**DB Version Control Module SOP**

Purpose: There is a need to store the upgrade history for a given database instance so it can safely and easily be upgraded when a given version of an application or module is deployed. This module will serve to inform data staff of which database version a given database instance is on and when each upgrade was applied to the instance. This module is used to apply the necessary database upgrades in order to deploy a given version of an associated application or module.

Repository: [git@pichub.pifsc.gov:application-development/centralized-tools.git](mailto:git@pichub.pifsc.gov:application-development/centralized-tools.git) in the DB\_version\_control folder

Related Document: [DB Version Control Module Documentation.docx](DB%20Version%20Control%20Module%20Documentation.docx)

Method:

* Utilize a given version control system to clearly document each version of the database and associated application(s) to clearly identify which database version is required for a given application/module version. Each code commit clearly identifies what version of the database is defined or used.
* All DDL and DML commands necessary to define the database structure and data (relevant reference data or application data but not any actual data that is being managed in the database) for a given database upgrade are saved as separate SQL files with standard headings.
* A simple modular database structure is implemented to track database upgrades over time in a given database instance so that all database upgrades applied are clearly identified. Each upgrade applied to a given database adds a record into the database upgrade table (DB\_UPGRADE\_LOGS) to indicate that the upgrade has been applied (the template for database upgrade log entries can be found in DB\_version\_control\_templates.txt under the "Database Upgrade Templates" section). Based on the current version of a given database instance and a desired version the database upgrades can be easily applied to upgrade the instance.
* Each database version is defined as [MAJOR].[MINOR] where [MAJOR] is the major version of the database and [MINOR] is the minor version of the database, both the major and minor version values are integers starting from zero that are incremented by one without a leading zero. The major and minor versions are up to the discretion of the developer(s).
  + The exception to the rule is that the first version of a database should be 0.1. Otherwise when each major version is incremented the minor version should be reset to 0 for that major version (e.g. version 3.0).
* Not every application update will necessitate a corresponding database update but each code commit should have its comments reference the version of the database it uses so that the database and application code versions on any master branch commit are in sync.
* Query the DB\_UPGRADE\_LOGS\_V View to review the upgrade history of all database modules on the given database instance to determine which database upgrade files need to be executed to upgrade a given database instance to a given version.

Procedure:

* Implement a standard folder structure (shown in [Figure 1](#figure_1)) depending on the type of development project. There are two templates in the DB\_version\_control\docs\repository\_templates folder that will define the folder structure for the "database root" directory.
  + The "SQL" directory is for projects that have a dedicated database model that is not shared between multiple applications/modules (e.g. DB Version Control). This SQL folder is stored in a given module's folder since it is module-specific.
  + The "Shared\_SQL" directory is for projects that have multiple applications/modules in one repository that use the same database (e.g. PIFSC Data Set Database with bulk download module, URL verification module, and APEX data management application). This Shared\_SQL folder is stored at the root of the repository since it is shared between multiple applications/modules.
* The database root directory contains a README.txt file that outlines the policies defined in this SOP.
* The "upgrades" folder contains each individual database upgrade file that is necessary to apply a given database upgrade. The naming convention is [DATABASE NAME]\_DDL\_DML\_upgrade\_v[MAJOR].[MINOR].sql. For example [DATABASE NAME]\_DDL\_DML\_upgrade\_v0.1.sql is the first minor version of a given database.
  + Requirements: Each upgrade must be able to be executed on a blank schema in order (major/minor version) from first to last to define the database to a given version. Executing a given [DATABASE NAME]\_DDL\_DML\_upgrade\_v[MAJOR].[MINOR].sql version on a database of the previous version will upgrade the database version to the given [MAJOR].[MINOR] version (e.g. executing [DATABASE NAME]\_DDL\_DML\_upgrade\_v1.14.sql will upgrade an instance of the v1.13 database to v1.14). Each upgrade file will contain a SQL statement to define and describe the upgrade in the Database Version Control Module. The template for the headings used in the upgrade files can be found in DB\_version\_control\_templates.txt under the "Database Upgrade Templates" section.
* The database root directory contains a combined DDL/DML file ([DATABASE NAME]\_combined\_DDL\_DML.sql where [DATABASE NAME] is the name of the database with underscores instead of spaces) that will deploy the given database version on a blank database instance.
  + The combined DDL/DML file will be updated each time a new database upgrade has been developed to append the DDL/DML from the upgrade file to the end of the combined DDL/DML file so that the combined file contains all of the DDL/DML necessary to generate the necessary objects and reference/application data for a new database deployment. The template for the headings used in the combined file can be found in DB\_version\_control\_templates.txt under the "Database Upgrade Templates" section.
* The database root directory contains a DB\_version\_control\_templates.txt file that provides some standard templates for code commit messages and database upgrade files. These can be modified to be project-specific so they can be easily copied and pasted into commit messages and database upgrade files. Each line in the template file that starts with "\*\*" is informational and will not be actually added to the commit message. The first section (Code Commit Templates) contains templates for code commits and the second section (Database Upgrade Templates) contains templates for database upgrade files. Placeholders (indicated by brackets) will be replaced with actual content as indicated in the informational content.
  + For ease of use the database upgrade template content has been included in the combined and individual database upgrade files in the repository\_templates folder so the placeholders will be replaced with actual values when the database upgrade is actually developed.
* When installing a new database module or upgrading an existing database module (e.g. DB version control module) on an existing project’s database (e.g. PIFSC Data Set Database) add the database module upgrade code to the project database’s [DATABASE NAME]\_DDL\_DML\_upgrade\_v[MAJOR].[MINOR].sql file. Add a comment in the file to indicate that the given database module is being installed/upgraded. Indent the installed database module’s DDL/DML code so it is apparent that all indented code following the comment is part of the database module installation/upgrade.
  + If the database module is being upgraded then copy all of the code in the given database module’s [DATABASE NAME]\_DDL\_DML\_upgrade\_v[MAJOR].[MINOR].sql file(s) necessary to make the upgrade into the project database’s upgrade file.
  + If the database module is being installed for the first time then copy all of the code in the given database module’s [DATABASE NAME]\_combined\_DDL\_DML.sql file for the given repository version (e.g. v 0.7) into the project database’s upgrade file.
* Test the database upgrade works as expected (requires a dedicated comparison database schema that exists to confirm that the current database upgrade script works as expected)
  + \*\*Note: the [SQL scripts/drop\_all\_objects.sql](../../SQL%20scripts/drop_all_objects.sql) script can be used to drop all objects in a given schema for the comparison database but be very careful when using this script so it is not used on the development, test, or production databases or work/data may be lost.
  + In a blank database comparison schema execute the combined DDL/DML script from the previous database version and then execute the pending database upgrade DDL/DML file using a database diff tool to confirm that the current development database and the comparison database have equivalent objects.
  + In a blank database comparison schema execute the combined DDL/DML script from the pending database upgrade using a database diff tool to confirm that the current development database and the comparison database have equivalent objects.
* When application/database code is committed to a version control system clearly document each version of the database and associated application(s) to clearly identify which database version is required for a given application/module version. Each code commit clearly identifies what version of the database is defined or used. The template for code commits can be found in DB\_version\_control\_templates.txt under the "Code Commit Templates" section.
  + (Git only) Tag the revision with a standard DB version tag to indicate that the database was upgraded in the given commit (e.g. auth\_app\_db\_v0.1)

**Figure 1: Standard DB Version Control Module folder structure**

