**Database Version Control Module SOP**

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

Resources:

* 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
* [DB Version Control Module Documentation](DB%20Version%20Control%20Module%20Documentation.docx)
* [Installing or Upgrading the DB Version Control Module](Installing%20or%20Upgrading%20the%20DB%20Version%20Control%20Module.docx)
* [Template - Installing or Upgrading the Database](Template%20-%20Installing%20or%20Upgrading%20the%20Database.docx)
  + This template is provided so a given module that has had the Database Version Control Module installed can change the placeholders and reference that document in the main technical document for the module.

Method:

* (Defining DB Upgrades) 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.
* (DB Upgrade Logs) 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.
* (Defining DB Module Versions) Each database module version is defined as [MAJOR].[MINOR] where [MAJOR] is the major version of the database module and [MINOR] is the minor version of the database module, 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 module 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).
* (Determining DB Module Versions) 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 database module's version.

Procedure:

* (Version Control Folder Structure) 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 module (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.
* (Readme File) The database root directory contains a README.txt file that outlines the policies defined in this SOP.
* (DB Upgrade Files) The "upgrades" folder contains each individual database module upgrade file that is necessary to apply a given database module 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 module.
  + Requirements: Each database module upgrade must be able to be executed on a blank schema in order (major/minor version) from the lowest to highest version to deploy the database module to a given version. Executing a given [DATABASE NAME]\_DDL\_DML\_upgrade\_v[MAJOR].[MINOR].sql version on a database module of the previous version will upgrade the database module 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 module to v1.14). Each upgrade file will contain a SQL statement to define and describe the database module 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.
* (Combined DB Definition File)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 module with underscores instead of spaces) that will deploy the given database module version on a blank database instance.
  + The combined DDL/DML file will be updated each time a new database module 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 module 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.
* (Code Commit and DB Upgrade Templates) The SQL/shared\_SQL 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 module 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 module upgrade template content has been included in the combined and individual database module upgrade files in the repository\_templates folder so the placeholders will be replaced with actual values when the database module upgrade is actually developed.
* (Standard DB Installation/Upgrade Documentation) The "Installing or Upgrading the Database Documentation.docx" is a standard document that provides information for installing/upgrading a given database module that should be linked from the given application's/module's main technical document so the database module upgrade process is clearly defined.
  + When installing the DB version control module the "[Template - Installing or Upgrading the Database Documentation.docx](Template%20-%20Installing%20or%20Upgrading%20the%20Database%20Documentation.docx)" should be copied to a given module's code repository's directory structure (and renamed) and all placeholders should be replaced with their actual values based on the guidance in the document comments.
* (Module Installations/Upgrades) 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 module'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 module's upgrade file.
* (Testing Database Upgrades) Test the database upgrade works as expected (requires a dedicated comparison database schema that exists to confirm that the current database module 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 to ensure 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 module version and then execute the pending database upgrade DDL/DML file. Use 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 module upgrade. Use a database diff tool to confirm that the current development database and the comparison database have equivalent objects.
* (Version Control Code Commits) When application/database code is committed to a version control system clearly document each version of the database module and associated application(s) to clearly identify which database module version is required for a given application/module version. Each code commit clearly identifies what version of the database module is defined or used, not every application update will necessitate a corresponding database module upgrade but each code commit should have its comments reference the version of the database module it uses so that the database module and application code versions on any master branch commit are in sync. 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 module version tag to indicate that the database module was upgraded in the given commit (e.g. auth\_app\_db\_v0.1)
    - (Git only) Git tag messages for DB module upgrades should have a standard format. Example: "version [VERSION NUMBER] of the [MODULE NAME] database" where [VERSION NUMBER] is the version number of the module and [MODULE NAME] is the name of the application/database module. (e.g. version 0.1 of the Authorization Application Module database)
  + (Git only) When installing/upgrading an external DB module define a tag on the revision that matches the tag defined for the version of the external DB module that was installed or upgraded to (e.g. db\_vers\_ctrl\_db\_v0.5)
    - (Git only) Git tag messages for external DB module upgrades should have a standard format. Example: "Installed [DB GIT TAG MESSAGE]" where [DB GIT TAG MESSAGE] is the tag message for the corresponding git tag in the version of the external DB module that was installed. (e.g. installed version 0.5 of the DB version control module database)

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

