**Data Validation Module Documentation**

* Terms used:
  + Parent record - The parent record for all child records in a given data stream. Each parent record has a foreign key reference (PTA\_ERROR\_ID) to the associated parent error record.
    - For example the SPT\_VESSEL\_TRIPS table contains the parent records for the RPL data stream and SPT\_UL\_TRANSACTIONS table contains the parent records for the UL data stream.
  + Parent error record (DVM\_PTA\_ERRORS) - The parent record that all error type and error records are associated with when the data validation module is processed.
  + Error record (DVM\_ERRORS) - These records represent individual instances of data validation issues identified by the data validation module and are associated with parent error records.
* System Requirements:
  + PHP 5.4.x
  + SVN repository "Shared\_Library" directory contents: svn://badfish.pifsc.gov/CRED/PHP\_code/web/crei/Shared\_Library
  + Git repository application\_code directory contents from Git repository: [git@pichub.pifsc.gov:sptt/sptt.git](mailto:git@pichub.pifsc.gov:sptt/sptt.git)
  + Windows Operating System configured to execute PHP from the command line (add the PHP module directory containing php.exe to the Windows system PATH variable)
  + php.ini file used when executing PHP via the command line (located in c:\Windows by default) must be configured to load the PHP OCI extension (e.g. extension=php\_oci8.dll) to enable PHP to connect to the Oracle database
  + OCI driver must be installed and configured (instructions can be found here: <http://pichub.pifsc.gov/centralized-data-tools/access-to-oracle-migration/blob/master/docs/Oracle_OCI_driver_install.docx>)
* Overview: The module was developed to provide a framework to validate data entered in the database based on flexible data validation criteria that can be developed and implemented in Oracle Views by a data manager/developer without requiring application development skills. A series of QC Views can be developed to identify problematic values in the database and implemented in the framework to allow each criteria to be evaluated on the given data stream. A given parent record (e.g. SPT\_VESSEL\_TRIPS) of a given data stream (e.g. RPL) and all associated child records (e.g. SPT\_VESSEL\_TRIP\_EVTS, SPT\_SET\_CATCH, etc.) will be validated as a group based on the QC Views defined. Each validation issue that is identified will be saved as a separate error record that includes detailed information about the type of error and a descriptive error message with contextual information that will allow a given issue to be quickly identified, these error records are associated with the parent error record. These error records can be queried and exported easily in a single unified report for resolution by data management staff.
* Features:
  + Point in time architecture (PTA): The module will save all error criteria that were active at the time that the data was first validated by the module. This will allow the data to be re-validated with the same criteria that were active when it was entered which was implemented so new criteria would not be evaluated on older data which can potentially cause some problems.
  + Data validation issues can be accepted based on manual annotations and an Error Resolution Type. This allows data issues that cannot be resolved or false positives to be marked as valid with the annotation as an explanation.
* Implementation
  + Current Implementation
    - PHP script that is chained to the XML import module and runs automatically on a specific fishing trip as each fishing trip is imported into the relational data model
      * This process is executed by running application\_code\rpl\_import.bat
    - PHP script that can be run as a stand-alone process to re-evaluate all data validation criteria defined at the time the data stream was first validated on existing data using the data validation framework. The XML data source files that are re-validated are determined by the results of an XML data source file query. All existing Error records are purged and replaced when a given fishing trip is re-evaluated
      * This process is executed by running application\_code\re\_run\_validation.bat
      * \*\*Note: there is a query in re\_run\_validation.php that is used to specify the XML\_FILE\_ID values of each SPT\_APP\_XML\_FILES parent error record that will be re-evaluated by the data validation module.
    - QC Queries are implemented for groups of tables that comprise a data stream and the resultant Error records are associated with the given parent error record for a given data stream.
      * \*\*Note: Since the Data Validation Module and the XML Import Module were developed together the module currently uses the SPT\_APP\_XML\_FILES parent error record table that represents the source XML data file the RPL data was loaded from to associate the errors for the associated parent record and all child records. There is a feature in the XML Import Module that allowed the rollback of all RPL data while maintaining the errors for reporting and resolution purposes.
  + How to define data validation criteria
    - See external document [here](How%20to%20Define%20Criteria%20in%20Data%20Validation%20Module.docx)
  + Error Records
    - Each individual data issue identified by the Data Validation Module is represented by a separate Error record (DVM\_ERRORS) that includes a description of the error that contains all relevant database values associated with the given data record(s) at the time of evaluation. The Error Type is specified as well as the severity of the error (e.g. warning vs. fatal error).
    - The ability to re-run validation has been implemented preliminarily. The framework purges all existing Error records and re-uses the existing DVM\_ERRORS and DVM\_ERROR\_TYPES records when evaluating the associated Error Types that existed when the parent record was first evaluated (defined by DVM\_PTA\_ERR\_TYP\_ASSOC records).
  + Error Resolution
    - When an error record represents a legitimate value or legitimate set of values a data manager has the ability to enter a manual annotation on the corresponding DVM\_ERRORS record itself and set the ERROR\_NOTES value to an explanation of the reason the error should be considered a false positive (e.g. fishing in IATTC area) or is otherwise exempted (e.g. there is no way to determine the field value). Also, the ERR\_RES\_TYPE\_ID value can be set to the PK of an existing Error Resolution Types (DVM\_ERR\_RES\_TYPES) record that corresponds to the resolution type (currently: No Data Available, Manually Reviewed and Accepted, and No Resolution Can be Reached Yet) to allow the resolution types to be grouped.
    - \*\*Note: there is currently no interface that has been developed to enter accept and annotate data validation issues. This can be developed potentially as part of a larger data management system.
  + Error Report Queries (all Views have comments on all columns and the object itself):
    - DVM\_PTA\_ERRORS\_V – (PTA Errors (View)) This View returns all unresolved Errors associated with a given PTA Error record that were identified during the last evaluation of the associated PTA Error Types. A PTA Error record can be referenced by any data table that represents the parent record for a given data stream (e.g. SPT\_VESSEL\_TRIPS for RPL data). The query returns detailed information about the specifics of each error identified as well as general information about the given Error's Error Type. Each associated date/time is provided as a standard formatted date in MM/DD/YYYY HH24:MI format.
    - DVM\_PTA\_ERROR\_TYPES\_V – (PTA Error Types (View)) This View returns all Error Types associated with a given PTA Error Type record. The PTA Error Type record is defined the first time the given data stream is first entered into the database, all active Error Types at this point in time are saved and associated with a new PTA Error Type record and this is referenced by a given parent record of a given data stream (e.g. SPT\_VESSEL\_TRIPS for RPL data). Each associated date/time is provided as a standard formatted date in MM/DD/YYYY HH24:MI format. This relationship is used to determine the Error Types that were associated with a data stream when the given parent record is first entered into the database.
    - SPT\_XML\_PTA\_ERROR\_TYPE\_V – (XML File PTA Error Types (View)) This View returns all PTA Error Types associated with each XML file that has been processed to date based on the date the given data stream was first entered into the database (defined by CREATE\_DATE). The purpose of this table is to save all active QC Error Type criteria and associate it with a given data stream when it is first entered so that the same validation criteria can be re-evaluated instead of just the QC criteria that is currently active.
      * \*\*Note: this is used specifically for the eTunaLog XML Import module
    - SPT\_XML\_PTA\_ERRORS\_V – (XML File PTA Errors (View)) This View returns all outstanding PTA Errors associated with each XML file that has been processed to date based on the last time the QC validation was evaluated (defined by LAST\_EVAL\_DATE). All dates are returned as native data types as well as standard formatted date strings and all reference table values are included in the result set.
      * \*\*Note: this is used specifically for the eTunaLog XML Import module
  + Error Rule Query
    - DVM\_QC\_CRITERIA\_V – (QC Criteria (View)) This View returns all QC Criteria (Error Types) defined in the database and their associated QC Object, Error Severity, and Error Category. This query is used to define all PTA Error Types when a data stream is first validated in the database
* Algorithm Used:
  + Actual QC validation process is performed by the xml\_data\_loader class defined in the class definition file: application\_code\functions\xml\_data\_loader.php
  + Configuration File (application\_code\constants.php):
    - DB\_HOST, DB\_USER, DB\_PASS constants determine the Oracle database instance and data schema used by the validation process
    - APPLICATION\_INCLUDE\_PATH constant determines the location of the functions directory (for application portability). This will be defined as the local directory the Git repository application\_code directory contents are located
    - SHARED\_LIBRARY\_INCLUDE\_PATH constant determines the location of the shared library (for application portability) This will be defined as the local directory the SVN repository "Shared\_Library" directory contents are located
    - XML\_BASE\_DIR constant determines the XML data directory that is examined by the XML Import Module
  + XML Import Module Execution:
    - When a set of records is entered into the database (e.g. fishing trip with associated trip events and catch information) the parent record (e.g. SPT\_VESSEL\_TRIPS) is then used to validate its own values as well as all associated child record values using the parent record’s primary key value to filter the various records.
      * Before the SPT\_APP\_XML\_FILES record is inserted into the database a new record is inserted into the parent error record table first (DVM\_PTA\_ERRORS) and then the SPT\_APP\_XML\_FILES record is inserted with the PK value from the PTA tables for the PTA\_ERROR\_ID field. This record allows any parent record with this foreign key field (PTA\_ERROR\_ID) to be associated with multiple Error Types and Errors.
        + A requirement of any parent record for a given data stream must have a foreign key field defined: PTA\_ERROR\_ID and associated with the DVM\_PTA\_ERRORS table so that the data validation module can associate errors and error types directly with the parent record.
    - The retrieve\_QC\_error\_info() method of the xml\_data\_loader.php class retrieves all QC criteria for all active Error Types (defined when ERR\_TYPE\_ACTIVE\_YN = 'Y') for active QC Objects (defined when QC\_OBJ\_ACTIVE\_YN = 'Y'). These validation criteria are stored in the xml\_data\_loader object in the property (xml\_validation\_rules) for evaluation
      * \*\*Note: The process is currently hardcoded for the eTunaLog RPL data stream since it is currently chained to the XML import process
    - [Evaluate QC Criteria](#Eval_QC_Criteria)
  + Re-Run Validation Execution:
    - Purge the associated Error records (DVM\_ERRORS) for the existing parent error record
    - Query for all Error Types that are associated with the existing parent error record that were active when the given parent record was first processed by the data validation module. These validation criteria are stored in the xml\_data\_loader object in the property (xml\_validation\_rules) for evaluation
    - [Evaluate QC Criteria](#Eval_QC_Criteria)
  + Evaluate QC Criteria
    - The validation process is handled by the evaluate\_QC\_criteria() method of the xml\_data\_loader.php class.
    - Once the appropriate data validation criteria has been loaded in the xml\_data \_loader object for the given data stream parent record both types of data validation module executions evaluate the QC criteria.
      * The evaluate\_QC\_criteria() method loops through each QC View object defined in the xml\_validation\_rules object property and constructs a SELECT \* FROM [OBJECT\_NAME] from the DVM\_QC\_CRITERIA\_V query with a WHERE clause using the parent table’s primary key field (e.g. WHERE VESS\_TRIP\_ID = :vtid where :vtid is the primary key value of the SPT\_VESSEL\_TRIPS record being evaluated).
        + \*\*Note: Currently in the XML import module implementation the primary key field name is hardcoded and the value is taken from the PHP object after parent record insertion.
        + The results from each QC query are then looped through and an inner loop iterates through each IND\_FIELD\_NAME value to check its value in the result set.

If the corresponding field value is ‘Y’ it indicates the presence of a data validation issue so an element is added to an internal object array property (error\_array) for logging all errors associated with the parent record. These values stored in error\_array include the primary key value for the corresponding DVM\_ERROR\_TYPES record and the ERROR\_TYPE\_COMMENT\_TEMPLATE string value with all placeholders in brackets replaced by the corresponding field values in the same query result set to generate a meaningful error message that provides specific context to the nature of the validation issue and how to identify the record(s) to examine.

If the corresponding field value in 'N' it indicates the absence of a data validation issue so no action is needed.

* + - * + \*\*Note (Applies to XML Import Module only): log\_validation\_error\_types() runs to associate each active DVM\_ERROR\_TYPES record with the parent error record by inserting DVM\_PTA\_ERR\_TYPE\_ASSOC records for the validation criteria that was active when the data validation module was first evaluated on the given parent record to allow subsequent validation module executions to re-use the same rules.
        + log\_validation\_errors() runs to log each individual data validation issue that was found during the QC evaluation process. The internal array property (error\_array) is iterated through and an error record is inserted (DVM\_ERRORS) for each data validation issue instance.

\*\*Note: the current implementation is to associate these error records with the SPT\_APP\_XML\_FILES but in the future it should be modified to allow the error records to be associated with the parent record directly.

* Core tables (all tables have comments on all columns and the object itself):
  + DVM\_QC\_OBJECTS – (Data QC Objects) This is a reference table that defines all of the QC validation views that are executed on the data model after a given data stream is loaded into the database (e.g. SPT\_VESSEL\_TRIPS, SPT\_UL\_TRANSACTIONS, SPT\_APP\_XML\_FILES).
    - DVM\_ERROR\_TYPES – (Data Error Types) This is a reference table that defines the different QC Data Error Types that indicate how to identify QC errors and report them to end-users for resolution. Each Data Error will have a corresponding Data Error Type.
      * DVM\_DATA\_STREAMS - (Data Streams) This is a reference table that defines all data streams that are implemented in the SPTT data set. This reference table is referenced by the DVM\_ERROR\_TYPES to define the data stream that the given error type is associated with. This reference table is also referenced by the SPT\_APP\_XML\_POST\_PROC to define the data stream that a given post processing procedure is executed for. Examples of data streams are RPL, eTunaLog, UL, FOT, LFSC. This is used to filter these records based on the given context of the processing/validation
      * DVM\_ERR\_SEVERITY – (Error Severity) This is a reference table that defines all error severities for error type criteria. This indicates the status of the given error type criteria (e.g. warnings, data errors, violations of law, etc.)
      * DVM\_ERRORS – (Data Errors) This is an error table that represents any specific data error instances that a given data table/entity contains (e.g. SPT\_VESSEL\_TRIPS, SPT\_UL\_TRANSACTIONS, SPT\_APP\_XML\_FILES, etc.). These records will be used to communicate errors to the data entry and data management staff
        + DVM\_ERR\_RES\_TYPES - (Error Resolution Types) This is a reference table that defines all error resolutions types that will be used to define the status of a given QC data validation issue. When an error is marked as resolved it will have an annotation to explain the resolution of the error. The types of error resolutions include no data available, manually reviewed and accepted, no resolution can be reached yet. Certain error resolutions types will cause an error to be filtered out from the standard error reports.)
  + Parent Table – This is the given parent table that is enabled in the data validation module (e.g. SPT\_VESSEL\_TRIPS, SPT\_CANN\_TRANSACTIONS, etc.).
    - DVM\_PTA\_ERRORS – (Errors (PTA)) This table represents a generalized intersection table that allows multiple Error records to reference this consolidated table that allows multiple Errors to be associated with the given table record (e.g. SPT\_VESSEL\_TRIPS, SPT\_UL\_TRANSACTIONS, etc.).
      * DVM\_PTA\_ERR\_TYP\_ASSOC – (Error Type Associations (PTA)) This intersection table allows multiple Error Types to be associated with a given table (e.g. SPT\_VESSEL\_TRIPS, SPT\_UL\_TRANSACTIONS, etc.). These associations represent the Error Types that are defined at the time that a given table record is created so that the specific rules can be applied for subsequent validation assessments over time.

Legend:

Completed

* Future plans for additional/modified functionality:
  + Develop import process to allow the QC template spreadsheet to be imported directly into the DVM\_QC\_OBJECTS and DVM\_ERROR\_TYPES tables to remove the requirement to manually enter this data which can be time-consuming and potentially error-prone (possibly just import into temp table and use merge queries to insert/update records into the corresponding tables as necessary).
  + Update algorithm to associate the error records directly with the parent record (e.g. SPT\_VESSEL\_TRIPS vs. SPT\_APP\_XML\_FILES since only the RPL data entered via XML import module will have an associated SPT\_APP\_XML\_FILES record). When this change is made there would be no distinction between parent error records and parent records.
  + Enhance the re\_run\_validation() method to maintain existing error records that are identical to the new error records and retain any annotations made to these records so as not to lose the manual work completed by data management staff
    - Potential Soln: Identify existing records with the same generated ERROR\_DESCRIPTION values (excluding PK value references) and maintain those existing error records if those errors persist after the data update in order to maintain the error resolution code (ERR\_RES\_TYPE\_ID) and error annotations (ERROR\_NOTES). This could be done using the Oracle Function REMOVE\_ERR\_MESSAGE\_PK\_REFS when applied to the ERROR\_DESCRIPTION of a new error and the existing error to identify matches between the errors.
  + Implement the Data Validation Module as a stand-alone PL/SQL package that can be used directly in the database and executed via APEX, PHP, etc. The logic in the evaluate\_QC\_criteria() and re\_run\_validation() methods would need to be implemented in stored procedures in a package.
    - DB update - define the primary key field name in the DVM\_DATA\_STREAMS table so that it can be used as an argument in the data validation package.
    - Procedures could accept the primary key value and the DATA\_STREAM\_CODE and that should be enough to determine which record and what parameter is included in each query.
    - A main procedure could actually execute the Views and enter the Error records with the generated contextual content and could be called once the QC Views/Error Types that are used to validate have been determined from both the evaluate\_QC\_criteria and re\_run\_validation procedures.
    - Include a stored procedure that can batch re-evaluate records
  + Implement URL generating capabilities into the framework to allow the QC Views to generate their own URLs for application pages that will allow the given error to be resolved easily (e.g. APEX edit trip event link, PHP DM page link). We would need to add to the fields in the DVM\_ERRORS table and define a designated IND\_FIELD to pull this information from (e.g. DATA\_URL). Could implement in foundational views as well and just pull directly into QC query as the link could be valuable in other contexts
  + There is a slow query that needs to be fixed when time permits (SPT\_QC\_EVT\_DIST\_ISSUE\_V). The execution slows down considerably as the database is populated.
  + Develop simple generalized interface to allow errors to be annotated in web interface
    - Could be part of a more generalized application that could be used throughout the Center
  + Look into combining and generalizing the SPT\_PTA\_ERRORS and SPT\_PTA\_ERROR\_TYPES tables (e.g. SPT\_PTA\_VALIDATIONS) since they seem to be redundant. Could use a single key that two different tables reference. This would simplify the logic and reduce the requirements for implementing framework on other data streams.
  + Generalize objects and use less-specific names for the different database entities. (e.g. should not refer to things as errors since some are warnings, maybe use the term "data issue" instead). All object names should be changed accordingly as well.