from the release package as shown below. Once extracted, individual files can be copied to the source directory. This process of copying files must be clearly outlined in the readme.<version>.txt file.

Connect to tanto.env.gov.bc.ca using secure connection (ssh) as 'application owner' e.g. coors, ems, hdms, swis
cd stage

Run the following Kornshell script to clear out the stage directory
/apps_ux/oraapp/bin/clearStage

files compressed with **compress**
zcat /apps_source/release/<appName>/<appName>.<version>.tar.Z | tar -xvf -
files compressed with ***gzip***
gzcat /apps_source/release/<appName>/<appName>.<version>.tar.gz | tar -xvf -
cd <version>
*cp -r * /apps_source/source/<appName>*

After extraction, build and deployment, clean up stage:

cd stage

Run the following Kornshell script to clear out the stage directory
/apps_ux/oraapp/bin/clearStage

Enter a NOTICE VMAD version status record to let IMB Deliveries know that the version is ready for QA.

6.2 Deploying to DLVR

These directions focus on Solaris deployments, but can be generalized to fit other technology deployments.

1. At the start of your deployment action change to a stage directory on tanto /apps_ux/<appName>/stage/
2. Clear the stage. Use script "clearStage", "rm -r" or a File Manager.
3. Use script "stageApp" to copy the entire code archive to the stage directory.
4. Configure the code. This may involve editing files to include credentials, passwords, partner URLS, host names or environment designators. Note that each edit must be documented in detail in the readme for new applications, changed configurations, or in previous readmes for existing configurations.
5. Backup application configuration files from current application code on the target server using ant task to /apps_ux/<appName>/stage/config
6. Build executable objects, including the new/updated configuration code. For Java apps this is "ant" (restore application configuration files backed up in previous step before building). For Oracle Forms, this is Forms Build. This step is very technology dependant.
7. Deploy the executable objects. For Java apps this is "oc4jdeploy". For Forms and Reports this is "package_FormsAndReports.ksh. For CRON this is simple move of the files out of stage and into /apps_ux/<appName>/admin. This step is very technology dependant.

8. Clear the stage. Use "clearStage", "rm -r" or a File Manager.
9. Enter a DLVR version status record to VMAD to record the deployment to the Delivery environment.

6.3 Database Changes

Database changes include any changes to tables, views, and stored procedures etc that apply directly to the operational or warehouse database. This portion of the delivery may require coordination with ***IMB Deliveries*** if DBA level access to the database is required. Please schedule time for this activity through the ***IMB Business Portfolio Manager*** by indicating if this is to be scheduled as "assisted" or "unassisted"; otherwise database changes to the delivery instance are performed by the ***Vendor Delivery Personnel***.

All supplied scripts and instructions must be run on the Unix server using SQL*Plus and must spool output to a .lst file of the same name. For multiple scripts that need to be executed in sequence then a main.sql script must be provided to execute these scripts in the required sequence, if any failures happen in any of the individual scripts and the following script depends on it, then halt the main script (use **whenever sqlerror** command with appropriate arguments) otherwise continue. Use of tools like TOAD or PL/SQL Developer to run the scripts, or running the scripts from a Windows version of SQL*Plus is not permitted.

Do not include passwords in scripts or README's. Do not ever commit a file containing a password into a code archive.

Provide examples or use the deployment environment's database name in the README.

Note that size of datasets is restricted in DLVR, TEST and TRAIN and that if your capacity plan or data need is greater than 500MB then prior notice and approval is required.

Extended delivery instructions are contained in the Reports Server Delivery Standards document.

6.4 Readme Instructions

Each release for file system code repository applications must include a readme.<version>.<app_name>.txt file that resides in the "docs" sub-directory.

All releases for each application in the subversion code repository share one readme.<app_name>.txt with versions controlled by subversion (Code Repository Tool); a sample readme is included at the end of this document. It must contain complete directions on how to configure and deploy the version of the application being delivered.

NOTE: Readme files no longer are to describe how to deposit (commit to Subversion or update file system based archives). They are to begin with section 1 containing instructions as to how to stage the application from the code archive.

NOTE: Any files from previous deliveries that are to be deleted (i.e. no longer required), must be identified in the readme file. Identify in section #1 any files that were deleted.

NOTE: If application components (eg: Oracle Reports) are not included in a deployment, include nothing in the readme about them.

After deployment to TEST, whenever a readme file is updated or modified (such as correcting typos or issues related with deployment), that fact must be documented in the "Modification History" section of the header in the readme file. This allows ***IMB Deliveries*** to easily see if the readme file has been updated to accommodate required changes.

6.5 Oracle WebForms

The delivery process for Oracle WebForms includes the compiling of all the Oracle WebForms components in the Ministry's environment on the UNIX delivery server -- **tanto.env.gov.bc.ca**. During the delivery, the Oracle Forms modules are first extracted onto the UNIX delivery server, copied to the stage directory and then compiled by the ***Vendor Delivery Personnel*** on the UNIX delivery Server. Additional information specific to the delivery of Oracle WebForms for applications is available in the document Application Delivery Procedures for Oracle WebForms (Intranet only).

6.6 Oracle Reports

Reports that execute on a Ministry Report Server should be first extracted from the code archive to the UNIX delivery server as outlined in section 6.3, then compiled. After compilation, package_FormsAndReports.ksh must be run. At this point, if configuration file changes or additions are required, ***IMB Deliveries*** must be notified and asked to update the command key file, cgicmd.dat in order to deploy and system test the reports.

6.7 Application CASE Dumps

CASE Dump files are not deposited to the code archive.

Deposit CASE dumps to the Ministry by:

1. ***Data Administrator*** approves ***Vendor*** offsite design work in Oracle Designer.
2. ***Vendor*** uses FTP or web submission technology to deposit the .dmp to Ministry infrastructure. Obtain information on this technology from the ***Data Administrator***.
3. ***Vendor*** notifies the ***Data Administrator***.
4. ***Data Administrator*** loads the .DMP into WA_WIP_DEVELOPMENT.
5. ***Data Administrator*** discards .DMP or allows auto-clean to take effect.

6.8 Data Conversion

Any large scale data conversion or loading from other systems is the responsibility of the ***Vendor*** on all the instances. It is also the ***Vendor***'s responsibility to ensure that new statistics are generated after the data load process.

It is strongly recommended that the ***Vendor*** coordinate backup procedures for affected schema and or tables prior to the conversion so that data can be recovered by the ***Vendor*** in a timely manner if the conversion process runs into difficulties.

6.9 Emergency Updates

Emergency updates should only be considered in order to resolve problems that are seriously affecting the operation of an application. If there are emergency updates that need to be delivered, the delivery will need to be coordinated between the ***External Project Manager***, the ***Application Administrator*** and the ***IMB Business Portfolio Manager***. The ***IMB Business Portfolio Manager*** will be responsible for submitting a request to the application delivery calendar email account. Communication at this point is critical to ensure that operational problems are resolved in a timely manner.

An Emergency requiring immediate priority must be characterized by:

- A business unit is barred from doing critical business due to the lack of IT facility. Data is a commodity of the business unit and it is infeasible to postpone delivery of data to customers.

OR

- Deficiencies in the extant PROD configuration items is resulting in corruption of data.

OR

- Deficiencies in the extant PROD configuration items is resulting in the inability of the business unit to record ongoing business data. Data is being lost or it is infeasible to postpone data entry.

6.10 Detailed List of Deposit/Deployment Tasks

- IMB Business Portfolio Manager schedules QA and DLVR deployment by editing Available Booking records in VMAD;
- Deposit cycle: (Trigger = prepared change deliverables)
    - Vendor verifies the deliverables at the ***Vendor***'s site and/or in DLVR;
    - Vendor commits into Subversion or deposits a package and updates the file system code archive;
    - Vendor notifies IMB Business Portfolio Manager and IMB Deliveries that QA may commence by entering a NOTICE record to VMAD;
- QA cycle: (Trigger = Notice from Deposit Cycle)
    - IMB Deliveries performs QA and provides QA approval or feedback and instructions;
    - IMB Deliveries enters QA-PASSED or QA-FAILED records to VMAD;
    - If required, Vendor updates and improve the deliverables and readme and re-deposits;
- DLVR Deployment cycle. (Trigger = Vendor confidence that QA-PASSED will be obtained without deliverable changes, or QA-PASSED)
    - Vendor copies the application source using stageApp to /apps_ux/<appName>/stage.
    - Request that IMB Deliveries run DDL requiring DBA access.
    - Vendor and possibly IMB Deliveries: Configure the application source following the instructions in the readme file.
    - Vendor: Build application deployment objects following the instructions in the readme file; Compile all Oracle Forms, Oracle Reports and Java classes and use provided script package_FormsAndReports.ksh to facilitate automated forms/reports deployment.
    - Vendor and possibly IMB Deliveries: Deploy the application following the instructions in the readme file. Deployment includes running of SQL scripts. Use oc4jdeploy to facilitate java deployment. Deploy CRON tasks.
    - If the DLVR deployment "fails", a Deposit and possibly a QA cycle will be triggered. No worries, this happens on average three times before DLVR is completed. Revise the readme instructions if any errors or discrepancies are encountered, but there is no need for Modification lines at this point.
    - Vendor performs any data conversion processes that are required.
    - Vendor and Application Administrator and IMB Business Portfolio Manager verify the deployment and functionality of the application in DLVR. This is termed "System Testing".
    - Vendor notifies the IMB Business Portfolio Manager and IMB Deliveries that the DLVR deployment has been completed by entering a DLVR version status record to VMAD.
    - Clean up stage.
- TEST Deployment cycle. (Trigger = QA-PASSED, DLVR deployment completed, IMB Business Portfolio Manager approval to proceed)

    - IMB Deliveries copies the application source to /apps_ux/<appName>/stage. (on reamer, not tanto)
    - IMB Deliveries facilitates the running of DDL requiring DBA access.
    - IMB Deliveries configures the application source following the instructions in the readme file for new applications or new configurations. Backup application configuration files from current application code on the target server using ant task to /apps_ux/<appName>/stage/config, including changes as indicated.
    - IMB Deliveries (ant task restores application configuration files backed up in previous step before building) builds application deployment objects following the instructions in the readme file excepting Oracle Forms and Oracle Reports.
    - IMB Deliveries deploys the application following the instructions in the readme file.
    - If the TEST deployment fails, a Deposit, a QA cycle, a DLVR and a TEST cycle will be triggered. A modification line in the readme and explicit re-deployment instructions will be required.
    - Vendor performs any data conversion processes that are required.
    - IMB Deliveries notifies the Application Administrator and IMB Business Portfolio Manager that the TEST deployment has been completed by entering a TEST version status record to VMAD.
    - Clean up stage.
    - Application Administrator and IMB Business Portfolio Manager verify the deployment and complete functionality of the application in TEST including regression testing. This is termed "User Acceptance Testing or UAT".
- PROD Deployment. (Trigger = TEST deployment completed, IMB Business Portfolio Manager approval to proceed)
    - IMB Deliveries creates a Subversion tag for the version label OR IMB Deliveries write protects the release package.
    - IMB Deliveries copies configured application code to /apps_ux/<appName>/stage. (on broach)
    - IMB Deliveries facilitates the running of DDL requiring DBA access.
    - IMB Deliveries configures the application source following the instructions in the readme file for new applications or new configurations. Backup application configuration files from current application code on the target server using ant task to /apps_ux/<appName>/stage/config, including changes as indicated.
    - IMB Deliveries (ant task restores application configuration files backed up in previous step before building) builds application deployment objects following the instructions in the readme file excepting Oracle Forms and Oracle Reports.
    - IMB Deliveries deploys the application following the instructions in the readme file.
    - If the PROD deployment fails a new change cycle is initiated, with a new version label assigned.
    - Vendor performs any data conversion processes that are required.
    - IMB Deliveries notifies the Application Administrator and IMB Business Portfolio Manager that the PROD deployment has been completed by entering a PROD version status record in VMAD.
    - Clean up stage.
    - Application Administrator and IMB Business Portfolio Manager verify the deployment of the application in PROD.

7. POST DELIVERY DEPLOYMENT TASKS

7.1 Application Verification

Application verification includes the high level testing of all application software to ensure completeness of the deliverables. Responsibility for this activity is shared by the ***Vendor External Manager*** and the ***Application Administrator***. The goal of this delivery phase is to ensure that the delivery includes all of the expected functionality.

7.2 Module Verification

The ***Vendor External Manager*** and the ***Application Administrator*** are responsible for verification of each application module. This includes the specific testing of key modules which are critical to the success of the delivery. To facilitate this process, the ***Vendor External Manager*** must furnish the ***Application Administrator*** with a checklist that specifies which application modules require verification -- this checklist should include all modules that have changed or have been added in the current delivery.

As part of the verification process, the ***Application Administrator*** must also be certain that all components of the application are functioning correctly. The application delivery environment is structured to be virtually identical to the production environment; thus any problems or deficiencies in the application identified on the UNIX delivery server should be corrected during the delivery phase -- failure to correct known problems at the delivery phase will almost certainly result in unsuccessful testing during the user acceptance phase and project delays as new releases must be scheduled and have QA performed.

7.3 Notifications

Upon completion of application verification, both the ***Vendor External Manager*** and the ***Application Administrator*** should be satisfied that the application performs as expected. At this point, the ***Application Administrator*** must notify the ***IMB Business Portfolio Manager*** that the application is ready to be migrated to the TEST environment for User Acceptance Testing (UAT). The IMB Business Portfolio Manager will then notify IMB Deliveries by entering a TEST-CONFIRMED version status record into VMAD.

The ***IMB Business Portfolio Manager*** is responsible for confirming the transition of the application from the delivery phase to the test phase, and ultimately to production. In order to expedite the delivery process, the ***IMB Business Portfolio Manager*** will often schedule both the delivery of the application and the deployment of the application to the test environment in advance of the actual application delivery; however, the deployment to test will be delayed or postponed if problems are identified at the application delivery stage.

7.4 Changes

Once a new version of an application has been deployed into production, no changes to the code base will be permitted until a new release number has been designated. If any changes are required to the application after the patch or upgrade has been migrated to production, then an additional patch will be required.

Once a version of an application has been deployed to test changes are allowed but a process must be followed that ensures that the changes are properly deployed to DLVR and to TEST. This requires detailed communication from the Vendor to IMB Deliveries.

A Subversion tag is created immediately before a move to production.

If changes are committed to Subversion between TEST deployment completion and PROD deployment commencement the changes will be not be applied to PROD. These changes are assumed to be part of a new change cycle.

8. DEPLOYMENT DETAILS

8.1 Deployment to the DLVR and TEST Environments

Deployment to the DLVR Environment

- The ***IMB Business Portfolio Manager*** will schedule a DLVR deployment date.
- ***The Vendor*** will commit deliverables into the Subversion code archive or will deposit a package and update the file system based code archive.
- ***The Vendor, and IMB Deliveries*** if required, will perform the DLVR deployment.
- If ***IMB Deliveries*** is required, then QA must have been passed before the DLVR deployment.
- ***The Vendor*** may perform an independent DLVR deployment before QA has passed but will have to redo DLVR deployment if QA fails.
- ***The Vendor*** will notify IMB Deliveries both when the DLVR deployment and the (presumed) final deposit for the release have been completed.
- VMAD will notify the ***IMB Business Portfolio Manager*** and the application administrator that the DLVR deployment is complete;
- ***Vendor*** will perform data conversion tasks as required.

Deployment to the TEST Environment

- The ***IMB Business Portfolio Manager*** will schedule a TEST deployment date.
- ***IMB Deliveries*** will perform the TEST deployment.
- VMAD will notify the Business Portfolio Manager and the Application Administrator that the TEST deployment is complete.
- ***Vendor*** will perform data conversion tasks as required.

NOTE: In the event that ***IMB Deliveries*** determines when deploying to TEST that a script is non-functional the ***IMB Business Portfolio Manager***, the ***Application Administrator***, and the ***External Project Manager*** will be notified and the deployment will be cancelled. Once a deployment is cancelled, a new deployment time must be scheduled through the ***IMB Business Portfolio Manager***.

8.2 When changes are required in TEST after TEST deployment

Changes to a release that has been approved for deployment to TEST are only allowed when the release fails TEST deployment or the release has failed UAT and cannot be moved to PROD. Changes for new business functionality are not allowed.

When changes are required to a release that has already gone to TEST, but has not gone to PROD, there are two choices for the IMB Business Portfolio Manager.

1. Fix the issues in a new patch. Normally such a patch would be bundled with the previous, erroneous release. Full scheduling and process is required. This alternative is seldom the most cost-effective.
2. Redo the release undergoing testing.

Workflow for making changes to TEST after TEST deployment is:

- IMB Deliveries, IMB Business Portfolio Manager or Acceptance Testers identify a problem.
- Vendor diagnoses and builds a fix for the problem.
- Vendor modifies the readme.
    - Insure a modification line (WHEN WHO WHAT) is included.
    - Clearly and in detail identify steps in the readme that need to be rerun and/or avoided on re-deployment to DLVR and to TEST. This may be done in a separate email or in the readme.
- Vendor commits changes to Subversion or updates the package and the file system based code archive.

- IMB Deliveries software agents notify IMB Deliveries through VMAD that a re-deposit has occurred.
- Vendor notifies IMB Deliveries that the final change has been committed by entering a NOTICE version status record in VMAD.
- No formal booking is required for re-QA, DLVR or TEST. IMB Business Portfolio Manager approval and communication and approval for re-deployment is required.
- IMB Deliveries performs a QA on the revised package.
- Vendor performs steps to re-deploy into DLVR.
- IMB Deliveries performs steps to re-deploy into DLVR, if required.
- If required, Vendor performs steps to re-deploy into TEST.
- IMB Deliveries performs steps to re-deploy into TEST.

8.3 Deployment to the Production Environment

- The ***IMB Business Portfolio Manager*** will schedule a deployment date.
- The ***Application Administrator*** will notify the ***IMB Business Portfolio Manager*** that the deployment to production is to proceed as scheduled **at least 1 working day** prior to the scheduled deployment date.
- The ***Application Administrator*** will notify all users and other affected parties that the application will be unavailable during the period of the scheduled upgrade.
- The ***IMB Business Portfolio Manager*** will confirm the deployment.
- ***IMB Deliveries*** will perform the application deployment.
- The ***Vendor*** will perform any data conversion as required.
- ***IMB Deliveries*** will notify the ***IMB Business Portfolio Manager*** and the ***Application Administrator*** when the deployment is complete.
- The ***Application Administrator*** will verify the functionality of the application in production.
- The ***IMB Business Portfolio Manager*** will notify all parties that the rollout has been completed.
- The ***Application Administrator*** will notify users and other affected parties that the deployment to production is complete (i.e. a new version of the application is now in production).

8.4 Optional Training Environment

A separate schedule request is necessary for deployments to the Train environment.

- The Production deployment must be completed prior to the Training environment scheduling.
- The ***IMB Business Portfolio Manager*** will schedule a deployment date.
- The ***Application Administrator*** will notify the ***IMB Business Portfolio Manager*** that the deployment to training is to proceed as scheduled **at least 1 working day** prior to the scheduled deployment date.
- The ***Application Administrator*** will notify all users and other affected parties that the application will be unavailable during the period of the scheduled upgrade.
- The ***IMB Business Portfolio Manager*** will confirm the deployment.
- ***IMB Deliveries*** will perform the application deployment.
- ***IMB Deliveries*** will notify the ***IMB Business Portfolio Manager*** and the ***Application Administrator*** when the deployment is complete.
- The ***IMB Business Portfolio Manager*** will notify all parties that the rollout has been completed.
- The ***Vendor Delivery Personnel*** will perform any data conversion as required.
- The ***Application Administrator*** will verify the functionality of the application in training.
- The ***Application Administrator*** will notify users and other affected parties that the deployment to training. is complete (i.e. a new version of the application is now available for training).

9. DATA REFRESHES

At the request of the ***Application Administrator***, a copy of the current production data may be loaded to the TRAIN, TEST or DLVR database instance. The ***IMB Business Portfolio Manager*** must coordinate this request with IMB Deliveries and Database and Middleware Services.

NOTE Requests for a refresh of data into the training, test or delivery database should be made no more than **once** yearly. Each refresh may take from 1/2 day to 2 days to complete.

IMPORTANT With the increasing complexity of applications, it is now expected that the ***Vendor*** will supply detailed instructions on the process if anything more than a simple export/import of a single application is required.

10. SECURITY

Files or scripts bundled as part of a delivery must never include a valid username or password. When scripts require a username or password these should be prompted for during the execution of the script. Note that scripts run under the UNIX operating system must never be invoked with the password specified as a command-line parameter. Command-line parameters under UNIX may be viewed by anyone on the system using the UNIX ps command.

If there are UNIX shell scripts or files that need to have username and password information included in them in order for the software to run, you must make sure that these files do not have read privileges for group or world (other). This can be accomplished by changing the files privileges using the UNIX 'chmod' command.

11. CRON JOBS

11.1 CRON Basics
A CRON job must be wrapped in Korn shell (.ksh) Solaris compatible script.

CRON jobs are run by the <application> account in DELIVERY or Application Administrator accounts on shave, and by the Application Delivery account (oraapp) in TEST, TRAIN and PROD.

11.2 CRON Naming
CRON job scripts should be named <appName>_<cronjob>.ksh

11.3 CRON Output
It is a UNIX best practice for CRON jobs that they only output to 'stdout' or 'stderr' when some sort of error or exception occurs. Otherwise, and normally, a cron script outputs absolutely nothing to 'stdout' or 'stderr'. The result is that the cron owner only gets an email when there is an exception and/or a problem.
If a cron job outputs anything in a normal run, emails accumulate uselessly. This type of "noise" in mail bars detection of real problems.

Simple redirection of output to /dev/null is not acceptable.

A utility script called /apps_ux/oraapp/bin/mailtee.ksh is to be used for all CRON jobs as part of the CRON definition line.

An example:
* * * * * /apps_ux/<appName>/admin/<appName>_.ksh 2>&1 | /apps_ux/oraapp/bin/mailtee.ksh /apps_ux/<appName>/logs/<appName>_.ksh.log app.manager@gov.bc.ca <appName>_<cronjob>.ksh false

The "false" means "do not append to log, overwrite it". "true" means "append to log".

Email addresses can be multiple. Put a comma and no spaces between addresses. E.g. "addr1,addr2"

Application Administrators and IMB Business Portfolio Managers may request that IMB Deliveries modify email lists in TEST and PROD.

File output from this script must go to /apps_ux/<appName>/logs/<appName>_<cronjob>.ksh.log.

11.4 CRON Logging

If a Java job is contained in the cron, following the standard in Java Application Delivery Standards, IMB Deliveries needs a simple way to change to ERROR level (log4j.properties). Include instructions to do this into the readme (particularly for TEST/PROD).

Log output from <appName>_<cronjob>.ksh should go to /apps_ux/<appName>/log/<appName>_<cronjob>.log.

Alternatively, log output from <appName>_<cronjob>.ksh could go to /apps_ux/logs/<appName>_<cronjob>.log.

Initial level for logging must be delivered as ERROR.

11.5 CRON Monitoring

CRON job generated emails and log output are expected to be monitored by the ***Vendor*** in DLVR. CRON job generated emails and log output are monitored by the Application Administrator and/or by IMB Deliveries in TEST and PROD.

IMB Deliveries monitors daily the resulting emails from TEST and PROD. IMB Business Portfolio Managers are notified of any output, at least once. IMB Deliveries stops monitoring in TEST after a release has been deployed to PROD. "Noisy" log monitoring is terminated after IMB Business Portfolio Manager notification.

11.6 CRON Code Management

The proper directory to deposit CRON jobs to is: source/<appName>/admin. Also acceptable is: source/<appName>/bin.

The proper directory to deploy CRON jobs to is /apps_ux/<appName>/admin. Also acceptable is: /apps_ux/<appName>/admin

Passwords must not be put into CRON scripts. Passwords should be obtained via a reference a file that contains the password. This file needs to have permissions rw-------.

11.7 CRON Testing

No CRON job will be deployed to PROD that has not been deployed to TEST and UAT tested by the Application Administrator and by IMB Deliveries.
Readme's should specify that cronjobs in TEST should be disabled after 1 period if the job would damage TEST in any way if continued, or if it is simply pointless to continue because UAT has ended.

If it is not technically possible to run the job in TEST or Delivery, due to lack of supporting infrastructure or supporting partner applications or supporting files or data, then IMB Deliveries cannot insist on testing the CRON job. In such a case the first period in PROD must serve as a UAT for the job. If this is the case then this MUST be noted in the README.

From IMB Deliveries point of view critical UAT QA points are:
- Does it run without producing error messages to stdout, stderr or the log(s)?
- Does it produce exception output only to stdout and stderr?
- Does it only produce ERROR level output to java logs?

11.8 CRON Examples

```
# CRON on tanto for GEN
5 1 * * * /apps_ux/<appName>/admin/<appName>_<cronjob>.ksh  2>&1 |
/apps_ux/oraapp/bin/mailtee.ksh /apps_ux/<appName>/logs/<appName>_<cronjob>.ksh.log
app.administrator@gov.bc.ca <appName>_.ksh false
# FALSE = do not append
#
# SAMPLE crontab when logged on TANTO as GEN
27 2 * * * /apps_ux/gwl/bin/gen_survey.ksh  2>&1 | /apps_ux/oraapp/bin/mailtee.ksh
/apps_ux/logs/gwl/gen_survey.ksh.log application.administrator@gov.bc.ca gen_survey.ksh false
#EOF#
```

12. CONCLUSION

This document provides a set of standards and guidelines for the deposit and deployment of applications to the Ministry.

The goal of documenting deposit and deployment standards is to facilitate a smooth and orderly delivery process. By following these standards and guidelines, better organization, communication and streamlined software installations will be achieved.

A key success factor in the application delivery process is to ensure that good lines of communication are maintained during the delivery and subsequent deployments to test and production. It is recommended that a meeting be held before each major delivery to ensure that all of the participants in the delivery are familiar with their expected roles and responsibilities.

APPENDIX A - SAMPLE Java README File

NOTE: configuration steps (step 3 in this readme file) must be handled by Ant backup and restore tasks as in build.xml sample of Appendix B. Manual Configuration steps are required just for the new application in the readme file.

NOTE: If a deposit/deployment does not contain certain components then those sections should be deleted.

NOTE: The following is a sample of a Java application delivery. For a sample of a Forms/reports delivery readme, see webforms_application_delivery_standards.html

```
************************************************************
* General Error and Information Collection System
* (GENERIC)
*
* Release: 2.1.0
* Date: September 17, 2010
* Author: John Doe, Best Vendors Inc.
*
* Modification History
```

*
****

This file describes steps to install the GENERIC v.2.1.0 web application. v2.1.0 includes changes to the GEN schema and to the GEN Java application.

OVERVIEW

This document describes the installation process for a patch version of the GENERIC web application.

REQUIREMENTS

Prior to installing the GENERIC application, the following requirements must be verified:

- This installation assumes installation of version 2.0.9.

BUG FIXES

Not applicable.

MODIFIED COMPONENTS

Not applicable.

OBSOLETE COMPONENTS

Not applicable.

DATABASES
---
envdlvr1, envtest1, envprod1

SERVER LOCATIONS
---
tanto, reamer, broach

DEPLOYMENT INSTRUCTIONS
---

1. PREPARING FOR THE INSTALL

1.0. Log on to the UNIX Delivery Server using the GEN account.

1.1. Export from Subversion
cd /apps_ux/gen/stage
clearStage
svn export HTTPS://axe.env.gov.bc.ca:8443/svn/gen/trunk
(Note that the Subversion service URL may change.)

2. MODIFY THE GEN DATABASE SCHEMA OBJECTS

2.1. Change to the Scripts Directory
cd scripts

2.2. Connect to the target database as **SYSTEM** using sqlplus. This step is done by IMB Database Administrators. Databases involved are envdlvr1, envtest1 and envprod1.
. setdb ora10g (or .setdb oasds if compiling Oracle Forms or Reports)
sqlplus system/<password>@<instance>

2.3. Create the tablespaces.
start gen.2.1.0.tbs

NOTE: You will be prompted for <table datafile name> and <index datafile name>. Respond with the full path, without the trailing slash, to the directory that will store the GEN_TABLES and GEN_INDEXES tablespaces. For example:
/fs/u02/oracle_data/envdlvr1

2.4. Create the GENERIC owner account(GEN) and application PROXY account (PROXY_GEN) and assign appropriate privileges.
start gen.2.1.0.usr

NOTE: You will be prompted to provide passwords for GEN and PROXY_GEN.

commit;
exit;

2.5. Connect to the target database as GEN . These steps are done by the ***Vendor***.
connect gen/<password>@<instance>

2.6. Create roles used by GEN and assign to PROXY_GEN_REPORTS.
start gen.2.1.0.rle

2.7. Create tables, views, releases etc.
start gen.2.1.0.sql

2.9. Commit and exit sqlplus
commit;
exit

2.10. Review spooled output for errors.
Notify Vendor and IMB Business Portfolio Manager if any errors have occurred and halt the deployment.

more gen.2.1.0.lst

3. CONFIGURE THE GEN WEB APPLICATION

NOTE TO VENDORS The following step(s) will be application and technology specific. Please ensure that ANY files that need to be modified are listed as well as the parameters in each environment that must be updated in each file. For example an included instruction step might look like this:

3.1 Edit the URL for reporting into applicationResources.properties
Edit web-src/web/WEB-INF/classes/config/applicationresources.properties.
Replace token report.server.url=<reports_url> with the appropriate reports service URL.
DLVR: report.server.url=HTTPS://delivery.a100.gov.bc.ca/reportsint/rwservlet
TEST: report.server.url=HTTPS://test.a100.gov.bc.ca/reportsint/rwservlet
PROD: report.server.url=HTTPS://a100.gov.bc.ca/reportsint/rwservlet
</reports_url>

4. DEPLOY THE GEN WEB APPLICATION

NOTE TO VENDORS The following step(s) will be application and technology specific. The example is for a OC4J Java application.

4.1 Compile
cd web-src
ant

4.2 Deploy EAR
cd web-src/deployment
cp -p gen.ear /apps_ux/gen/deployment/gen.ear
cp -p gen.ear /apps_ux/gen/deployment/gen.ear.2.1.0

4.3 oc4jdeploy
oc4jdeploy
Redeploy application
home e to h
/apps_ux/gen/deployment
Submit

5. TESTING

Test the application using one of the following URLS:
HTTPS://delivery.a100.gov.bc.ca/int/gen
HTTPS://test.a100.gov.bc.ca/int/gen
HTTPS://a100.gov.bc.ca/int/gen

6. NOTIFICATION

Notify IMB Deliveries that the deployment is complete.

APPENDIX B - SAMPLE Ant build.xml File

```
<!--

  ANT build file to compile and deploy VMAD web application

-->

  <project name="vmad" default="ear">

  <!-- name of app -->

  <property name="name" value="vmad" />

  <!-- path to working directory -->

  <property name="current_dir" value="." />
```

```
<!-- path to common libraries -->
<property name="common_lib" value="/sw_ux/lib/" />
<!-- path to temp. config files-->
<property name="config_temp" value="${current_dir}/config_temp" />
<!-- path to config files source within target server-->
<property name="config_src" value="/apps_ux/applications/vmad/vmad-web/WEB-INF" />
<!-- path to config files destination on the target source code-->
<property name="config_dest" value="${current_dir}/web/WEB-INF" />
<!-- source -->
<property name="source_web" value="${current_dir}/web" />
<property name="source_webinf" value="${source_web}/WEB-INF" />
<property name="source_metainf" value="${current_dir}/META-INF" />
<property name="source_src" value="${current_dir}/src" />
<property name="source_classes" value="${source_webinf}/classes" />
<property name="source_lib" value="${source_webinf}/lib" />
<!-- path to ear deployment directory -->
<property name="ear_dir" value="${current_dir}/deployment" />
<!-- path to build directory -->
<property name="build_dir" value="${current_dir}/build" />
<!-- build -->
<property name="build_vmadweb" value="${build_dir}/${name}-web" />
<property name="build_webinf" value="${build_vmadweb}/WEB-INF" />
<property name="build_metainf" value="${build_dir}/META-INF" />
<property name="build_classes" value="${build_webinf}/classes" />
<property name="build_lib" value="${build_webinf}/lib" />
<!--  classpath to java libraries -->
```

```
  <path id="classpath">
    <fileset dir="${source_lib}">
      <include name="mjf-utils-01_00_00.jar" />
      <include name="struts.jar" />
    </fileset>
    <fileset dir="${common_lib}">
      <include name="jakarta.apache.org/commons/logging/1.0.4/commons-logging.jar" />
      <include name="jakarta.apache.org/log4j/1.2.8/log4j.jar" />
      <include name="www.oracle.com/9.2.0.3/jdk.1.3/classes12.jar" />
      <include name="www.oracle.com/oc4j/9.0.3/oc4j.jar" />
      <include name="jakarta.apache.org/taglibs/standard/1.0.3/jdbc2_0-stdext.jar" />
      <include name="java.sun.com/servlet/2.3/servlet.jar" />
    </fileset>
    <fileset dir="${common_lib}/gov.bc.ca/webade/4.2" includes="*.jar"/>
    <fileset dir="${common_lib}/gov.bc.ca/encc"       includes="*.jar"/>
  </path>
<!--  delete all created files and directories -->
  <target name="clean">
    <echo message="Deleting build/deploy files and directories..." />
      <delete dir="${ear_dir}" />
      <delete dir="${build_dir}" />
  </target>
<!--  backup configuration files -->
  <target name="backup">
    <echo message="Backup configuration files..." />
      <mkdir dir="${config_temp}" />
```

```
    <copy todir="${config_temp}"><fileset dir="${config_src}" includes="**/*.xml"/></copy>
</target>
<!--  restore configuration files -->
<target name="restore" depends="backup">
  <echo message="Restore configuration files..." />
    <copy todir="${config_dest}"><fileset dir="${config_temp}" includes="**/*.xml"/></copy>
    <echo message="Delete config_temp directory..." />
    <delete dir="${config_temp}" />
</target>
<target name="prepare" depends="clean">
  <echo message="Creating build/deploy files and directories..." />
  <mkdir dir="${build_dir}" />
  <mkdir dir="${build_vmadweb}" />
  <mkdir dir="${build_webinf}" />
  <mkdir dir="${build_metainf}" />
  <mkdir dir="${build_classes}" />
  <mkdir dir="${build_lib}" />
  <mkdir dir="${ear_dir}" />
  <echo message="Copying source objects into build" />
  <copy todir="${build_metainf}"><fileset dir="${source_metainf}"/></copy>
  <copy todir="${build_vmadweb}"><fileset dir="${source_web}"/></copy>
</target>
<!--  compile java class files -->
<target name="compile" depends="restore, prepare">
  <echo message="Compiling ${name} classes..." />
  <javac srcdir="${source_src}" destdir="${build_classes}" deprecation="on" debug="on" >
```

```
    <classpath refid="classpath" />

   </javac>

  </target>

  <!--  build war -->

  <target name="war" depends="compile">

   <echo message="Building WAR file..." />

   <jar jarfile="${build_dir}/${name}-web.war" update="false">

    <fileset dir="${build_vmadweb}"/>

   </jar>

  </target>

  <!--  build ear -->

  <target name="ear" depends="war">

   <echo message="Building EAR file..." />

   <ear earfile="${ear_dir}/${name}.ear" appxml="${build_metainf}/application.xml"
update="false">

    <fileset dir="${build_dir}"

     includes="${name}-web.war,META-INF/orion-application.xml" />

   </ear>

  </target>

  </project>
```

APPENDIX C - SAMPLE main.sql File

```
************************************************************
* General Error and Information Collection System
* (GENERIC)
*
* Date: Feb. 6, 2012
* Author: Karim Mustafa, IMB
*
* Modification History
*
```

```
************************************************************
/*
        Use whenever sqlerror command with appropriate arguments based on developer judgment
*/

spo main.lst

WHENEVER SQLERROR CONTINUE

@script1.sql

WHENEVER SQLERROR EXIT SQL.SQLCODE

@script2.sql

spo off

-------------------------------------------------------------------------------------------------------------------
-
```

Optional list of included files:

```
2.1.0/web-src/web/WEB-INF/struts-config.xml
2.1.0/web-src/web/WEB-INF/web.xml
2.1.0/web-src/web/WEB-INF/tld/struts-bean.tld
2.1.0/web-src/web/WEB-INF/tld/struts-html.tld
2.1.0/web-src/web/WEB-INF/tld/struts-logic.tld
```