Name of Standard: **NRS Standards for Auto Delivery Using Jenkins**

Version: **1.0.2**

Standard Custodian: **Clecio Varjao**

Submitter Name: **Clecio Varjao**

Stakeholder(s) Impacted by Change: **App Deliveries, BPM, Vendor**

Type of Change: **LOW**

Change(s) to Standard:

- **Section 6.2.1 - Added Emergency Release Label**

Standard

In an effort to increase the speed and capacity of the Application Delivery team, we are now making use of Jenkins to automatically deliver (build, configure and deploy) applications. The standards set forth in this document are specific to this type of delivery and are not required in any other case. Over time however, this will become the standard way to deliver for the vast majority of applications.

Version Control

Date	**Author**	**Version**
Feb 22, 2017	Clecio Varjao	1.0.2

1. Introduction – Increase the speed and capacity of the Application Delivery team through automated delivery process.

2. List of any SDLC Deliverables this standard directly relates to:
    - o Application Delivery Checklist,
    - o Migration Package (Source Code)

3. Standard

    (Attached)

Supporting Documentation NONE

Corporate Services for the Natural Resource Sector

Information Management Branch

Standards for Auto Delivery Using Jenkins

Last Updated:	February 22, 2017
Version:	1.0.2
Document:	Standards_for_Auto_Delivery_Using_Jenkins_1.0.2.docx

Table of Contents

1. Version Control

Document Version	Revision	Date	Author(s)	Change Reference
1.0.0	Final	October 2013	Deliveries (Dylan Dawson)	Initial release
1.0.1	Final	January 2014	Deliveries (Dylan Dawson)	Added omission from section 6.8.2 and Appendix A – Example readme.txt
1.0.2	Final	February 2017	Clecio Varjao	Adding Emergency release label section (6.2.1)

2 Document Conventions

The following conventions are used in this document:

	Description
CAPS	Used when capital characters are to be entered.
Bold	Indicate a proper name or system command to be performed.
Bold Italics	Indicates a role or function to be performed.
<appName>	Represents the short name or acronym that has been assigned to the application.
<version>	Represents a versioning format #.#.#.# where the first # represents a major version release, the second # represents a minor version release, the third # represents a patch version release and the final # represents the release iteration number (RIN).
`Courier`	Code listings.
UTF-8	Refers to UTF-8 without BOM (Byte Order Mark)
#{repo}	Used to indicate the root of the SVN repository.
#{trunk}	Used to indicate the /trunk folder in Subversion. For example, for the application <appName>, trunk means: https://a100.gov.bc.ca/svn/<appName>/trunk
#{source}	Used to indicate the location of application source code in SVN.

3 Introduction

3.1 *Purpose*

In an effort to increase the speed and capacity of the Application Delivery team, we are now making use of Jenkins to automatically deliver (build, configure and deploy) applications. The standards set forth in this document are specific to this type of delivery and are not required in any other case. Over time however, this will become the standard way to deliver for the vast majority of applications.

3.2 *Audience*

The audience for this document is primarily vendors on contract with the Natural Resource Sector (NRS), who are developing or maintaining applications for the business clients and BPM's that are guiding development.

3.3 *Scope/Exclusions*

This document is intended to be used as a supplement to the NRS Application Delivery Standards; it adds specific standards for applications that will use Auto Delivery (Jenkins).

3.4 *Assumptions*

It is assumed that developers understand application development and the NRS Systems Development Lifecycle (SDLC).

3.5 *Contacts*

All inquiries regarding these standards should be directed to the Business Portfolio Manager assigned to the project.

4 Overview

Automating the compilation, configuration and deployment of applications has numerous benefits to everyone involved. For vendors, Jenkins facilitates Continuous Integration; developers check modified code into Subversion and use Jenkins to deliver the application. Once this iterative process is complete, an Application Delivery Specialist can apply it to subsequent environments (test, production) at the push of a button or on a schedule. This allows vendors to release much more quickly, streamlining the delivery process and relieving the Application Delivery Specialist from repetitive work that is, in this framework, completely automated.

4.1 *Converting to Auto Delivery*

Modifying an application to use Jenkins is relatively simple. The following tasks are necessary:

1. Ensure that all source code is in Subversion.
2. Clean up and remove unnecessary files.
3. Identify all configuration properties/parameters/userids, etc… that will change per environment (delivery/test/prod). Provide these to the Application Delivery team.

4. Create configuration placeholders that Jenkins will find/replace with the actual value per environment.
5. Migrate library dependencies from embedded to Apache Ivy. Application Deliveries may assist in the creation of your ivy.xml file.
6. Convert the readme.txt into an Application Blueprint – an xml file that Jenkins uses to compile, configure and deploy the application. This step will be completed by and Application Delivery Specialist.

4.2 *Application Blueprint*

Auto Delivery using Jenkins requires that many parts of the readme.txt file that normally accompanies a release be rewritten in a machine-readable format (.xml). This file only needs to be created once but must be maintained by the vendor during subsequent releases to ensure that all configuration parameters are present. Currently, IMB Deliveries creates the first version of Application Blueprint.

4.3 *readme.txt*

Since much of the configuration and deployment information for an application is captured in the Application Blueprint, the readme.txt file will be much simpler. Refer to Appendix A – Example readme.txt.

5 Prerequisites

Vendors will need access to Subversion to deposit source code changes, and Jenkins to deliver their application. Contact Application Delivery to set this up. If it is your first time, please allow four working days for access to Jenkins, as your IDIR ID needs to be added to a security group.

6 Standards

6.1 *Source Code Repository*

Apache Subversion (SVN) is used as the standard source code repository and revision control. All text files submitted must be encoded using ANSI or UTF-8 (without BOM). Unless otherwise required by a specific technology, text files must also use Unix style End-Of-Line (EOL) markers (LF).

The URL for your source code will be: https://a100.gov.bc.ca/pub/svn/<appName> where <appName> is the application acronym that is registered in IRS.

6.1.1 Folder Structure

The CSNR Subversion repository is organized, for each application, as follows:

Directory	Description
#{repo}/trunk	Master media image of DELIVERY level source code. This directory must

	contain only configuration items (source code, readme, documentation, etc.)
#{repo}/tags	Subversion tags will be created only by the IMB Deliveries. Tags are created as part of the change control process each time a QA is requested. Refer to section 6.2 for more information on release tagging.
#{repo}/branches	Optional development line that does not disrupt the trunk.
#{repo}/data-fix	Database DML scripts (refer to data fix defined in section 6.8.4)

Under #{trunk}, you must use the following folder structure:

#{trunk}/docs	readme.txt
#{trunk}/documentation	Application documentation (Diagrams, etc.)
#{trunk}/database	Database object source code. See section 6.8 below.
#{trunk}/source	Application source code.
#{trunk}/scripts	Version-specific (one-time) SQL Scripts

6.1.2 File / Directory Naming

File and directory names must be Unix-friendly and are assumed to be case-sensitive. Allowed characters:

- Lowercase alphanumeric (a-z0-9)
- Underscore (_)
- Full stop (.)
- Hyphen-minus (-)
- Tilde (~)

All other characters must not be used, including, but not limited to:

- Spaces
- Special characters

6.1.3 readme.txt

The application blueprint is intended to largely replace the readme.txt file. At this time however, Jenkins is not configured to automate all aspects of every delivery, such as database work.

Please note that the readme.txt:

- Must be located at #{trunk}/docs/readme.txt
- Must be plain text
- Must be ASCII or UTF-8 encoded.

Refer to Appendix A for a template readme.txt file for Auto Delivery.

6.1.4 Licensing

The vendor is responsible for including the license for any external work (e.g., library, component, font, resource) in their delivery.

6.2 *Release Labels (Version Numbering)*

Release labels are of the format #.#.#.# where the first '#' represents a major release, the second '#' represents a minor release, the third '#' represents a patch release, and the fourth '#' represents the Release Iteration Number (RIN).

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.

The major, minor and patch release numbers together are considered the official release version.

The RIN is added by Application Deliveries each time they QA a release candidate. The RIN starts at 0 and increments by 1 each time a QA is requested. For example:

4.2.0.0	Version: 4.2.0 RIN: 0 QA: 1
4.2.0.1	Version: 4.2.0 RIN: 1 QA: 2
4.2.0.2	Version: 4.2.0 RIN: 2 QA: 3
4.2.0.3	Version: 4.2.0 RIN: 3 QA: 4
4.2.1.0	Version: 4.2.1 RIN: 0 QA: 1

Each release will be tagged before QA and made available in Subversion in the tags directory of each repository as defined in section 6.1.1.

6.2.1 Emergency Release Label

In the event that a release needs to be quickly fast-tracked all the way to production, the label will have the emergency iteration number (EIN) qualifier (-#) appended to the release version (*Major*.*Minor*.*Patch*-*EIN*) and the release label (*Major*.*Minor*.*Patch*-*EIN*.*RIN*). Where the EIN is a 1-based sequential number incremented by 1, prefixed by a dash, and the first 3 #s are the same as the current version in production.

Examples:

4.2.0-1.0	Version: 4.2.0 EIN: 1 RIN: 0 QA: 1
4.2.0-1.1	Version: 4.2.0 EIN: 1 RIN: 1 QA: 2
4.2.1-1.0	Version: 4.2.1 EIN: 1 RIN: 0 QA: 1
4.3.0-1.0	Version: 4.3.0 EIN: 1 RIN: 0 QA: 1

Emergency released are expected to have a turn-around measured in hours, no more than a couple of days.

6.3 *Dependency Management*

An application may have many library dependencies (e.g.: jar, dll), and all dependencies must be managed by submitting an Apache IVY file (ivy.xml) at the root of each module or project.

6.4 *Modules Source Code*

If an application uses more than one technology, the modules must be organized within #{trunk}/source as follows:

#{source}/ear	Java EE Project(s)
#{source}/java	Generic Java Projects (non Java EE)
#{source}/dotnet	Dot Net Project(s)
#{source}/scripts	Shell scripts (batch, ksh, sh, bash)
#{source}/bin	Binary/Executables tools. must have approval
#{source}/oracle-forms	Oracle WebForms
#{source}/oracle-reports	Oracle Reports
#{source}/tasks	Only file allowed in the source folder

6.4.1 Java

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 CSNR's Java Application Development Standards

In Subversion, the Java source code folder layout must be based on Maven:

- src/main/java
- src/main/resources
- src/main/webapp
- src/main/application

6.4.1.1 Logging

Apache LOG4J must be used whenever logging is required. SLF4J may also be used in conjunction with LOG4J.

In addition, the default LOG4J initialization mechanism and configuration file must be used. The configuration file will be automatically generated upon deployment; however, exemptions may be granted by Application Deliveries.

6.4.1.2 Java EE

Java EE application must be stored at #{source}/ear and must follow the following directory structure:

#{source}/ear/application	EAR file root configuration files, such as: - META-INF/application.xml - META-INF/weblogic-application.xml - *-jdbc.xml (see JDBC Modules)
#{source}/ear/<name>-web	Java EE Web Module
#{source}/ear/<name>-ejb	Java EE EJB Module
#{source}/ear/<name>-jar	Java Library (jar)

6.4.1.3 Web Application (WAR)

WEB-INF/classes and WEB-INF/lib must not be included, and svn:ignore should be used to avoid accidental commit of those folders.

6.4.1.4 Standalone Applications (JAR)

Java standalone applications packaged as jar files are considered to be Java EE module type called "jar", except that it will not be wrapped in an EAR file. The resulting jar can be manually run, or scheduled task.

6.4.1.5 JDBC

WebADE should be used for obtaining database connection. However, if WebADE is not used, and the application needs a JDBC connection:

- JDBC URL[1] must be externalized to a property file;
- JDBC URL must use TNSNAME identifier;
- Username and password must not be part of the URL, and must be set separately;

6.5 *Microsoft .NET*

- All projects and solutions must be stored at #{source}/dotnet
- Visual Studio must not be a requirement
- Must be compiled and packaged via command line tools: MSBuild, Aspnet_compiler
- Must not include `bin` folder(s), and svn:ignore must be used
- Must not include `obj` folder(s), and svn:ignore must be used
- All libraries third-party libraries must be retrieved via Ivy which will copy to `lib` folder folder. (HINT: use hintpath attribute for references in the project file)
- Must not include user-specific configuration files (e.g.: *.suo, *.scc, *.vbproj.user, *.csproj.user)

[1] http://docs.oracle.com/cd/B14117_01/java.101/b10979/urls.htm#BEIJFHHB

6.6 *Oracle Reports*

Reports and Libraries must be prefixed by the application acronym.
Oracle Report configuration (cgicmd.dat) must be included with the source code and:

- Must be named according to the deployment context as follow:
  cgicmd-int.dat
  cgicmd-ext.dat
  cgicmd-pub.dat
- Must use Unix-style EOL (LF)
- Must be ASCII encoded
- Database password, and instance must use the respective placeholder:`#{password}`, `#{database}`;

Do not include the compiled files (*.rep, *.plx), and svn:ignore must be used to avoid accidental commits.

6.7 *Oracle Forms*

Forms, Menus, and Libraries must be prefixed by the application acronym.
Oracle WebForms configuration must be included and:

- Must be named “webforms.cfg”
- Must use Unix-style EOL (LF)
- Must be ASCII encoded
- `jacob.jar` must come first in `webUtilArchive` and `archive` entries

If custom icons are used, they must be in a “webicons” subfolder.

Do not include the compiled files (*.fmx, *.plx, *.mmx), and svn:ignore must be used to avoid accidental commits

6.8 *Database*

6.8.1 Database Object State Scripts

All database objects “owned” by the application must be submitted and maintained as DDL scripts and:

- Must be stored in “#{trunk}/database” (also referred as #{database})
- Each object must have its own DDL file
- Must be grouped (in subfolders) by type using the following template:
  “#{trunk}/database/#{object_type}/#{object_name}.sql”
- If the application uses multiple schema and/or database, the files must be grouped in a subfolder (logical identifier), such as:
  “#{trunk}/database/#{logical_identifier}/#{object_type}/#{object_name}.sql”
  Where, #{logical_identifier} is any name used to group the database/schema combination.

- The DDL scripts must be re-executable whenever possible.
  - For Oracle databases “CREATE OR REPLACE” statement must be used (when available)
  - For MSSQL databases, it is achievable by a DROP followed by a CREATE.

6.8.2 Release-Specific Scripts

Release-specific scripts are all SQL scripts that are not stored as defined in section 6.8.1. Those scripts are one-time script (not re-executable), and directly related to a specific release, including the release main script and references to re-executable scripts.

All release-specific scripts must be stored at #{trunk}/scripts/#{release}/#{iteration} with the following folder structure:

Folder	Description
docs	Readme file for DBA
dml	Data manipulation scripts.
ddl	Data definition scripts for when it is not possible to create re-executable scripts as defined in section 6.8.1.
database	Convenience link (relative svn:externals) to #{trunk}/database so that scripts can refer to the database folder as a subfolder.

Where,

#{release} is the release label defined in section 6.2.
#{iteration} is a 2-digit, 0-padded, 0-based sequential number for each iteration. The iteration is increased whenever a DBA executes those scripts. Please check prior to submitting scripts if a new iteration is required or if the current iteration can be edited

Every effort must be made to create re-executable scripts as defined in section 6.8.1. If it is not possible (such as DML scripts) however, those scripts must be submitted as one-time scripts.

6.8.3 WebADE

WebADE DML scripts may be deposited as re-executable scripts as defined in section 6.8.1 at #{database}/webade. Otherwise, it must be deposited as release-specific scripts as defined in section 6.8.2 at #{scripts}/webade.

6.8.4 Data Fix

Data fixes are DML scripts required to patch data in an existing database. DDL (structural changes) are not allowed in this category, and if required, must be delivered as a new application version/release.

These scripts are place into a “data-fix” folder in subversion. Each package is in its own sub-folder following this template:

#{repo }/data-fix/#{TIMESTAMP}

Where #{TIMESTAMP} is the date specified using the YYYY-MM-DD format.

6.9 *Scripts*

All shell scripts (e.g.: korn, bash, batch, PowerShell) must be stored at #{source}/scripts

- Scripts must use the target OS-specific encoding
- Scripts must use ASCII or UTF-8 encoding

6.10 *Scheduled Tasks*

Scheduled tasks are OS-specific. Unix uses CRON, and Windows uses Task Scheduler.

- Each and every task must have its own OS-specific script file as defined in Section 6.9 where the file name have the following format:
  task_<task_name>.<script_extension>
- The script may execute any other binary (e.g.: java, dotnet)
- Each and every task must have its own scheduling configuration file stored at #{source}/tasks
  - For Unix OS, in the form of a CRONTAB entry file name <task_name>.crontab
    - Must use Unix-style EOL (LF)
    - Must use ASCII encoding
  - For Windows OS, in the forms of a Task XML file, named <task_name>.xml

6.11 *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 <appName>_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 stored at #{trunk}/documentation/xmi. XMI Exports must be in XMI Version 2.1.

7 Procedures

The following procedures are provided as a convenience for the reader; they are not standards and they are subject to change.

7.1 *Deploying to DELIVERY*

Deploying to the delivery environment using Jenkins consists of two discrete steps – a build (compile) of your code and a promotion (configure and deploy to the application server).

To build using Jenkins:

1. Login to Jenkins with your personal IDIR account: https://a100.gov.bc.ca/int/jenkins
2. Select the application you want to build. It will likely be named APP-<appName>
3. Select 'Build with Parameters'.
4. Select the location in Subversion you wish to build from (e.g., the trunk or a specific tag).
5. Enter the version number as #.#.# (Major.Minor.Patch). Do not add the RIN.
6. Select 'build'.
7. To view the output of the build select 'Console Output'.
    1. If you have errors, fix them and commit your changes to Subversion.
8. Repeat until you have no errors.

To promote using Jenkins:

1. Login to Jenkins with your personal IDIR account: https://a100.gov.bc.ca/int/jenkins
2. Select the build that you wish to promote.
3. Select 'Promotion Status'.
4. Enter the configuration parameters required by your app.
5. Select 'Approve'.
6. To view the output of the promotion, select 'Console Output'.
7. Test your application
    1. If you have errors, fix them and commit your changes to Subversion.
    2. Depending on your changes, you may need to rebuild the application.
8. Repeat until your application is deployed.

When you are satisfied that your application has successfully been installed in the delivery environment, Enter a NOTICE status in VMAD to indicate that you would like a tag of the trunk and Application Deliveries can proceed with QA.

Appendices

Appendix A – Example readme.txt

The template for an Auto Delivery readme.txt can be found here:
https://a100.gov.bc.ca/pub/svn/standards/trunk/auto-delivery/readme.txt

The template for a Database readme.txt can be found here:
https://a100.gov.bc.ca/pub/svn/standards/trunk/auto-delivery/readme_DBA.txt

Note: requires VPN access.

Appendix B – Sample <appName>.main.sql File

```
************************************************************
* General Error and Information Collection System
* (GENERIC)
*
* Date: Feb. 6, 2012
* Author: IMB Applications Deliveries
*
* 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

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

Appendix C – Setup and Run SQLPLUS Procedure*

1. To set up – add the following to your user or service account .profile

```
ORACLE_SID=ora11g;
export ORACLE_SID
ORACLE_BASE=/fs/u02/sw_ux/oracle;
export ORACLE_BASE
. ~oracle/xfdisplay
```

2. To test

```
userid@blewit$ . oraenv
```

```
ORACLE_SID = [ora11g] ? ora11g
The Oracle base for ORACLE_HOME=/sw_ux/oracle/product/11.2.0cl is
/fs/u02/sw_ux/oracle
```

3. To run

```
userid@blewit$ sqlplus irs@envdlvr1
```

```
SQL*Plus: Release 11.2.0.1.0 Production on Thu Mar 29 11:58:46 2012
Copyright (c) 1982, 2009, Oracle.  All rights reserved.
```