BIOCHEM Database Application System

BIOCHEM Archival/Edits System Document

Prepared for

The Department of Fisheries and Oceans Canada
Application Services Division – Bedford Institute of Oceanography
Informatics Branch, Maritimes Region
Version 5.0 as of 14 May 2003

BIOCHEM Archival Application

The BIOCHEM Archival/Edits Application is designed to facilitate the validation, archival, and editing of Discrete and Plankton data sets. Once archived this data may be queried using the BIOCHEM Query Application version 4.0. This document provides a detailed description of the Oracle Forms 6i and Oracle Reports 6i objects that make up the BIOCHEM Archival/Edits Application version 5.0, which is scheduled to be deployed regionally, via the SMS, in early May 2003. The following information is provided in an effort to describe the application from the Application developer's perspective. It can be used as a starting point for future development to the System.

Before modifying the application components that make up the system it is important to understand how the system functions.

Data is initially loaded into one of a number of remote data manager accounts where it is validated against business rules specified by the client. Once validated, the data is transferred from the data manager account to the main data repository. After being loaded to the archive it can be queried using the BIOCHEM Query Application. Although the current version of the BIOCHEM Application only supports Discrete and Plankton data, future version will also support Continuous, Blob, Environmental and Profile data, thus making the BIOCHEM system a widely used and valuable tool by many different regions.

This document makes reference to many database and BIOCHEM Query components that are beyond its scope. For descriptions of these components the following documents may be consulted:

BCEDITS_data_dictionary_v5.0 – Describes all the objects contained in a data manager account.
BCEDITS_installation_guide_v5.0 – Describes the steps and scripts involved in setting up a data manager account, as well as, hardware and hardware requirements for optimal performance.
BCARCHIVE_data_dictionary_v5.0 – Describes all the objects contained in the archive/audit account.
BCARCHIVE_installation_guide_v_5.0 - Describes the steps and scripts involved in setting up the archive/audit account, as well as, hardware and hardware requirements for optimal performance.
BCQUERY_data_dictionary_v4.0 – Describes all the objects contained in the query account.
BCQUERY_installation_guide_v4.0 **-** Describes the steps and scripts involved in setting up the archive/audit account, as well as, hardware and hardware requirements for optimal performance.
BCQUERY_System_document_v4.0 – Describes all the Oracle Forms 6i that make up the BIOCHEM Query Application.

While the above documents are targeted mainly at the Application Developer or Database Analyst the following documentation my be consulted for a users perspective of the systems:

BIOCHEM_Archival_User_Guide_v5.0 – Describes the functionality of the BIOCHEM Archival System version 3.2 from the perspective of the end user.
BCQUERY_User_Guide_v3.0 – Describes the functionality of the BIOCHEM Query Application from the user's perspective.

BIOCHEM Archival Application Components

This section provides a list of all the Oracle Forms 6i and Oracle Reports 6i objects that make up version 5.0 of The BIOCHEM Archival/Edits Application System. All components listed in this section reside on the ASD development server: D05. The objects reside in multiple folders depending on the stage of development they are in.
For the most recent deployed versions of the objects visit the folder called *Currently Deployed/Biochem/L_Forms.*
For the most recent developed versions of the objects visit the folder called *In Development/Biochem/Forms.*

Oracle FORMS 6i Objects

The Oracle Forms 6i objects that make up the Biochem Archival/Edits Application version 5.0 are categorised in one of three types: Form modules, menu module, or library modules.

Form modules
The Form modules can logically be separated into four types of forms: Lookup table forms, data EDIT table forms, menu form, switchboard form, or download forms.

Lookup table forms - supports the insertion, deletion, and modification of look-up table records. There is one forms module for every lookup table supported by BioChem.
Data EDIT table forms - support the deletion, and modification of the scientific data of interest. There is one forms module for every data manager EDIT table supported by BioChem.
Menu forms - supports the selection of section of interest.
Switchboard forms - supports the processing of Plankton and Discrete data.
Download forms - supports the download of Discrete or Plankton data from the archive.

The following table lists the Oracle Forms 6i modules that make up the Biochem Archival/Edits Application version 5.0. Many of the modules listed below are associated with a database table from which its name was derived.

Form Source	**Purpose/Type**
LookUp Table Forms	
BC_ANALYSIS.FMB	BCANAYLSIS Lookup Table Form
BC_COLLECTION_METHOD.FMB	BCCOLLECTIONMATHODS Lookup Table Form.
BC_DATA_RETRIEVAL.FMB	BCDATARETRIEVALS Lookup Table Form.
BC_DATA_TYPE.FMB	BCDATATYPES Lookup Table Form.
BC_NATNL_TAXONCODE.FMB	BCNATNLTAXONCODES Lookup Table Form.
BC_GEARS.FMB	BCGEARS Lookup Table Form
BC_LIFE_HISTORY.FMB	BCLIFEHISTORIES Lookup Table Form.
BC_PRESERVATION.FMB	BCPRESERVATIONS Lookup Table Form
BC_PROCEDURE.FMB	BCPROCEDURES Lookup Table Form
BC_QUAL_CODE.FMB	BCQUALCODES Lookup Table Form
BC_SAMPLE_HANDLING.FMB	BCSAMPLEHANDLINGS Lookup Table Form
BC_SECURITY.FMB	Displays a record from the BCUSERS. User data may be inserted (new user) or edited in order to correct validation errors.
BC_SEX.FMB	BCSEXES Lookup Table Form
BC_STORAGE.FMB	BCSTORAGES Lookup Table Form
BC_TROPHIC_DESCRIPTOR.FMB	BCTROPHICDESCRIPTORS Lookup Table Form
BC_UNIT.FMB	BCUNITS Lookup Table Form
BC_VOLUME_METHOD.FMB	BCVOLUMEMETHODS Lookup Table Form
BC_DATA_TYPE_CLASSES.FMB	BCDATATYPECLASSES Lookup Table Form
BC_NATNL_TAXON_CLASS.FMB	BCNATNLTAXONCLASSES Lookup Table Form
Data EDIT Table Forms	
BC_MISSIONEDITS.FMB	BCMISSIONEDITS Data Table Form

BC_EVENTEDITS.FMB	BCEVENTEDITS Data Table Form
BC_DISCRETE_HEADEREDITS.FMB	BCDISCRETEHEDREDITS Data Table Form
BC_DISCRETEDETAILEDITS.FMB	BCDISCRETEDTAILEDITS Data Table Form
BC_DISCRETEREPLICATEEDITS.FMB	BCDISREPLICATEDITS Data Table Form
BC_PLANKTON_HEADEREDITS.FMB	BCPLANKTNHEDREDITS Data Table Form
BC_PLANKTON_GENERALEDITS.FMB	BCPLANKTNGENRLEDITS Data Table Form
BC_PLANKTON_FREQUENCYEDITS.FMB	BCPLANKTNFREQEDITS Data Table Form
BC_PLANKTON_DTAILEDITS.FMB	BCPLANKTNDTAILEDITS Data Table Form
BC_PLANKTON_INDIVEDITS.FMB	BCPLANKTNINDIVEDITS Data Table Form
Menu Forms	
BC_MAIN.FMB	The main menu form for the BIOCHEM application.
BC_REFERENCE_OBJECTS.FMB	BIOCHEM toolbar associated with all the forms in this table.. Included objects in this form are: Form-level triggers: MENU_COUNT, MENU_EXEQUERY, MENU_ENTQUERY, MENU_SAVE, MENU_EXIT, KEY-EXEQRY, KEY-ENTQRY, KEY-EXIT, KEY-COMMIT Forms-Controls-BUTTONS: EXIT, SAVE, INSERT_BUT, QUERY, EXECUTE_QUERY, CANCEL_Q, FIRST_RECORD, PREVIOUS_RECORD, NEXT_RECORD, LAST_RECORD, BACK, GO_RECORDS_BUT, EDIT_BUT, LOOK_UP_BUT, MAIN_BUT
Switchboard Forms	
BC_BATCH.FMB	Discrete Switchboard form for validating, editing, archiving and own loading Discrete data.
BC_PLANKTON_BATCH.FMB	Plankton Switchboard form for validating, editing, archiving and own loading Plankton data.
Download Forms	
BC_DISCRETE_DOWNLD.FMB	Discrete Download form for downloading discrete data from the archive.
BC_PLANKTON_DOWNLD.FMB	Plankton Download form for downloading plankton data from the archive.

Global Variables

Global variables are used in the forms for such tasks as cross-form communication and user identification. This section provides a list of all global variables used in the Application and a description of each. It has been recommended as a future task to investigate and provide alternatives to the use of these global variables.

variable name	**Description**
global.error	Error flag used in the initial validation procedures.
global.bc_unit_seq	Stores the selected unit_seq value from the Bc_data_retrieval and Bc_data_type forms when the user chooses to open the BC_unit form from either the Bc_data_retrieval and Bc_data_type forms. This value is used to query the BCUNITS table and display the results in the Bc_unit form.
global.bc_storages_seq	Stores the selected storage_seq value from the Bc_data_type form when the user chooses to open the BC_Storage form from the Bc_data_type form. This value is used to query the BCSTORAGES table and display the results in the Bc_ Storage form.
global.bc_samplehandlings_seq	Stores the selected samplehandling_seq value from the Bc_data_type form when the user chooses to open the BC_Sample_handling form from the Bc_data_type form. This value is used to query the BCSAMPLEHANDLINGS table and display the results in the BC_Sample_handling form.

global.bc_preservation_seq	Stores the selected preservation_seq value from the Bc_data_type form when the user chooses to open the BC_Preservation form from the Bc_data_type form. This value is used to query the BCPRESERVATIONS table and display the results in the BC_Preservation form.
global.bc_analyses_seq	Stores the selected analysis_seq value from the Bc_data_type form when the user chooses to open the BC_Analysis form from the Bc_data_type form. This value is used to query the BCANALYSES table and display the results in the BC_Analysis form.
global.user	Stores the user name of the logged in user.
global.bc_data_retrieval_seq	Stores the selected data_retrieval_seq value from the Bc_data_type form when the user chooses to open the BC_data_retrieval form from the Bc_data_type form. This value is used to query the BCDATARETRIEVALS table and display the results in the BC_data_retrieval form.
global.bc_natnltaxoncodes_seq	This variable is no longer used since the BCNODCMASTERS table and form no longer exist in the application.
global.header	Stores the Discrete Header sequence value (dis_headr_edt_seq) from the button press of the VIEW_DISCRETE_DETAILS button to open the Discrete Details form from the Discrete Header form. Value is accessed when querying the data in the DISCRETEDETAILEDITS_V view by setting the default-where clause in the button press of the View Records Button on the C_DISCRETEHEDREDITS canvas in the BC_DISCRETEDETAILEDITS form. This action will limit the retrieved records based on the value of the global variable if it is NOT NULL.
global.mission	Stores the Mission sequence value (mission_edt_seq) from the button press of the EVENT_BUT button (to open the Events form from the Mission form. Value is accessed when querying the data in the EVENTEDITS_V view by setting the default-where clause in the button press of the View Records Button on the C_EVENTEDITS_V canvas in the BC_EVENTEDITS form. This action will limit the retrieved records based on the value of the global variable if it is NOT NULL.
global.event	Stores the Event sequence value (event_edt_seq) from the button press of the HEADER_BUT button (to open either the BC_PLANKTON_HEADEREDITS or BC_DISCRETE_HEADEREDITS form from the Event form. Discrete: Value is accessed when querying the data in the DISCRETEHEDREDITS_V view by setting the default-where clause in the button press of the View Records Button on the C_DISCRETEHEDREDITS_V canvas in the BC_DISCRETEHEDREDITS form. Plankton: Value is accessed when querying the data in the PLANKTONHEDREDITS_V view by setting the default-where clause in the button press of the View Records Button on C_PLANKTONHEDREDITS_V canvas in the BC_PLANKTONHEADEREDITS form. This action will limit the retrieved records based on the value of the global variable if it is NOT NULL.
global.plankton	Stores a flag value to denote if the Bc_PLANKTON_batch form is open ('Y'). This variable is initialised in the WHEN-NEW-FORM-INSTANCE trigger in the BC_PLANKTON_BATCH form. Referenced by the HEADER_BUT button press on the C_EVENTEDITS canvas on the BC_EVENTEDITS form to open the BC_PLANKTONHEADEREDITS form if the variable value is 'Y' and otherwise open the BC_DISCRETEHEDREDITS form.

global.batch_seq	Stores the value of the user selected batch_seq value from the BC_Batch form or the BC_PLANKTON_batch form. This value is assigned in the button pressed triggers of the buttons, which open the various edit forms (Mission, Event, Discrete Header, Discrete Detail, Discrete Replicate, Plankton Header, and Plankton Detail).
global.username	Calls the get_application_property built-in to initialise the variable with the logged in user name of the user. This is performed in the OPEN_FORMS procedure in the BIOCHEM_LIB.
global.TAXmaster	Variable not referenced any longer. Meant to reference the BCNODCMASTERS code selected to pass the value to the Bc_Natnl_taxoncode form.

Menu module
The Biochem menu module inthe following table is associated with all the forms listed above

Menu Source	Purpose
MAIN.MMB	Contains the menu options available to the currently active form

Library module
The **BIOCHEM** library module is the central point of execution and code maintenance at the forms level. The library includes packages to handle security, data input, validation, forms navigation, and message reporting. The next section provides a detailed description of the packages that are contained in the library along with the description of the procedures in each package. The library module is the connect point between the application forms and the PL/SQL code stored in the data manager accounts.

Library Source	Purpose
BIOCHEM_LIB.PLL	Contains all packages/procedure that are shared among the forms.

Oracle Reports 6I objects

The Biochem Archival/Edits Application v4.0 currently supports the generation of two Oracle Reports 6I reports. The reports provide a summary of errors at the two validation points of the system.

Report Source	Description
BC_STATN_DATA_ERRORS.RDF	Details the initial validation errors with the data from the BCDISCRETESTATNEDITS, BCDISCRETEDATAEDITS, OR BCPLANKTONSTATNEDITS, BCPLANKTONDATAEDITS tables. Report is executed from the Plankton or Discrete batch summary error processing forms.
BC_EDITS_ERROR_SUMMARY.RDF	Details/summarises the validation errors for a selected batch of data. Executed from the Plankton or Discrete batch summary error processing forms.

BIOCHEM_LIB Library Module

The **BIOCHEM_LIB.PLL** library is the central point of execution and code maintenance at the forms level. The library is made up of packages, procedure, and functions to handle security, data input, validation, data archival, data deletion, forms navigation, and message reporting. Some of the packages in the library monitor the execution of stored procedures and functions that are contained in the data manager database account. Procedures in these

packages call the PL/SQLstored procedures that are contained in the data manager database account, and control the forms execution based on the return values observed.

This section provides a detailed description of the **BIOCHEM_LIB** library packages and their procedures. It lists the procedures contained in each package, the name of the objects that it uses, along with the names of the objects that use it. Also provided in this section is a detailed description of the procedures and functions that make up each library package. This description includes the names of any stored procedures that are called the respective procedure/function.

Many of the objects that are listed in this section are database objects that are either stored in the main BIOCHEM archival database or the data manager accounts. The description for these objects is beyond the scope of this document. For a description of the objects please consult one of the following documents:

BCEDITS_data_dictionary_v4.0 – Describes all the objects contained in a data manager account.
BCARCHIVE_data_dictionary_v4.0 – Describes all the objects contained in the archive/audit account.

This library contains the following programming objects:

Library Object Name	Object Type	Description
ARCHIVED_BATCH	Function	Returns true if the batch has been previously loaded to archive.
ARCHIVED_MISSION	Function	Returns the mission associated with the given archived batch.
DELETE_PKG	Package	Contains the Procedure to delete a record(of any type) using the toolbar delete button.
FORM_SECURITY	Package	This package contains the functions and procedures to authenticate and set the privileges of the use.
GET_DATA	Package	This package contains the procedures and functions to populate a form field, supplied as an input parameter, with a text value using the numeric code value.
OPEN_FORMS	Procedure	This procedure declares and initializes all of the global variables needed for the application. One global variable currently exists for each form in order to track if it is open or not. Global variables also exist for the purpose of "passing" data to called forms and to "pass" data to called procedures.
PROCS	Package	This package contains procedures, which have varied purposes. It contains an assortment of procedures that did not logically fit in any of the other packages contained in the library. They are grouped together to make the code more manageable.
SAVED_PROCS_PKG	Package	This package contains one procedure that is executed when the user pushed the save button on the application tool bar. The procedure re-executes the validation if there have been changes to the form fields that have not been saved.
TOOLBAR_MENU_BUTTON	Package	This package contains procedures that take care of the some of the processing contained in the CONTROL blocks and menus on all of the forms. This provides a central code maintenance point for the CONTROL block and menu functionality.

FUNCTION ARCHIVED_BATCH (batch_seq_par IN NUMBER) RETURN BOOLEAN

Uses:
BCLOCKEDMISSIONS - Archival Table.
BCLOCKEDMISSIONS - Data Manager synonym.
BCMISSIONEDITS - Data Manager Table.

Used By:
PROCS - Library Package.

FUNCTION ARCHIVED_MISSION (batch_seq_par IN NUMBER) RETURN BOOLEAN

Uses:
BCMISSIONEDITS - Data Manager Table.

Used By:
PROCS - Library Package.

DELETE_PKG

Procedures/Functions:
PROCEDURE DELETE_RECORD (current_form_par IN VARCHAR2, functional_area_par IN VARCHAR2)

Uses:
PROCS - Library Package.
BCMISSIONEDITS - Data Manager Table.
BCEVENTEDITS - Data Manager Table.
BCDISCRETEHEDREDITS - Data Manager Table.
BCDISCRETEDTAILEDITS - Data Manager Table.
BCDISREPLICATEITS - Data Manager Table.
BCPLANKTNHEDREDITS - Data Manager Table.
BCPLANKTNGENERLEDITS - Data Manager Table.
BCPLANKTNDTAILEDITS - Data Manager Table.
BCPLANKTNFREQEDITS - Data Manager Table.
BCPLANKTNINDIVDLEDITS - Data Manager Table.
BCDISCRETEREPLICTEDITSDEL - Data Manager Table.
GET_FUNCTIONAL_AREA - Data Manager Stored Function.

Used By:
BC_REFERENCE_OBJECTS - forms module.

FORM_SECURITY

Procedures/Functions:
FUNCTION CHECK_PERMISSIONS RETURN NUMBER;
PROCEDURE CHECK_USER (block_name_par IN VARCHAR2, control_1_par IN VARCHAR2,control_2_par IN VARCHAR2);

Uses:
BCUSERS – Archival Table.

Used By:
TOOLBAR_MENU_BUTTON – Library Package .

FUNCTION CHECK_PERMISSIONS
Checks the username in the BCUSERS table to determines which group they belong to and what permissions they have. The return value of this function represents a numeric value for the level of privilege associated with a user. This user privilege information is obtained from the BCUSERS table; the following values represent the level of privilege (stored in the BCUSERS table):

'BC_USER'
'BC_DATA_MANAGER'
'BC_SUPER_USER'

The 'BC_DATA_MANAGER' and 'BC_SUPER_USER' have all the rights to the application and database tables. The 'BC_USER' may only view the data via the application and may not insert or save any data.

PROCEDURE CHECK_USER
This procedure sets the properties of the form based on the value that is returned BCUSERS by the check_permissions function. The passed block name referenced has its INSERT_ALLOWED, UPDATE_ALLOWED, DELETE_ALLOWED properties set to FALSE if the user is a 'BC_USER'. If the user is of type BC_DATA_MANAGER', 'BC_SUPER_USER' then the appropriate properties remain TRUE and the passed control names referenced are enabled in order to permit editing.

GET_DATA

The procedures in this package, suffixed with “GET_” retrieve data from the database tables based on the code value passed as a parameter. The retrieved data is then written to the block.column name referenced as a passed parameter via the COPY built-in. The remaining procedures:

DISCRETE_COUNT_OF_RECORDS
DISCRETE_SUM_ERRS_BATCH_FORM
PLANKTON_COUNT_OF_RECORDS
PLANKTON_SUM_ERRS_BATCH_FORM

call the "GET_SUM_..." or "GET_COUNT_..." procedures to retrieve and populate the various calculated columns in the forms. The calculated columns refer to the count of the records in a table (level of data) for a batch of data, and the sum of all errors at a level of data (table) for a batch. This will aid the user in identifying where errors are found.

Procedures/Functions:

PROCEDURE GET_DATA_CENTER_NAME(datacenter_number_par IN NUMBER, datacenter_column_par IN VARCHAR2);
PROCEDURE GET_BATCH_NAME(batch_seq_in_par IN NUMBER, batch_column_par IN VARCHAR2);
PROCEDURE GET_VOLUME_METHOD_NAME(volume_seq_in_par IN NUMBER, vol_method_column_par IN VARCHAR2);
PROCEDURE GET_COLLECTION_METHOD_NAME(collection_seq_in_par IN NUMBER, collection_column_par IN VARCHAR2);
PROCEDURE GET_PROCEDURE_NAME(procedure_seq_in_par IN NUMBER, procedure_column_par IN VARCHAR2);
PROCEDURE GET_PRESERVATION_NAME(preservation_seq_in_par IN NUMBER, preservation_column_par IN VARCHAR2);
PROCEDURE GET_STORAGE_NAME(storage_seq_in_par IN NUMBER, storage_column_par IN VARCHAR2);
PROCEDURE GET_SEX_NAME(sex_seq_in_par IN NUMBER, sex_column_par IN VARCHAR2);
PROCEDURE GET_TROPHIC_NAME(trophic_seq_in_par IN NUMBER, trophic_column_par IN VARCHAR2);
PROCEDURE GET_LIFE_STAGE_NAME(life_seq_in_par IN NUMBER, life_column_par IN VARCHAR2);
PROCEDURE GET_NATIONAL_TAXON_NAME(taxon_seq_in_par IN NUMBER, taxon_column_par IN VARCHAR2);
PROCEDURE GET_GEAR_CODE_NAME(gear_code_par IN NUMBER, gear_code_column_par IN VARCHAR2);
PROCEDURE GET_DATATYPE_NAME(datatype_code_par IN NUMBER, datatype_column_par IN VARCHAR2);
PROCEDURE GET_UNIT_NAME(unit_seq_in_par IN NUMBER, unit_column_par IN VARCHAR2);
PROCEDURE GET_ANALYSIS_NAME(analysis_seq_in_par IN NUMBER, analysis_column_par IN VARCHAR2);
PROCEDURE GET_SAMPLE_HANDLING_NAME(sample_seq_in_par IN NUMBER, sample_column_par IN VARCHAR2);
PROCEDURE GET_DATA_RETRIEVAL_NAME(data_retrieval_seq_in_par IN NUMBER, data_retrieval_column_par IN VARCHAR2);
PROCEDURE GET_SUM_MISSION_ERRORS(batch_seq_in_par IN NUMBER, mission_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_EVENT_ERRORS(batch_seq_in_par IN NUMBER, event_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_DISCRETE_HEDR_ERRORS(batch_seq_in_par IN NUMBER, hedr_errs_in_par IN VARCHAR2);

PROCEDURE GET_SUM_DISCRETE_DETAIL_ERRORS(batch_seq_in_par IN NUMBER, detail_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_DISCRETE_REPLIC_ERRORS(batch_seq_in_par IN NUMBER, replicate_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_PLANKTON_HEDR_ERRORS(batch_seq_in_par IN NUMBER, hedr_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_PLANKTON_INDIV_ERRORS(batch_seq_in_par IN NUMBER, indiv_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_PLANKTON_FREQ_ERRORS(batch_seq_in_par IN NUMBER, freq_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_PLANKTON_DTAIL_ERRORS(batch_seq_in_par IN NUMBER, dtail_errs_in_par IN VARCHAR2);
PROCEDURE GET_SUM_PLANKTON_GENERL_ERRORS(batch_seq_in_par IN NUMBER, general_errs_in_par IN VARCHAR2);
PROCEDURE GET_COUNT_MISSION_RECS(batch_seq_in_par IN NUMBER, mission_in_par IN VARCHAR2);
PROCEDURE GET_COUNT_EVENT_RECS(batch_seq_in_par IN NUMBER, event_in_par IN VARCHAR2);
PROCEDURE DISCRETE_SUM_ERRS_BATCH_FORM(batch_seq_par IN NUMBER, mission_errs_par IN VARCHAR2, event_errs_par IN VARCHAR2, header_errs_par IN VARCHAR2, detail_errs_par IN VARCHAR2, replicate_errs_par IN VARCHAR2);
PROCEDURE PLANKTON_COUNT_OF_RECORDS(batch_seq_par IN NUMBER, batch_name_par IN VARCHAR2, mission_count_par IN VARCHAR2, event_count_par IN VARCHAR2, header_count_par IN VARCHAR2, generals_count_par IN VARCHAR2);
PROCEDURE PLANKTON_SUM_ERRS_BATCH_FORM(batch_seq_par IN NUMBER,mission_errs_par IN VARCHAR2, event_errs_par IN VARCHAR2, header_errs_par IN VARCHAR2, general_errs_par IN VARCHAR2);
PROCEDURE DISCRETE_COUNT_OF_RECORDS(batch_seq_par IN NUMBER, batch_name_par IN VARCHAR2, mission_count_par IN VARCHAR2, event_count_par IN VARCHAR2, header_count_par IN VARCHAR2, detail_count_par IN VARCHAR2, replicate_count_par IN VARCHAR2);

Uses:

BCDATACENTERS – Archival LookupTable
BCBATCHES – Data Manager Table
BCVOLUMEMETHODS – Archival LookupTable
BCCOLLECTIONMETHODS – Archival LookupTable
BCPROCEDURES – Archival LookupTable
BCPRESERVATION – Archival LookupTable
BCSTORAGES – Archival LookupTable
BCSEXES – Archival LookupTable
BCTROPHICDESCRIPTORS – Archival LookupTable
BCLIFEHISTORIES – Archival LookupTable
BCNATNLTAXONCODES – Archival LookupTable
BCGEARS – Archival LookupTable
BCDATATYPES – Archival LookupTable
BCUNITS – Archival LookupTable
BCANALYSES – Archival LookupTable
BCSAMPLEHANDLINGS – Archival LookupTable
BCDATARETRIEVALS – Archival LookupTable
MISSIONEDITS_FULL_V – Data Manager View
EVENTEDITS_V – Data Manager View
DISCRETEHEDREDITS_V – Data Manager View
DISCRETEDETAILEDITS_V – Data Manager View
BCDISCREPLICATES_V – Data Manager View
PLANTNHEDREDITS_V – Data Manager View
PLANKTNGENERLEDITS_V – Data Manager View
PLANKTNDTAILEDITS_V – Data Manager View
PLANKTNFREQEDITS_V – Data Manager View

PLANKTNINDIVEDITS_V – Data Manager View
BCMISSIONEDITS – Data Manager Table
BCEVENTEDITS – Data Manager Table
BCPLANKTNHEDREDITS – Data Manager Table
BCPLANKTNGENERLEDITS – Data Manager Table
BCDISCRETEHEDREDITS – Data Manager Table
BCDISCRETEDTAILEDITS - Data Manager Table
BCDISREPLICATEDITS - Data Manager Table
BCPLANTNFREQEDITS - Data Manager Table
BCPLANKTNDTAILEDITS - Data Manager Table
BCPLANKTNINDIVEDITS - Data Manager Table

Used By:
All lookup and Data EDIT forms.

PROCEDURE GET_DATA_CENTER_NAME
This procedure retrieves the data center name from the BCDATACENTERS table based on the value of datacenter_number_par (the passed parameter, which references the data center, code value). Call the COPY built-in to write the name into the column name (on the form) passed in the datacenter_column_par parameter.

PROCEDURE GET_BATCH_NAME
This procedure retrieves the batch name from the BCBATCHES table based on the value of batch_seq_in_par (the passed parameter, which references the batch, sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the batch_column_par parameter.

PROCEDURE GET_VOLUME_METHOD_NAME
This procedure retrieves the volume method name from the BCVOLUMEMETHODS table based on the value of volume_seq_in_par (the passed parameter, which references the volume method, sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the vol_method_column_par parameter.

PROCEDURE GET_COLLECTION_METHOD_NAME
This procedure retrieves the collection method name from the BCCOLLECTIONMETHODS table based on the value of collection_seq_in_par (the passed parameter, which references the collection method, sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the collection_column_par parameter.

PROCEDURE GET_PROCEDURE_NAME
This procedure retrieves the procedure name from the BCPROCEDURES table based on the value of procedure_seq_in_par (the passed parameter which references the procedure sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the procedure_column_par parameter.

PROCEDURE GET_PRESERVATION_NAME
This procedure retrieves the preservation name from the BCPRESERVATIONS table based on the value of preservation_seq_in_par (the passed parameter which references the preservation sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the preservation_column_par parameter.

PROCEDURE GET_STORAGE_NAME
This procedure retrieves the storage name from the BCSTORAGES table based on the value of storage_seq_in_par (the passed parameter which references the storage sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the storage_column_par parameter.

PROCEDURE GET_SEX_NAME
This procedure retrieves the sex name from the BCSEXES table based on the value of sex_seq_in_par (the passed parameter which references the sex sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the sex_column_par parameter.

PROCEDURE GET_TROPHIC_NAME
This procedure retrieves the trophic descriptor name from the BCTROPHICDESCRIPTORS table based on the value of trophic_seq_in_par (the passed parameter, which references the trophic descriptor, sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the trophic_column_par parameter.

PROCEDURE GET_LIFE_STAGE_NAME
This procedure retrieves the life history (stage) name from the BCLIFEHISTORIES table based on the value of life_seq_in_par (the passed parameter which references the life histories sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the life_column_par parameter.

PROCEDURE GET_NATIONAL_TAXON_NAME
This procedure retrieves the national taxonomic name from the BCNATNLTAXONCODES table based on the value of taxon_seq_in_par (the passed parameter which references the national taxonomic sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the taxon_column_par parameter.

PROCEDURE GET_GEAR_CODE_NAME
This procedure retrieves the gear name from the BCGEARS table based on the value of gear_code_par (the passed parameter which references the gear sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the gear_code_column_par parameter.

PROCEDURE GET_DATATYPE_NAME
This procedure retrieves the datatype name from the BCDATATYPES table based on the value of datatype_code_par (the passed parameter which references the datatype sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the datatype_column_par parameter.

PROCEDURE GET_UNIT_NAME
This procedure retrieves the unit name from the BCUNITS table based on the value of unit_seq_in_par (the passed parameter which references the unit sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the unit_column_par parameter.

PROCEDURE GET_ANALYSIS_NAME
This procedure retrieves the analysis name from the BCANALYSES table based on the value of analysis_seq_in_par (the passed parameter which references the analysis sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the analysis_column_par parameter.

PROCEDURE GET_SAMPLE_HANDLING_NAME
This procedure retrieves the sample handling name from the BCSAMPLEHANDLINGS table based on the value of sample_seq_in_par (the passed parameter which references the sample handling sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the sample_column_par parameter.

PROCEDURE GET_DATA_RETRIEVAL_NAME
This procedure retrieves the data retrieval name from the BCDATARETRIEVALS table based on the value of data_retrieval_seq_in_par (the passed parameter which references the data retrieval sequence value). Call the COPY built-in to write the name into the column name (on the form) passed in the data_retrieval_column_par parameter.

PROCEDURE GET_SUM_MISSION_ERRORS
This procedure retrieves the sum of the errors in the BCMISSIONEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the mission_errs_in_par parameter.

PROCEDURE GET_SUM_EVENT_ERRORS

This procedure retrieves the sum of the errors in the BCEVENTEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the event_errs_in_par parameter.

PROCEDURE GET_SUM_DISCRETE_HEDR_ERRORS
This procedure retrieves the sum of the errors in the BCDISCRETEHEDREDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the hedr_errs_in_par parameter.

PROCEDURE GET_SUM_DISCRETE_DETAIL_ERRORS
This procedure retrieves the sum of the errors in the BCDISCRETEDTAILEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the detail_errs_in_par parameter.

PROCEDURE GET_SUM_DISCRETE_REPLIC_ERRORS
This procedure retrieves the sum of the errors in the BCDISREPLICATEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the replicate_errs_in_par parameter.

PROCEDURE GET_SUM_PLANKTON_HEDR_ERRORS
This procedure retrieves the sum of the errors in the BCPLANKTNHEDREDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the hedr_errs_in_par parameter.

PROCEDURE GET_SUM_PLANKTON_GENERL_ERRORS
This procedure retrieves the sum of the errors in the BCPLANKTNGENERLEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the general_errs_in_par parameter.

PROCEDURE GET_SUM_PLANKTON_DTAIL_ERRORS
This procedure retrieves the sum of the errors in the BCPLANKTNDTAILEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the detail_errs_in_par parameter.

PROCEDURE GET_SUM_PLANKTON_FREQ_ERRORS
This procedure retrieves the sum of the errors in the BCPLANKTNFREQEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the frequency_errs_in_par parameter.

PROCEDURE GET_SUM_PLANKTON_INDIV_ERRORS
This procedure retrieves the sum of the errors in the BCPLANKTNINDIVEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the individual_errs_in_par parameter.

PROCEDURE GET_COUNT_MISSION_RECS
This procedure retrieves the count of the records in the BCMISSIONEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the

COPY built-in to write the calculated value into the column name (on the form) passed in the mission_in_par parameter.

PROCEDURE GET_COUNT_EVENT_RECS
This procedure retrieves the count of the records in the BCEVENTEDITS table based on the value of batch_seq_in_par (the passed parameter which references the batch sequence value of the dataset). Call the COPY built-in to write the calculated value into the column name (on the form) passed in the event_in_par parameter.

PROCEDURE DISCRETE_SUM_ERRS_BATCH_FORM
This procedure retrieves the sum of the errors in the BCMISSIONEDITS, BCEVENTEDITS, BCDISCRETEHEDREDITS, BCDISCRETEDTAILEDITS, and BCDISCRETEREPLICATEDITS tables based on the value of batch_seq_in_par (the passed parameter, which references the batch, sequence value of the dataset).
Calls the following procedures (defined above):

GET_SUM_MISSION_ERRORS(batch_seq_par, mission_errs_par);
GET_SUM_EVENT_ERRORS(batch_seq_par, event_errs_par);
GET_SUM_DISCRETE_HEDR_ERRORS(batch_seq_par, header_errs_par);
GET_SUM_DISCRETE_DETAIL_ERRORS(batch_seq_par, detail_errs_par);
GET_SUM_DISCRETE_REPLIC_ERRORS(batch_seq_par, replicate_errs_par);

Each of these procedures calls the COPY built-in to write the calculated values into the column names (on the form) passed in the mission_errs_par, event_errs_par, header_errs_par, detail_errs_par, and replicate_errs_par parameters.

PROCEDURE PLANKTON_COUNT_OF_RECORDS
This procedure retrieves the count of the records in the BCMISSIONEDITS, BCEVENTEDITS, BCPLANKTNHEDREDITS, and BCPLANKTNGENERLEDITS tables based on the value of batch_seq_in_par (the passed parameter, which references the batch, sequence value of the dataset).
Calls the following procedures (defined above):

GET_BATCH_NAME(batch_seq_par, batch_name_par);
GET_COUNT_MISSION_RECS(batch_seq_par, mission_count_par);
GET_COUNT_EVENT_RECS(batch_seq_par, event_count_par);

Cursors are created to calculate the count of the number of records in a batch for each of the BCPLANKTNHEDREDITS and BCPLANKTNGENERLEDITS tables. Each of these procedures calls the COPY built-in to write the calculated values/column names into the column names (on the form) passed in the mission_count_par, event_count_par, header_count_par, generals_count_par, and batch_name_par parameters.

PROCEDURE PLANKTON_SUM_ERRS_BATCH_FORM
This procedure retrieves the sum of the errors in the BCMISSIONEDITS, BCEVENTEDITS, BCPLANKTNHEDREDITS, and BCPLANKTNGENERLEDITS tables based on the value of batch_seq_in_par (the passed parameter, which references the batch, sequence value of the dataset).
Calls the following procedures (defined above):

GET_SUM_MISSION_ERRORS(batch_seq_par, mission_errs_par);
GET_SUM_EVENT_ERRORS(batch_seq_par, event_errs_par);
GET_SUM_PLANKTON_HEDR_ERRORS(batch_seq_par, header_errs_par);
GET_SUM_PLANKTON_GENERL_ERRORS(batch_seq_par, general_errs_par);
GET_SUM_PLANKTON_DTAIL_ERRORS(batch_seq_par, detail_errs_par);
GET_SUM_PLANKTON_FREQ_ERRORS(batch_seq_par, frequency_errs_par);
GET_SUM_PLANKTON_INDIV_ERRORS(batch_seq_par, individual_errs_par);

Each of these procedures calls the COPY built-in to write the calculated values into the column names (on the form) passed in the mission_errs_par, event_errs_par, header_errs_par, and general_errs_par parameters.

PROCEDURE DISCRETE_COUNT_OF_RECORDS
This procedure retrieves the count of the records in the BCMISSIONEDITS, BCEVENTEDITS, BCDISCRETEHEDREDITS, BCDISCRETEDTAILEDITS, and BCDISCRETEREPLICATEDITS tables based on the value of batch_seq_in_par (the passed parameter, which references the batch, sequence value of the dataset).
Calls the following procedures (defined above):

GET_BATCH_NAME(batch_seq_par, batch_name_par);
GET_COUNT_MISSION_RECS(batch_seq_par, mission_count_par);
GET_COUNT_EVENT_RECS(batch_seq_par, event_count_par);

Cursors are created to calculate the count of the number of records in a batch for each of the BCDISCRETEHEDREDITS, BCDISCRETEDTAILEDITS, and BCDISCRETEREPLICATEDITS tables. Each of these procedures calls the COPY built-in to write the calculated values/column names into the column names (on the form) passed in the mission_count_par, event_count_par, header_count_par, detail_count_par, replicate_count_par, and batch_name_par parameters.

OPEN_FORMS

Uses:
None

Used By:
All Forms.

PROCS

Procedures/Functions:
PROCEDURE CREATE_BATCH_SEQ_NAME(created_date_in_par IN DATE, functional_area_par IN VARCHAR2);
PROCEDURE ASSIGN_BATCH_SEQ(created_date_in_par IN DATE, in_column_par IN VARCHAR2, functional_area_par IN VARCHAR2);
PROCEDURE VALIDATE_STATION_DATA(created_date_par IN DATE, functional_area_par IN VARCHAR2, created_date_column_par IN VARCHAR2);
PROCEDURE UPDATE_DISCRETE_DETAIL(dis_detail_edt_seq_par IN NUMBER, batch_seq_par IN NUMBER);
PROCEDURE LOAD_TO_ARCHIVALS(batch_seq_par IN NUMBER, batch_column_name IN VARCHAR2, functional_area_par IN VARCHAR2);
PROCEDURE INTIAL_ERRORS_REPORT(created_date_par IN DATE, date_column_name IN VARCHAR2, functional_area_par IN VARCHAR2);

Uses:
POPULATE_DISCRETE_EDITS_PKG – Data Manager Stored Package
BATCH_VALIDATION_PKG - Data Manager Stored Package
POPULATE_PLANKTON_EDITS_PKG - Data Manager Stored Package
VALIDATE_DISCRETE_STATN_DATA - Data Manager Stored Package
VALIDATE_PLANKTON_STATN_DATA - Data Manager Stored Package
ITEM_VALIDATION_PKG - Data Manager Stored Package
ARCHIVE_BATCH - Data Manager Stored Package
BC_PROCS_DB_PKG - Data Manager Stored Package
BCBATCHES – Data Manager Table
BCDISCRETESTATNEDITS – Data Manager Table
BCPLANKTONSTATNEDITS – Data Manager Table
BCUSERS – Archival Table
DUAL – Table
BCDISCRETEDATAEDITS – Data Manager Table
BCPLANKTONSTATNEDITS – Data Manager Table
BCSTATNDATAERRORS – Data Manager Table
BCDISCRETEDTAILEDITS – Data Manager Table
BCQUALCODES – Archival LookupTable
BCDISREPLICATEDITS – Data Manager Table
BCERRORS – Archival Table
BCACTIVITYEDITS - Data Manager Table
BATCH_SEQ – Data Manager Sequence
BCLOCKEDMISSIONS - Archival Table

Used By:
SAVED_PROCS_PKG – Library Package

PROCEDURE CREATE_BATCH_SEQ_NAME
This procedure creates the batch sequence name by parsing the distinct batch_seq values for the dataset being processed. The distinct batch_seq values are converted to a character string concatenated together and then truncated to 30 characters and assigned as the batch name. If there is only one distinct value for the batch_seq value then that one value is assigned as the batch name. If there is no batch_seq value provided in the "Station/Data" tables then the batch name takes on the value of the newly generated batch_seq value stored in BCBATCHES.batch_seq column. The parameters are described as follows:

created_date_in_par IN DATE - the user selected created_date column value which specifies the dataset, which is to be validated.

functional_area_par IN VARCHAR2 - the functional area reference variable which enables the procedure to process the data correctly. This procedure will handle both Discrete and Plankton data. The acceptable values for this parameter are 'DISCRETE' or 'PLANKTON'.

PROCEDURE ASSIGN_BATCH_SEQ
This procedure processes the data in the BCDISCRETESTATNEDITS/BCDISCRETEDATAEDITS or BCPLANKTONSTATNEDITS/BCPLANKTONDATAEDITS tables (dependant upon the area of functionality the user has executed the command). The processing of the data begins with the execution of the CREATE_BATCH_SEQ_NAME procedure, which "generates" a batch name for the dataset and assigns it to the BCBATCHES.name table column.
The processing of the data continues with the loading of the data from the 'Station/Data' tables into the Edit tables (BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, BCACTIVITYEDITS) using procedures stored in the data manger database account. The following are the stored procedures called by this procedure:
POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS
POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS

These procedures perform the initial validation of the data. All data validation procedures write any validation errors to the BCERRORS table and assign the process_flag column value 'ERR' to signify that the record is not valid due to a column in the record failing the validation. If an exception fires in the validation procedures, it is propagated out to the forms front end in order to inform the user of the error in processing the data. The data is committed and then all of the blocks on the discrete or plankton batch switchboard form (dependant upon the area of functionality) are re-queried via the EXECUTE_QUERY built-in.

PROCEDURE VALIDATE_STATION_DATA
This procedure performs the initial validation of the "Station/Data" table data based on the value of the created_date_par parameter (refers to the user selected created_date value which defines a dataset). If records fail the initial validation, the errors are written to the BCSTATNDATAERRORS table and each record that fails the validation is assigned a process_flag of 'SVE' , for "Station" table data or 'DVE' , for "Data" table data. The user may then generate a report (BC_Edits_Error_Summary.rdf) to view the "invalid" data, correct the data, reset the process_flags for the records in the dataset to 'NR', then try to validate the data once again.

If the user has not selected a created_date value from the list then display a message to prompt the user to select a valid value. The created_date value is passed, as a parameter, to the following stored procedures which will perform the initial validations on the data, for the selected batch of data, in the "Station/Data" tables:
VALIDATE_DISCRETE_STATN_DATA.VALIDATE_DISCRETE_STATION (created_date_par)
VALIDATE_DISCRETE_STATN_DATA.VALIDATE_DISCRETE_DATA (created_date_par)
VALIDATE_PLANKTON_STATN_DATA.VALIDATE_PLANKTON_STATION(created_date_par)
VALIDATE_PLANKTON_STATN_DATA.VALIDATE_PLANKTON_DATA (created_date_par)

If the data in the "Station/Data" tables fail the initial validations, then the data is flagged as invalid and the user may generate a report to display the problems in the data. If the data passes the initial validations then the following procedure is called:
ASSIGN_BATCH_SEQ(created_date_par, 'batch_seq', functional_area_par) (Described above)

PROCEDURE UPDATE_DISCRETE_DETAIL
This procedure updates a BCDISCRETEDTAILEDITS record with the averaged data value, max value of the detection limit (or NULL if any detection limit is NULL), and the max value of the QC code values (or 0 if any one of the QC code values is 0) for all replicate data that has the same collector sample id value and the same data type sequence value in the BCDISREPLICATEDITS table. The data value validation check is then performed to ensure that the averaged data value is still valid. This procedure calls the following stored function:
ITEM_VALIDATION_PKG.VALIDATE_DATA_VALUE

PROCEDURE LOAD_TO_ARCHIVALS
This procedure calls the stored package procedures, to load the data from the Data Manager Edit tables into the BIOCHEM Archival tables. The passed parameters are defined as follows:

batch_seq_par IN NUMBER - value of the user selected batch_seq column. Used to select the dataset which is to be processed.

batch_column_name IN VARCHAR2 - the block name for the column name of the column that contains the batch_seq value, which is selected by the user.

functional_area_par IN VARCHAR2 - the area of functionality which the user is currently using. Valid values are 'DH' for Discrete and 'PL' for Plankton.

The procedure ensures that a valid value was selected by the user. If not, the user is prompted to insert a valid value in order to proceed. If a valid value has been selected, the functional_area_par value is tested to ensure that the proper procedure call is executed since each area of functionality has separate load to archival procedures. The stored procedures that perform the load to the archival tables are:
ARCHIVE_BATCH.ARCHIVE_DISCRETE_BATCH(batch_seq_par, proc_status_v);
ARCHIVE_BATCH.ARCHIVE_PLANKTON_BATCH (batch_seq_par, proc_status_v);
If the proc_status_v parameter is NOT NULL then display a message to the user informing them of an error while loading he data from the Edit tables to the Archival tables. If the load is successful, display a message to the user to notify them of the successful load of the data from the Edits tables to the Archival tables.

PROCEDURE INTIAL_ERRORS_REPORT
This procedure initiates the errors report (BC_Statn_Data_Errors.rdf) to display the validation errors encountered during the initial validation. These error are stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS tables. This report will display all of the initial validation errors from these tables. Then delete the data in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS tables, which have batches of data containing a process_flag value of 'SVE' or 'DVE'. If the user does not select a valid batch_seq value a message is displayed alerting the user that they have not selected a valid batch to process.

- Allow the user to select a batch of data to process (report or delete)(rather than all or nothing approach).
- Provided the user with decision points where they may delete the selected batch of data without running the report or cancelling the operation all together.
- Re-formatted the report and now pass the report a parameter list consisting of the user selected created_date column value (on which to base the report) and the command to shut down the Oracle Reports runtime engine once the report has been closed.
- The user now has the option to delete the records from the BCPLANKTONSTATNEDITS/BCPLANKTONDATAEDITS or the BCDISCRETESTATNEDITS/BCDISCRETEDATAEDITS (dependent upon the functional area from which the report is executed) tables based on the user selected created_date column value (on which the report was based).

The procedure parameters are defined as follows:

created_date_par IN DATE - the user selected created_date column value on which the report is to be based.

date_column_name IN VARCHAR2 - the block name dot column name of the column that contains the user selected created_date value.

functional_area_par IN VARCHAR2 - the area of functionality which the user is currently using. Valid values are 'DISCRETE' for Discrete and 'PLANKTON' for Plankton. Value is further passed to the BC_PROCS_DB_PKG.DELETE_FROM_STATN_DATA procedure to point to

the proper portion of the procedure to execute. The following stored package procedure is called to delete the "Station/Data" records based on the value of the passed functional_area_par and the user selected created_date_par passed value.(BC_PROCS_DB_PKG.DELETE_FROM_STATN_DATA(created_date_par, functional_area_par)). The data is then re-queried.

SAVED_PROCS_PKG

Procedures/Functions:
PROCEDURE MAIN_SAVE_PROC(current_form_par IN VARCHAR2);

Uses:
TOOLBAR_MENU_BUTTON – Library Package.
MISSION_EEVENT_REC_VALID_PKG – Stored Package
DISCRETE_REC_VALID_PKG – Stored Package
PROCS – Library Package
PLANKTON_REC_VALID_PKG – Stored Package
ITEM_VALIDATION_PKG – Stored Package
BCUSERS – Table

Used By:
TOOLBAR_MENU_BUTTON – Library Package.

PROCEDURE MAIN_SAVE_PROC
This procedure validates the data and then saves the changes to the data on the form.
The procedure accepts the current_form_par parameter which passes the name of the form from which the MAIN_SAVE_PROC procedure has been called (using the built-in: GET_APPLICATION_PROPERTY(current_form_name).
The MAIN_SAVE_PROC procedure is called from the Save button in the BC_REFERENCE_OBJECTS form (which in turn is referenced by all of the forms via object referencing), and from the TOOLBAR_MENU_BUTTON.EXITPROMPT library procedure. This procedure tests the value of the passed current_form_par parameter and executes the following data manager stored procedures/functions as is appropriate in order to validate the data before committing it to the database tables:

MISSION_EVENT_REC_VALID_PKG.missionedits_validate_form - validates the Mission form record(s).
MISSION_EVENT_REC_VALID_PKG.eventedits_validate_form - validates the Event form record(s).
DISCRETE_REC_VALID_PKG.VALIDATE_DISCRETEHEDREDITS_FRM - validates the Discrete Header form record(s).
DISCRETE_REC_VALID_PKG.VALIDATE_DISCRETEDTAILEDIT_FRM - validates the Discrete Detail form record(s).
DISCRETE_REC_VALID_PKG.VALIDATE_DISREPLICATEDIT_FRM - validates the Discrete Replicate form record(s).
PROCS.UPDATE_DISCRETE_DETAIL - validates the Discrete Detail form record(s).
PLANKTON_REC_VALID_PKG.VALIDATE_BCPLANKTNHEDREDIT_FRM - validates the Plankton Header form record(s).
PLANKTON_REC_VALID_PKG.VALIDATE_BCPLNKTNGNRLEDITS_FRM - validates the Plankton General form record(s).
PLANKTON_REC_VALID_PKG.VALIDATE_BCPLNKTNFREQEDITS_FRM - validates the Plankton Frequency form record(s).
PLANKTON_REC_VALID_PKG.VALIDATE_BCPLNKTNDTAILEDITS_FRM - validates the Plankton detail form record(s).
PLANKTON_REC_VALID_PKG.VALIDATE_BCPLNKTNINDIVEDITS_FRM - validates the Plankton Individual form record(s).
ITEM_VALIDATION_PKG.VALIDATE_NAME_ITEM - validates the individual

columns for the specified tables' records. This validation is called many times since many of Code Table forms only require this level of validation for one or more columns. If the specified validation passes the record is saved. If the specified validation fails, then the record is not committed, and a message to alert the user of invalid data is displayed.

IF the current form has been called from either the 'BC_DATA_RETRIEVAL' or 'BC_DATA_TYPE' then POST the changes to the data rather than committing them. Alert the user of the success/failure for committing the data.

TOOLBAR_MENU_BUTTON

Procedures/Functions:

PROCEDURE delete_button_press (table_name1 IN VARCHAR2, table_name2 IN VARCHAR2, table_name3 IN VARCHAR2, table_name4 IN VARCHAR2, table_name5 IN VARCHAR2, table_name6 IN VARCHAR2, table_name7 IN VARCHAR2, table_name8 IN VARCHAR2, column_name IN VARCHAR2, block_column_name IN VARCHAR2);
PROCEDURE delete_button_press (table_name1 IN VARCHAR2, table_name2 IN VARCHAR2, table_name3 IN VARCHAR2, table_name4 IN VARCHAR2, table_name5 IN VARCHAR2, table_name6 IN VARCHAR2, table_name7 IN VARCHAR2, table_name8 IN VARCHAR2, table_name9 IN VARCHAR2, column_name IN VARCHAR2, block_column_name IN VARCHAR2);
PROCEDURE delete_button_press (table_name1 IN VARCHAR2, table_name2 IN VARCHAR2, table_name3 IN VARCHAR2, table_name4 IN VARCHAR2, table_name5 IN VARCHAR2, table_name6 IN VARCHAR2, table_name7 IN VARCHAR2, table_name8 IN VARCHAR2, table_name9 IN VARCHAR2, table_name10 IN VARCHAR2, table_name11 IN VARCHAR2, column_name IN VARCHAR2, block_column_name IN VARCHAR2);
PROCEDURE DELETE_FROM_TABLES(table_name_in_par IN VARCHAR2, column_name_in_par IN VARCHAR2, column_value_in_par IN VARCHAR2);
PROCEDURE TOGGLE_ENABLE_ITEM_PROPERTY(control_par IN VARCHAR2, setting_par IN VARCHAR2);
PROCEDURE ENTQUERY(control_1_par IN VARCHAR2, control_2_par IN VARCHAR2, control_3_par IN VARCHAR2);
PROCEDURE EXEQUERY(control_1_par IN VARCHAR2, control_2_par IN VARCHAR2, control_3_par IN VARCHAR2);
PROCEDURE INSERT_PROC;
PROCEDURE RECORD_STATUS_CHECK_AND_NAV(navigate_par IN VARCHAR2);
PROCEDURE DISPLAY_CONTROL(control_name_par IN VARCHAR2, text_par IN VARCHAR2, position1_par IN NUMBER, position2_par IN NUMBER, size1_par IN NUMBER, size2_par IN NUMBER);
PROCEDURE DISPLAY_CONTROL_OFF(control_name_par IN VARCHAR2);
PROCEDURE OPEN_FORM_BUTTON(form_name_par IN VARCHAR2);
PROCEDURE EXITPROMPT;
PROCEDURE AUTO_HINT_TEXT(control_name_par IN VARCHAR2, display_mode_par IN VARCHAR2);
PROCEDURE GO_RECORD_DETAILS(block_name_par IN VARCHAR2, item_name_par IN VARCHAR2);
PROCEDURE LARGE_COMMENT_EDITOR(comment_column_par IN VARCHAR2, editor_name_par IN VARCHAR2);
PROCEDURE LOV_CHOICES(lov_name_par IN VARCHAR2);
PROCEDURE REVALIDATE_BATCH;
PROCEDURE RUN_ERROR_SUMMARY_REPORT(batch_seq_par IN NUMBER, batch_column_name IN VARCHAR2);

Uses:

SAVED_PROCS_PKG – Library Package

Used By:

SAVED_PROCS_PKG – Library Package
FORM_SECURITY – Library Package
BATCH_VALIDATION_PKG – Data Manager Stored Packege
BCUSERS – Archival Table
BCACTIVITYEDITS - Data Manager table.

PROCEDURE delete_button_press
The DELETE_BUTTON_PRESS procedures use the procedure "overloading" method to maintain the use of the same procedure name although each procedure is executed dependant upon the number of parameters that are passed as part of the procedure call. Each of these procedures listed below, named DELETE_BUTTON_PRESS deletes specified records (dependant on the value obtained by the NAME_IN built-in when it is passed the block_column_name parameter) from the tables listed as parameters passed to the procedures (referenced as table_name1 ... table_name11). The column_name parameter forms the column name in the where clause and the block_column_name is the reference to the block.columnname that contains the data value on which to base the deletion from the specified tables.

PROCEDURE DELETE_FROM_TABLES
This procedure is called by the delete_button_press procedures to process each of the deletions. This procedure may also be called independently to delete data from a specified table. The parameters are defined as follows:

table_name_in_par = refers to the table name from which data is to be deleted.
column_name_in_par = refers to the column specified in the WHERE clause.
column_value_in_par = refers to the column value specified in the WHERE clause.

PROCEDURE TOGGLE_ENABLE_ITEM_PROPERTY
This procedure toggles the enabled property of the passed in control name (control_par) to true or falsebased on whether the control should be enabled or not. The setting_par parameter must contain a value of 'TRUE' or 'FALSE': TRUE = enabled, FALSE = disabled.

PROCEDURE ENTQUERY
This procedure is used to change various properties of the controls then change the mode of the form to QUERY mode. The TOGGLE_ENABLE_ITEM_PROPERTY procedure is called for each of the passed parameters in order to set the ENABLED property for each of them. The passed parameters are defined as follows:

control_1_par IN VARCHAR2 - passes the name of a control which is to have its ENABLED property set to FALSE (I.e. PROPERTY_OFF).
control_2_par IN VARCHAR2 - passes the name of a control which is to have its ENABLED property set to TRUE (I.e. PROPERTY_ON).
control_3_par IN VARCHAR2 - passes the name of a control which is to have its ENABLED property set to TRUE

PROCEDURE EXEQUERY
This procedure is used to change various properties of the controls then change the mode of the form to QUERY mode. The TOGGLE_ENABLE_ITEM_PROPERTY procedure is called for each of the passed parameters in order to set the ENABLED property for each of them. The passed parameters are defined as follows:

control_1_par IN VARCHAR2 - passes the name of a control which is to have its ENABLED property set to FALSE (I.e. PROPERTY_OFF).
control_2_par IN VARCHAR2 - passes the name of a control which is to have its ENABLED property set to TRUE (I.e. PROPERTY_ON).
control_3_par IN VARCHAR2 - passes the name of a control which is to have its ENABLED property set to TRUE

PROCEDURE INSERT_PROC
This procedure clears the text and check box items on the current form.

PROCEDURE RECORD_STATUS_CHECK_AND_NAV

This procedure checks the record status of the current record. If the record has a status of 'INSERT' or 'NEW' then an alert to save the current record before entering a new record is displayed. Otherwise, call the procedure corresponding to the passed parameter (navigate_par) value (I.e. 'INSERT_PROC', 'NEXT_RECORD').

PROCEDURE DISPLAY_CONTROL
This procedure alters the property settings of a control on a form (control_name_par). This procedure alters the position (position1_par, position2_par) and size (size1_par, size2_par) of a control to the position and size values provided in the parameters, changes the controls displayed
text and displays the object.

PROCEDURE DISPLAY_CONTROL_OFF
This procedure changes the display property to false for the control passed as a parameter.

PROCEDURE OPEN_FORM_BUTTON
This procedure to check if a form is already opened. If the form is opened then display it, otherwise open the form. The form name is passed as a parameter.

PROCEDURE EXITPROMPT
This procedure calls the FORM_SECURITY.check_permissions function which returns a value signifying the level of permissions granted to a user. If the returned value is greater than 2, then exit the form with out committing any data. If the value is equal to or less than 2 and the record status is 'CHANGED' then prompt the user to save their changes... if they click Yes, then execute the SAVED_PROCS_PKG.MAIN_SAVE_PROC procedure to validate the record and save the changes. If the record status is something other than 'CHANGED' and the calling form is 'BC_DATA_RETRIEVAL' or 'BC_DATA_TYPE' then exit the form with no commit and no rollback (since the changes have already been posted to the database, otherwise, exit the form.

PROCEDURE AUTO_HINT_TEXT
This procedure displays or hides the AUTO_HINT_text of the passed control name. The display_mode_par accepts a value of 'TRUE' or 'FALSE' to toggle the property setting to TRUE and FALSE in order to display or hide the AUTO_HINT text.

PROCEDURE GO_RECORD_DETAILS
This procedure accepts the passed parameter values block_name_par and item_name_par and places the cursor focus to the specified block name and column name. If the record status is 'CHANGED' then prompt the user to save the changes. If yes, call the SAVED_PROCS_PKG.main_save_proc procedure.

PROCEDURE LARGE_COMMENT_EDITOR
This procedure is used to display the comment editor for the form specified in the passed parameter editor_name_par for the column specified in the comment_column_par parameter.The changes are saved back to the column.

PROCEDURE LOV_CHOICES
This procedure displays the LOV named in the lov_name_par parameter. If a value has not been selected from the LOV, then display a message to the user to prompt them to select a valid value from the LOV.

PROCEDURE REVALIDATE_BATCH
This procedure calls the validation functions to re-validate an existing batch of data in the "normalised" Edit Tables. Begin by verifying that the user has selected a valid value (which has been passed in the batch_seq_in_par parameter). If the passed parameter is NOT NULL then proceed with the procedure processing, otherwise... display a message to the user to inform them that a valid value must first be selected before proceeding with the re-validation of the data. The call to the following functions is dependent upon the functional area in which the application is currently in. Therefore, if the functional_area_par parameter passed is 'DISCRETE' then call the following functions:

BATCH_VALIDATION_PKG.CHECK_BATCH_DISHEDR_ERRORS - Discrete Header validation,

BATCH_VALIDATION_PKG.CHECK_BATCH_DISDETAIL_ERRORS - Discrete Detail validation,
BATCH_VALIDATION_PKG.CHECK_BATCH_DISREPLIC_ERRORS - Discrete Replicate validation.

If the functional_area_par parameter passed is 'PLANKTON' then call the following functions:
BATCH_VALIDATION_PKG.CHECK_BATCH_PLANK_HEDR_ERRORS - Plankton Header validation,
BATCH_VALIDATION_PKG.CHECK_BATCH_PLANK_GENERL_ERRS - Plankton General validation.

The following functions are always called since they apply to all of the areas of functionality:
BATCH_VALIDATION_PKG.CHECK_BATCH_MISSION_ERRORS - Mission validation,
BATCH_VALIDATION_PKG.CHECK_BATCH_EVENT_ERRORS - Event validation.

PROCEDURE RUN_ERROR_SUMMARY_REPORT
This procedure runs the Validation Error Summary Report. It verifies that the user has selected a valid value (the batch_seq_par passed parameter is NOT NULL). If the batch_seq_par is NOT NULL then proceed to call the Run_Product built-in to run the 'BC_Edits_Error_Summary.rdf' report passing the value of the batch_seq_par in the parameter list (which is passed to the report). The report will display all of the validation errors for the user selected batch_seq value. The batch_column_name parameter refers to the column name that contains the user selected batch_seq value.

Programmatic Validations

Programmatic Validations (BIOCHEM Account)

At the time of this document creation, there are no validations in place to verify the data stored in the Archival tables. Please see the next section: *Programmatic Validations (Data Manager Account)* for the validation rules and the generated error codes that represent the missing or incorrect values that are verified for the Code tables. This level of validation is performed from the Data Manager's Account and carried out against the set of Code tables (only set of Code tables) that exist in the BIOCHEM Oracle Account.

Programmatic Validations (Data Manager Account)

Discrete Data Functionality

The following information is a copy of the specifications documentation (named "Discrete Station to Mission Event Header2.doc") used to create the processing to handle the loading of the data from the BCDISCRETESTATNEDITS table (in the Data Manager Oracle Account) into the BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, BCACTIVITYEDITS, and the BCDISCRETEHEDREDITS tables. The document also outlines the column level validations that are enforced. This information has been updated in this document in order to more accurately reflect the developed data validation processing as is currently developed (as of August 01, 2000).

There are very few constraints in place in the BCDISCRETESTATNEDITS table in order to load all of the data that is contained in the input ASCII file. The NOT NULL constraints in the BCDISCRETESTATNEDITS table are on the following columns:

- created_by User name of the individual who loaded the data.

- created_date System date and time of the data entry.

The first step to validating the BCDISCRETESTATNEDITS table data is to ensure that the following columns are NOT NULL:

- dis_sample_key_value user-specified pseudo primary key value
- mission_descriptor
- event_collector_event_id

The validation must loop through all of the records loaded in the BCDISCRETESTATNEDITS table where the process_flag = 'NR' and the created_date = *a date/time of data entry* (these two columns in combination will clearly denote a dataset to process) in order to identify the records that have a NULL value for any of the three columns listed above. Once validation processing begins, each record is assigned a process_flag value of 'SVI' (Station data Validation In progress).

The records that are in violation of this validation (column NULL values) will be assigned a process_flag = 'SVE' (Station data Validation Error) and the error code 4001 is to be written to the BCSTATNDATAERRORS table. The second step to ensuring that the records in the BCDISCRETESTATNEDITS table are valid is to ensure that the dis_sample_key_value column value is unique within the dataset currently being processed. If the value stored in the dis_sample_key_value(discrete. Plank_sample_key_value for plankton data) is deemed to be non-unique, then assign the process_flag = 'SVD' (Station Validation Duplicate) for the violating record and the error code 4043 is to be written to the BCSTATNDATAERRORS table. Since this key value represents a header record it is assumed that the fields the logically identify a header are distinct for every key value. If the key values are diferent but the fields that logically identify the header are the same that the dataset be flagged with a 4043 error.(Oct 11, 2000)

These types of validation errors are termed 'fatal validation errors'; no further processing is allowed to take place since the pseudo Primary Key structure is not valid. The Data Manager, who is running the validation, is alerted that errors exist in one or more records in the BCDISCRETESTATNEDITS table. A report may be generated to display the record(s) that have a process_flag value of 'SVE' and/or 'SVD'. The dataset that contains the records with process_flag = 'SVE' and/ or 'SVD' may now be deleted from the BCDISCRETESTATNEDITS table and the corresponding records may also be deleted from the BCDISCRTEDATAEDITS table (where the value of the dis_sample_key_value column from the BCDISCRTEDATAEDITS table equal the same column's value in the BCDISCRETESTATNEDITS table). This will enable a Data Manager to view the error report, fix the errors by providing data for the offending record(s) in the input ASCII file, then reload the data via SQL*Loader; thus ensuring that a 'clean' initial input ASCII file exists for archival. A Data Manager may also feel comfortable enough to edit the data using SQL*Plus in the BCDISCRETESTATNEDITS and BCDISCRTEDATAEDITS tables, update the process_flags to 'NR' for all records in the dataset and then re-process the batch of data for initial validation errors. The user is given the choice to generate a report, or simply delete the records. If the report is generated, the user is prompted whether or not they would like to delete the records.

If the dataset passes the initial validation, a batch_seq number is assigned to all of the records in the dataset in order to categorise and group the data in the BCDISCRETESTATNEDITS table to be processed as one batch or dataset (assign a job id). The sequence value that is generated by this action is then assigned to the batch_seq column in these two tables for all records that are identified as loaded on a specified date/time, and have a process_flag = 'SVI' (the BCDISCRETEDATAEDITS records must be related to a parent record from the BCDISCRETESTATNEDITS table before the assigning of a batch_seq value). The batch_name is assigned the value of the distinct batch_seq value provided in the BCDISCRETESTATNEDITS table or the value of the newly created batch_seq value if values are not provided in the initial data load.

Once a batch_seq is assigned, this value is used throughout the remaining validation process in the Edit Table System to manage data in various stages of processing. The Data Manager is also able to view a list of batch_seq values, on the BCBATCHES form, and batch names (Data Manager assigned) that are currently in the edit table system. The Data Manager will select the batch name, which represents the data to be processed and duplicate checked. The selection of an existing batch_seq value will be very useful in order to process data in more manageable datasets, especially for procedures such as Duplicate Maintenance (a procedure that performs a very detailed duplicate check of data which is to be developed in the future).

The data is now prepared enough to be loaded to the normalized Edit Table System as defined in the column mappings in the tables below (beginning with the section titled Load BCMISSIONEDITS From BCDISCRETESTATNEDITS below).

Records, of a Data Manager specified batch_seq value, must all have a process_flag value of 'SVI' to be considered for loading into the edit table system from the BCDISCRETESTATNEDITS table. A dataset with any record with a process_flag = 'SVE' and its corresponding records for the same batch_seq value must not be loaded into the edit table system, but as mentioned earlier must be deleted or edited and re-processed.

As the data is being loaded from the BCDISCRETESTATNEDITS table to the BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, and BCDISCRETEHEDREDITS tables, the process_flag of the newly entered edit table system records, must be set to 'ENR' (Edit table New Record) while the process_flag of the record left in the BCDISCRETESTATNEDITS table must remain as 'SVI' (Station Validation In progress). The records in the BCDISCRETESTATNEDITS represented by the same batch_seq value and all having a process_flag = 'SVI' may be deleted only if all corresponding BCDISCRETEDATAEDITS records, of the same batch_seq value, have also been loaded to the edit tables (to the BCDISCRETEDTAILEDITS and BCDISREPLICATEDITS tables). And are assigned a similar process_flag = 'DVI' (Data Validation In progress) to signify that the data is now loaded into the Edit tables.

BCDISCRETESTATNEDITS datasets may contain records without a related BCDISCRETEDATAEDITS record. In this case, load the record(s) from the BCDISCRETESTATNEDITS into the BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, and BCDISCRETEHEDREDITS tables then assign a process_flag = 'SVI' (in the BCDISCRETESTATNEDITS). The records pertaining to a dataset with all records having a process_flag = 'SVI' must now be deleted especially if no corresponding records exist in the BCDISCRETEDATAEDITS table. If records do exist in the BCDISCRETEDATAEDITS table, then proceed to validate those records that correspond to the dataset being processed in the BCDISCRETESTATNEDITS table.

Please Note: Any given dataset may only contain:

- 'Mission' and 'Event' data only.
- 'Mission', 'Event', and 'Header' data only.
- 'Mission', 'Event', 'Header', and 'Data' records.

This data must be loaded to the appropriate Edit System tables.

Column Validation

All error validation in the Edit Table System is to be handled in the same manner. Records that pass the programmatic record level validation are assigned a process_flag = 'ECN' (Edit table record is CleaN and error free). Once a record is found that fails a specified programmatic column level validation (such as NOT NULL constraint violation), that record is assigned a process_flag = 'ERR' (edit table ERRor). The BCERRORS table must also be populated with the following values for each column in a record in error:

BCERRORS table description:

Column Name	***Column NULL Option***	***Data Value***
error_num_seq	NOT NULL	>call sequence generator<
edit_table_name	NOT NULL	Table name where validation error occurred.
record_num_seq	NOT NULL	Primary key value of the record where the validation error occurred.
column_name	NOT NULL	Column name where the validation error occurred.
error_code	NOT NULL	Error code assigned to the validation error. Assigned by the procedure and referenced from the BCERRORCODES table.
last_updated_by	NOT NULL	Defaults to the logged in Data Manager user name that is validating the data.
last_update_date	NOT NULL	Defaults to the sysdate of the record editing.

* See Column Validation section.

Once an invalid record is edited and corrected, the process_flag = ‘ECN’.

BCMISSIONEDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETESTATNEDITS and the BCMISSIONEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETESTATNEDITS (or other)	**BCMISSIONEDITS**
MISSIONS_SEQ.nextval	mission_edt_seq
NULL	mission_seq
data_center_code	data_center_code
mission_name	name
mission_descriptor	descriptor
mission_leader	leader
mission_sdate	sdate
mission_edate	edate
mission_institute	institute
mission_platform	platform
mission_protocol	protocol
mission_geographic_region	geographic_region
mission_collector_comment1	collector_comment
mission_data_manager_comment	data_manager_comment
*See special considerations below	more_comment
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
process_flag	process_flag
batch_seq	batch_seq

* Special Considerations:

The BCMISSIONEDITS.more_comment column stores a ‘Y’ or ‘N’ to signify that a BCDISCRETESTATNEDITS.mission_collector_comment2 exists (that is to say that the BCDISCRETESTATNEDITS.mission_collector_comment2 is NOT NULL). This being the case, the value of this column (the text) is to be stored in the BCCOMMENTEDITS table and the BCMISSIONEDITS.more_comment = ‘Y’.

When populating the BCMISSIONEDITS table, the validation procedures must ensure that no record is duplicated in the table when processing a ‘batch’ of data. Therefore, if there is an existing ‘Mission’ loaded in the BCMISSIONEDITS table, any subsequent ‘Mission’ data loaded must first ensure that the ‘Mission’ data does not already exist. If the record does not exist in the BCMISSIONEDITS table, then commit the new record to the table, otherwise, do not add the record (if it is exactly the same, that is to say that all column data for the specified ‘Mission’ record are the same).

- The first level of duplicate checking is to verify that the mission_descriptor value is different; if it is, then load the ‘Mission level data into the BCMISSIONEDITS table. If the mission_descriptor value is the same, then do not load the new ‘Mission’ level record.

- When a duplicate ‘Mission’ is found use the mission_edt_seq value of the existing ‘Mission’ level record and assign its value as the foreign key to the BCEVENTEDITS and/or the BCCOMMENTEDITS tables (if data exists in these two tables that is related to the Mission). It remains very important to preserve the PK/FK relationship of the data, therefore the linking of an existing ‘Mission’ record or newly entered ‘Mission’ record must be preserved to link to either the BCCOMMENTEDITS or the BCEVENTEDITS table records

Column Level Validations for the BCMISSIONEDITS table

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
mission_edt_seq	System generated sequence (unique value) created in the using the archival sequence. No validation required.	
mission_seq	No validation.	
data_center_code	Check that the value exists in the BCDATACENTERS table. Ensure that this column is NOT NULL.	4001 – NOT NULL error 4002 – Invalid Data Center Code
name	Free format. No validation required.	
descriptor	Ensure that this column is NOT NULL.	4001 – NOT NULL error
leader	Free format. No validation required.	
sdate	Check that the date is a valid date (with a 4-digit year representation) and that the date is not in the future, and that sdate must be <= edate.	4003 – Invalid date representation 4004 – Date is in the future. 4005 - sdate must be <= edate.
edate	Check that the date is a valid date (with a 4 digit year representation), that the date is not in the future, and that edate must be >= sdate.	4003 – Invalid date representation 4004 – Date is in the future. 4005 - sdate must be <= edate.
institute	Free format. No validation required.	
platform	Free format. No validation required.	
protocol	Free format. No validation required.	
geographic_region	Free format. No validation required.	
collector_comment	Free format. No validation required.	
data_manager_comment	Free format. No validation required.	
more_comment	Flag of Y or N to signify more comments exist in the BCCOMMENTS table. Only allow Y or N characters in this column.	4007 – Invalid data entered.
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	

last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required. The initial value is set to the same value as the created_by column.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required. The initial value is set to the same value as the created_date column.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	

EVENTEDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETESTATNEDITS and the BCEVENTEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETESTATNEDITS (or other)	**BCEVENTEDITS**
BCEVENTS_SEQ.nextval	event_edt_seq
NULL	event_seq
data_center_code	data_center_code
NULL	mission_seq
event_sdate	Sdate
event_edate	Edate
event_stime	Stime
event_etime	Etime
event_min_lat	min_lat
event_max_lat	max_lat
event_min_lon	min_lon
event_max_lon	max_lon
event_collector_stn_name	collector_station_name
event_collector_event_id	collector_event_id
event_UTC_offset	UTC_offset
event_collector_comment1	collector_comment
event_data_manager_comment	data_manager_comment
*See special considerations below	more_comment
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
process_flag	process_flag
batch_seq	batch_seq
BCMISSIONEDITS.mission_edt_seq as a foreign key to the PK record.	mission_edt_seq

*Special Considerations:

The BCEVENTEDITS.more_comment column stores a 'Y' or 'N' to signify that a BCDISCRETESTATNEDITS.event_collector_comment2 exists (that is to say that the BCDISCRETESTATNEDITS.event_collector_comment2 is NOT NULL). This being the case, the value of this column is to be stored in the BCCOMMENTEDITS table and the BCEVENTEDITS.more_comment = 'Y'.

When populating the BCEVENTEDITS table, the validation procedures must ensure that no record is duplicated in the table when processing a 'batch' of data. Therefore, if there is an existing 'Event' loaded in the BCEVENTEDITS table, any subsequent 'Event' data loaded must first ensure that the 'Event' data does not already exist. If the record does not exist in the BCEVENTEDITS table, then commit the new record to the table, otherwise, do not add the record (if it is exactly the same, that is to say that all column data for the specified 'Event' record are the same).

- The first level of duplicate checking is to verify that the collector_station_name and the collector_event_id column values are different; if they are, then load the 'Event' level data into the BCEVENTEDITS table. If the collector_station_name and the collector_event_id values are the same, then do not load the new 'Event' level record.

- When a duplicate 'Event' is found use the event_edt_seq value of the existing 'Event' level record and assign its value as the foreign key to the BCACTIVITYEDITS and/or the BCCOMMENTEDITS tables (if data exists in these two tables that is related to the 'Event'). It remains very important to preserve the PK/FK relationship of the data, therefore the linking of an existing 'Event' record or newly entered 'Event' record must be preserved to link to either the BCACTIVITYEDITS or the BCCOMMENTEDITS table records.

Column Level Validations for the BCEVENTEDITS table

Please note 1: when performing the stime/etime column validations, it is important to remember that an 'Event' stime may be on one date and the etime may occur on another date, therefore the validation for etime must be >= stime may not be true. Example: 'Event' stime = 2300 on May 1, 1999 and the 'Event' etime = 0230 on May 2, 1999. In this case, the etime must be >= stime" is not true on its own without including the sdate and edate values to ensure that etime must be >= stime.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
event_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
event_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error 4002 – Invalid Data Center Code
mission_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
sdate	Check that the date is a valid date (with a 4-digit year representation) and that the date is not in the future.	4001 – NOT NULL error 4003 – Invalid date representation 4004 – Date is in the future. 4005 - sdate must be <= edate.
edate	Check that the date is a valid date (with a 4 digit year representation), that the date is not in the future, and that edate must be >= sdate.	4003 – Invalid date representation 4004 – Date is in the future. 4005 - edate must be >= sdate.
stime	Check that the time is a valid time (using 4 digits to represent it), that the time is between 0000 and 2359. Start time must be <= End time.	4010 – Invalid time. 4011 – Start time not before end time.

etime	Check that the time is a valid time (using 4 digits to represent it), that the time is between 0000 and 2359, and that etime must be >= stime.	4010 – Invalid time. 4012 – End time not after start time.
min_lat	Ensure that latitude range is -90.00 to 90.00. Min_lat must be <= max_lat. Column must be NOT NULL.	4001 – NOT NULL error 4013 – Invalid latitude range. 4014 - Invalid latitude value.
max_lat	Ensure that latitude range is -90.00 to 90.00. Max_lat must be >= min_lat.	4013 – Invalid latitude range. 4014 - Invalid latitude value.
min_lon	Ensure that longitude range is -180.00 to 180.00. Min_lon must be <= max_lon. Column must be NOT NULL.	4001 – NOT NULL error 4015 – Invalid longitude range. 4016 - Invalid longitude value.
max_lon	Ensure that longitude range is -180.00 to 180.00. Min_lon must be >= max_lon.	4015 – Invalid longitude range. 4016 - Invalid longitude value.
collector_station_name	Free format. No validation required.	
collector_event_id	Column must be NOT NULL.	4001 – NOT NULL error
UTC_offset	Value must be between -12 and 12 with no more than 1 decimal place. Column may be NULL.	4017 - Invalid UTC offset value.
collector_comment	Free format. No validation required.	
data_manager_comment	Free format. No validation required.	
more_comment	Flag of Y or N to signify more comments exist in the BCCOMMENTS table. Only allow Y or N characters in this column.	4007 – Invalid data entered.
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
mission_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCMISSIONEDITS table. No validation required.	

BCCOMMENTEDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETESTATNEDITS and the BCCOMMENTEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETESTATNEDITS	BCCOMMENTEDITS
BCCOMMENTS_SEQ.nextval	comment_edt_seq
NULL	comment_seq
data_center_code	data_center_code
NULL	event_seq
NULL	mission_seq
mission_collector_comment2 and/or event_collector_comment2	edit_comment
*Ordinal number generator	comment_num
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
process_flag	process_flag
batch_seq	batch_seq
BCMISSIONEDITS.mission_edt_seq as a foreign key to the PK record.	mission_edt_seq
BCEVENTEDITS.event_edt_seq as a foreign key to the PK record.	event_edt_seq

*Ordinal number generator:

- This table stores comments which are a continuation of comments in the BCMISSIONEDITS and the BCEVENTEDITS tables, since each of these tables may only store comments up to 2000 characters. Once a comment is longer than 2000 characters, that portion of the comment is written to the BCCOMMENTEDITS table. If a comment in the BCCOMMENTEDITS table is longer than 2000 characters, that comment may be continued in a new record in the BCCOMMENTEDITS table, referencing the parent table (BCMISSIONEDITS or BCEVENTEDITS) and numbered with an 'ordinal value' (internal table sequence) which will keep track of the order of the comments. The ordinal value will default to a value of 1 (one) for each new record which is a continuation from the comment columns in either the BCMISSIONEDITS or BCEVENTEDITS tables. The value of the ordinal will be incremented by one for each comment, which is a continuation of a comment within the BCCOMMENTEDITS table.

Column Level Validations for the BCCOMMENTEDITS table

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
comment_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
comment_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error 4002 – Invalid Data Center Code
event_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
mission_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
edit_comment	Free format. No validation required. Column must be NOT NULL.	4001 – NOT NULL error

comment_num	System generated ordinal value to number the comments in sequence in order to categorize comments together. The comment_num value will reset to 1 for each new mission_seq or event_seq value (and increment by one for each comment, in a set of comments). Column must be NOT NULL.	4001 – NOT NULL error
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
mission_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCMISSIONEDITS table. No validation required.	
event_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCEVENTEDITS table. No validation required.	

BCACTIVITYEDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETESTATNEDITS and the BCACTIVITYEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETESTATNEDITS (or other)	**BCACTIVITYEDITS**
BCACTIVITIES_SEQ.nextval	activity_edt_seq
NULL	activity_seq
NULL	event_seq
data_center_code	data_center_code
'DH'	data_pointer_code
'ECN'	process_flag
batch_seq	batch_seq
BCEVENTEDITS.event_edt_seq as a foreign key to the PK record.	event_edt_seq

BCDISCRETEHEDREDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETESTATNEDITS and the BCDISCRETEHEDREDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETESTATNEDITS (or other)	**BCDISCRETEHEDREDITS**
BCDISCRETEHEDRS_SEQ.nextval	dis_headr_edt_seq
NULL	discrete_seq
data_center_code	data_center_code
NULL	event_seq
NULL	activity_seq
dis_headr_gear_seq	gear_seq
dis_headr_sdate	sdate
dis_headr_stime	stime
dis_headr_time_qc_code	time_qc_code
dis_headr_slat	slat
dis_headr_slon	slon
dis_headr_position_qc_code	position_qc_code
dis_headr_start_depth	start_depth
dis_headr_end_depth	end_depth
dis_headr_sounding	sounding
dis_headr_collector_deplmt_id	collector_deployment_id
dis_headr_collector_sample_id	collector_sample_id
dis_headr_collector	collector
dis_headr_collector_comment1	collector_comment
dis_headr_data_manager_comment	data_manager_comment
dis_headr_responsible_group	responsible_group
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
process_flag	process_flag
batch_seq	batch_seq
BCACTIVITYEDITS.activity_edt_seq as a foreign key to the PK record.	activity_edt_seq
BCEVENTEDITS.event_edt_seq as a foreign key to the PK record.	event_edt_seq

- When populating the BCDISCRETEHEDREDITS table, the validation procedures must ensure that no record is duplicated in the table when processing a ‘batch’ of data. Therefore, if there is an existing ‘Header’ loaded in the BCDISCRETEHEDREDITS table, any subsequent ‘Header’ data loaded must first ensure that the ‘Header’ data does not already exist. If the record does not exist in the BCDISCRETEHEDREDITS table, then commit the new record to the table, otherwise, do not add the record (if it is exactly the same, that is to say that all column data for the specified ‘Header’ record are the same).

    - The first level of duplicate checking is to verify that the collector_sample_id column values are different; if they are, then load the ‘Header’ level data into the BCDISCRETEHEDREDITS table.

- When a duplicate 'Header' is found use the dis_headr_edt_seq value of the existing 'Header' level record and assign its value as the foreign key to the BCDISCRETEHEDREDITS table (for detail data records). It remains very important to preserve the PK/FK relationship of the data; therefore the linking of an existing 'Header' record or newly entered 'Header' record must be preserved to link to the BCDISCRETEHEDREDITS table records.

Column Level Validations for the BCDISCRETEHEDREDITS table

Please Note: The Validation of the stime, etime, slat, slon, elat, and elon listed below do not take the assigned quality code value for the time_qc_code and position_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
dis_headr_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
discrete_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Check that the value exists in the BCDATACENTERS table. Column must be NOT NULL.	4001 – NOT NULL error 4002 – Invalid Data Center Code
event_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
activity_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
gear_seq	Check that the value exists in the BCGEARS table. Column must be NOT NULL.	4001 – NOT NULL error 4022 - Invalid gear sequence value.
Sdate	Check that the date is a valid date (with a 4-digit year representation) and that the date is not in the future. Column must be NOT NULL.	4001 – NOT NULL error 4003 – Invalid date representation 4004 – Date is in the future.
Stime	Check that the time is a valid time (using 4 digits to represent it), that the time is between 0000 and 2359.	4010 – Invalid time. 4011 – Start time not before end time.
etime	Check that the time is a valid time (using 4 digits to represent it), that the time is between 0000 and 2359.	4010 – Invalid time. 4012 – End time not after start time.
time_qc_code	Check that the value exists in the BCQUALCODES table.	4001 – NOT NULL error 4018 – Invalid Quality Code
slat	Ensure that latitude range is -90.00 to 90.00. Column must be NOT NULL.	4001 – NOT NULL error 4013 – Invalid latitude range.
elat	Ensure that latitude range is -90.00 to 90.00.	4013 – Invalid latitude range.
slon	Ensure that longitude range is -180.00 to 180.00. Column must be NOT NULL.	4001 – NOT NULL error 4015 – Invalid longitude range.
elon	Ensure that longitude range is -180.00 to 180.00.	4015 – Invalid longitude range.
position_qc_code	Check that the value exists in the BCQUALCODES table.	4001 – NOT NULL error 4018 – Invalid Quality Code

start_depth	Ensure that value is between 0 and 12000. Column must be NOT NULL.	4001 – NOT NULL error 4019 – Invalid depth range.
end_depth	Ensure that value is between 0 and 12000. Column must be NOT NULL.	4001 – NOT NULL error 4019 – Invalid depth range.
sounding	Ensure that value is between 0 and 12000. Column may be NULL.	4019 – Invalid depth range.
collector_deployment_id	Free format. No validation required.	
collector_sample_id	Column must be NOT NULL.	4001 – NOT NULL error
collector	Free format. No validation required.	
collector_comment	Free format. No validation required.	
data_manager_comment	Free format. No validation required.	
responsible_group	List of values representing the various DFO sectors that will store data in the BIO-CHEM database system. No validation required.	
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
activity_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCACTIVITYEDITS table. No validation required.	
event_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCEVENTEDITS table. No validation required.	

The following information is a copy of the specifications documentation (named "Convert Discrete DATA to Detail and Replicate.doc") used to create the processing to handle the loading of the data from the BCDISCRETEDATAEDITS table (in the Data Manager Oracle Account) into the BCDISCRETEDTAILEDITS and the BCDISREPLICATEDITS tables. The document also outlines the column level validations that are enforced. This information has been updated in this document in order to more accurately reflect the developed data validation processing as is currently developed.

The following spec pertains to the loading of data from the BCDISCRETEDATAEDITS table to the BCDISCRETEDTAILEDITS and the BCDISREPLICATEDITS tables.

All data records that are entered into the BCDISCRETEDATAEDITS table have a created_date column value of the date/timestamp of the data load (date and time). All data loaded at the same time will be assigned the

same date/timestamp value. All records are also assigned a process_flag of ‘NR’ to signify a new record in the table.

All data must be managed by the dis_sample_key_value (pseudo Foreign Key). The data in the dataset, which contains ‘fatal errors’, will be deleted based on the programmatically imposed PK/FK relationship from the BCDISCRETESTATNEDITS table record to the BCDISCRETEDATAEDITS table record(s). If a BCDISCRETESTATNEDITS table record is deemed to contain a fatal error, then those records (that caused the fatal error) may be printed in a report. The Data Manager is notified of the fatal errors in the processing of the ‘Station’ record(s) then the dataset that contains the fatal error may be deleted from the ‘Station’ table. Likewise, all corresponding records that exist in the ‘Data’ table may also be deleted once they too have been assessed for fatal errors. The fatal errors report should encompass both the ‘Station’ and Data’ records for a particular dataset to report the records that contain the fatal errors. Therefore the ‘Station’ records and the corresponding ‘Data’ records are validated for fatal errors before an error report is created or a dataset is deleted (deletion of PK values from the ‘Station’ table and the corresponding FK value from the ‘Data’ table).

Fatal Errors are those that consist of the following types of validations which result in error:

BCDISCRETEDATAEDITS		BCDISCRETESTATNEDITS
mission_descriptor	=	mission_descriptor
event_collector_event_id	=	event_collector_event_id
event_collector_stn_name	=	event_collector_stn_name
dis_header_start_depth	=	dis_header_start_depth
dis_header_end_depth	=	dis_headr_end_depth
dis_header_slat	=	dis_headr_slat
dis_header_slon	=	dis_headr_slon
dis_header_sdate	=	dis_headr_sdate
dis_header_stime	=	dis_headr_stime
dis_detail_collector_samp_id	=	dis_headr_collector_sample_id
dis_sample_key_value	=	dis_sample_key_value
data_center_code	=	data_center_code

If any of the above column validations fail, insert a record into the BCSTATNDATAERRORS table with the error_code value = 4042.

Another fatal error is one generated when a record contains a value for both the BCDISCRETEDATAEDITS.data_type_method and the BCDISCRETEDATAEDITS.dis_detail_data_type_seq. The data_type_method value must be checked in the BCDATATYPES table to ensure that the dis_detail_data_type_seq value is the true corresponding value. If the two values do not match, then assign the process_flag = ‘DVE’ and insert a record into the BCSTATNDATAERRORS table with the error_code value = 4051.

If the BCDISCRETEDATAEDITS.dis_detail_data_type_seq is NULL but the BCDISCRETEDATAEDITS.data_type_method contains a value, then reference the BCDATATYPES table to verify that the datatype exists in the table. If the datatype exists in the table, then assign the BCDISCRETEDATAEDITS.dis_detail_data_type_seq the value associated with the data_type_method. If the datatype does not exist, then Flag it as an Null error 4003 so that processing does not continue on the batch until the value is assigned a valid data type.

Otherwise if the BCDISCRETEDATAEDITS.dis_detail_data_type_seq is NOT NULL and the BCDISCRETEDATAEDITS.data_type_method is NULL, then do nothing since no true test condition exists.

All of the validation conditions listed in the previous table will generate a fatal error if the validation fails. When one of these validations fail, the process_flag column must be assigned a value of ‘DVE’ (Data Validation Error). This will mark the record that is to be included in a report to the Data Manager to notify them of any fatal errors in the BCDISCRETEDATAEDITS table dataset. As the records are processed, assign the process_flag column a value of ‘DVI’ (Data Validation In progress) to denote that the record passed this level of validation. If any "fatal validation errors" exist, once all records in the BCDISCRETEDATAEDITS dataset are processed (by

dis_sample_key_value reference from the ‘Station’ table), notify the Data Manager via a message, the user may then generate an error report to display all of the fatal errors, and delete the dataset that contains the errors. The Data Manager must then fix all of the errors in the originating ASCII file, then reload the data.

The data loaded into this table via SQL*Loader must reference a corresponding record in the BCDISCRETESTATNEDITS table via the dis_sample_key_value PK/FK relationship. If there is no record in the BCDISCRETESTATNEDITS table for a record in the BCDISCRETEDATAEDITS table (creating an orphan data record), then the record must be marked as invalid by assigning a process_flag = ‘DVO’ (Data Validation Orphan record) and insert a record into the BCSTATNDATAERRORS table with the error_code value = 4052.

Even if all ’Station’ data is successfully validated, the ‘Data’ table may contain an invalid record marked as a fatal error that will cause the full dataset to be marked for deletion (records from both ‘Station’ and ‘Data’). Therefore no records are deleted from the ‘Station’ or ‘Data’ tables until all records in a dataset from the BCDISCRETESTATNEDITS table and the corresponding records from the BCDISCRETEDATAEDITS are processed and assigned a process_flag of ‘SVE’, or ‘SVD’ (for the ‘Station’ records), or ‘DVE’, or ‘DVO’ (for the ‘Data’ records).

As stated above, once the data record passes the above listed validations, the process_flag is assigned a value of ‘DVI’ (Data Validation In progress). If a datasets process_flag column values are all set to ‘DVI’, then the records are deemed as passing the first validation layer (initial validation) and are prepared to be loaded into the Edit Table System.

The tables below outline the mapping of the data from the BCDISCRETEDATAEDITS table to each of the BCDISCRETEDTAILEDITS and the BCDISREPLICATEDITS tables. All data, in a dataset containing all records having a process_flag = ‘DVI’, are to be loaded from the BCDISCRETEDATAEDITS table to the BCDISCRETEDTAILEDITS or BCDISREPLICATEDITS tables.

As the records are loaded into the BCDISCRETEDTAILEDITS and BCDISREPLICATEDITS tables, the originating data record in the BCDISCRETEDATAEDITS table has a process_flag = ‘DVI’ (Data Validation In progress) and duplicate checking must occur to identify replicate data (more than one measurement with the same datatype_seq and collector_sample_id column value). Duplicate ‘Detail’ records must not exist in the BCDISCRETEDTAILEDITS for any one BCDISCRETEHEDREDITS (‘Header’) record, therefore once a record is identified as being a duplicate, it is written to the BCDISREPLICATEDITS table along with the record that was in the BCDISCRETEDTAILEDITS table, that was identified as being the record that was compared with. The records are then averaged, the averaged value is written to the BCDISCRETEDTAILEDITS.data_value column and the BCDISCRETEDTAILEDITS.averaged_data column is assigned a value of ‘Y’ (to signify that the value in the data_value column is an averaged value and that the values that make up the average are stored in the BCDISREPLICATEDITS table referenced by the pseudo Foreign Key column dis_detail_edt_seq. Assign the data_qc_code column the value of the max(BCDISREPLICATEDITS.data_qc_code) or take the value of 0 if any of the replicates have value of 0 for the data_qc_code column of the replicates with the same datatype_seq and collector_sample_id values that reference the same dis_detail_edt_seq value. Also take the highest detection_limit value for the records which are being compared, or take the value of NULL for the detection_limit column if any of the records being compared contain a NULL value for the detection_limit.

BCDISCRETEDTAILEDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETEDATAEDITS and the BCDISCRETEDTAILEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETEDATAEDITS (or other)	**BCDISCRETEDTAILEDITS**
BCDISCRETEDTAILS_SEQ.nextval	dis_detail_edt_seq
NULL	discrete_detail_seq

data_center_code	data_center_code
dis_detail_data_type_seq	data_type_seq
NULL	discrete_seq
dis_detail_data_value OR (averaged data value of all associated replicates if replicates are present)	data_value
NULL	data_flag
'Y' (to signify that the data_value is averaged), 'N' (to signify that the data_value is not averaged. The default value is 'N'.	averaged_data
dis_detail_data_qc_code	data_qc_code
NULL	qc_flag
dis_detail_detection_limit	detection_limit
dis_detail_detail_collector	detail_collector
dis_detail_collector_samp_id	collector_sample_id
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
'ENR'	process_flag
batch_seq	batch_seq
BCDISCRETEHEDREDITS.dis_headr_edt_seq	dis_headr_edt_seq

Column Level Validations for the BCDISCRETEDTAILEDITS table

Please Note: The Validation of the data_value column listed below does not take the assigned quality code value for the data_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	COLUMN VALIDATION	ERROR CODES
dis_detail_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
discrete_detail_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Verify that the column is NOT NULL. Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error 4002 - Invalid Data Center Code.
data_type_seq	Verify that the column is NOT NULL. Check that the value exists in the BCDATATYPES table.	4001 – NOT NULL error 4023 - Invalid Data Type value.
discrete_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	

data_value	Verify that the column is NOT NULL. Ensure that the data_value result falls between the ranges set for a specified datatype by the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_value columns for the specified datatype (If the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_values exist in the table). Correct Replicate data_value(s) for the specified collector_sample_id.	4001 – NOT NULL error 4025 - Invalid data_value result. OR If the averaged_data column is “Y” THEN 4039 - Invalid averaged data_value. *(See below).
data_flag	To be used by the Duplicate Maintenance Procedure (to be developed) No validation required.	
averaged_data	Verify that the column is NOT NULL.	4001 – NOT NULL error 4024 - Invalid averaged_data value.
data_qc_code	Verify that the column is NOT NULL. Check that the value exists in the BCQUALCODES table. Correct Replicate data_qc_code(s) for the specified collector_sample_id.	4001 – NOT NULL error 4018 - Invalid Quality Code value. OR If the averaged_data column is “Y” THEN 4040 - Invalid averaged data_qc_code. *(See below).
qc_flag	To be used by the Duplicate Maintenance Procedure (to be developed) No validation required.	
detection_limit	Free format. No validation required.	
detail_collector	Free format. No validation required.	
collector_sample_id	Column must be NOT NULL.	4001 – NOT NULL error
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
dis_headr_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCDISCRETEHEDREDITS table. No validation required.	

* For the error listed as 4039:

The validation procedure must check the value of the averaged_data column. If the value of the averaged_data column is “Y”, then the error message must be 4039 rather than 4025. This will alert the Data Manager to the fact that the data_value is an averaged one and that one of the replicate data_value(s) that make up the averaged data_value is potentially in error.

* For the error listed as 4040:

The validation procedure must check the value of the averaged_data column. If the value of the averaged_data column is “Y”, then the error message must be 4040 rather than 4018. This will alert the Data Manager to the fact that the data_qc_code is based on the highest qc_code value of the averaged data_value from the replicate detail table.

BCDISREPLICATEDITS Column Mapping

The table in this section displays the column level mapping between the BCDISCRETEDATAEDITS and the BCDISRELICATEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_DISCRETE_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCDISCRETESTATNEDITS and BCDISCRETEDATAEDITS data manager tables.

BCDISCRETEDATAEDITS (or other)	**BCDISREPLICATEDITS**
BCDISCRETEREPLICATES_SEQ.nextval	dis_repl_edt_seq
NULL	discrete_replicate_seq
data_center_code	data_center_code
NULL	discrete_detail_seq
dis_detail_data_type_seq	data_type_seq
dis_detail_data_value	data_value
dis_detail_data_qc_code	data_qc_code
dis_detail_detection_limit	detection_limit
dis_detail_detail_collector	detail_collector
dis_detail_collector_samp_id	collector_sample_id
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
‘ENR’	process_flag
batch_seq	batch_seq
BCDISCRETEDTAILEDITS.dis_headr_edt_seq	dis_detail_edt_seq

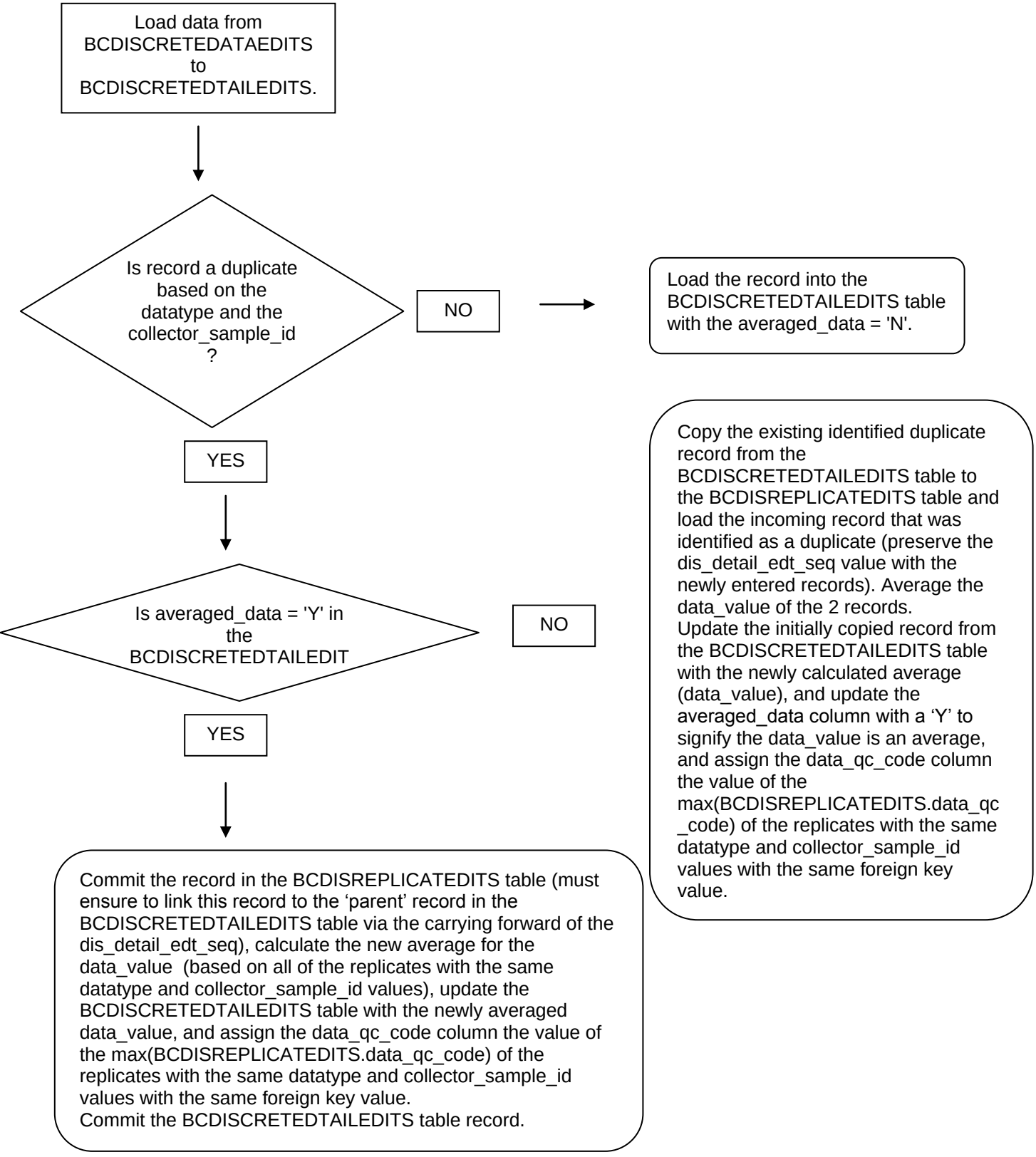
Column Level Validations for the BCDISREPLICATEDITS table

Please Note: The Validation of the data_value column listed below does not take the assigned quality code value for the data_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	**COLUMN VALIDATION**	**ERROR CODES**
dis_repl_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	

discrete_replicate_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Verify that the column is NOT NULL. Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error 4002 - Invalid Data Center Code.
discrete_detail_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_type_seq	Verify that the column is NOT NULL. Check that the value exists in the BCDATATYPES table.	4001 – NOT NULL error 4023 - Invalid Data Type value.
data_value	Verify that the column is NOT NULL. Ensure that the data_value result falls between the ranges set for a specified datatype by the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_value columns for the specified datatype (If the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_values exist in the table).	4001 – NOT NULL error 4025 - Invalid data_value result.
data_qc_code	Verify that the column is NOT NULL Check that the value exists in the BCQUALCODES table.	4001 – NOT NULL error 4018 - Invalid Quality Code value.
detection_limit	Free format. No validation required.	
detail_collector	Free format. No validation required.	
collector_sample_id	Free format. Column must be NOT NULL.	4001 – NOT NULL error
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
dis_detail_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCDISCRETEDTAILEDITS table. No validation required.	

The following diagram outlines the decisions and processing associated with replicate processing.



Plankton Data Functionality

The following information is a copy of the specifications documentation (named "Convert Plankton Station to Mission Event Header.doc") used to create the processing to handle the loading of the data from the BCPLANKTONSTATNEDITS table (in the Data Manager Oracle Account) into the BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, BCACTIVITYEDITS, and the BCPLANKTNHEDREDITS tables. The document also outlines the column level validations that are enforced. This information has been updated in this document in order to more accurately reflect the developed data validation processing as is currently developed (as of August 01, 2000).

Data is loaded into the BCPLANKTONSTATNEDITS and the BCPLANKTONDATAEDITS tables via SQL*Loader.

Once a Data Manager logs in to the BIO-CHEM application system and chooses to process data in the edit system (via the selection from the Edit Switchboard to process data from the 'Station'/'Data' tables), they will be alerted if data exists in the BCPLANKTONSTATNEDITS table that has not yet been validated and presented with a list of created_date values (containing the date/timestamp of the dataset data entry) which represents un-processed datasets. The Data Manager must now select a dataset they wish to process by selecting one of the created_date values from the list. The un-validated data (datasets) may have a process_flag = 'NR' or a process_flag = NULL and the created_date = *(the date and time of the load of data from ASCII into the BCPLANKTONSTATNEDITS table, created by SQL*Loader)*. Once the Data Manager selects a dataset to process (validate) by selecting one of the created_date values from the list. The application processing must then verify if the process_flag = NULL, then loop through the records in the BCPLANKTONSTATNEDITS table, based on the created_date selected (dataset), to assign process_flag = 'NR' (new record). The data is now identified (flagged) as ready to process.

There are very few constraints in place in the BCPLANKTONSTATNEDITS table in order to load all of the data that is contained in the input ASCII file. The NOT NULL constraints in the BCPLANKTONSTATNEDITS table are on the following columns:

- created_by User name of the individual who loaded the data.
- created_date System date and time of the data entry.

The first step to validating the BCPLANKTONSTATNEDITS table data is to ensure that the following columns are NOT NULL:

- plank_sample_key_value user-specified pseudo primary key value.
- mission_descriptor
- event_collector_event_id

The validation must loop through all of the records loaded in the BCPLANKTONSTATNEDITS table where the process_flag = 'NR' and the created_date = *a date/time of data entry* (these two columns in combination will clearly denote a dataset to process) in order to identify the records that have a NULL value for any of the three columns listed above. Once validation processing begins, each record is assigned a process_flag value of 'SVI' (Station data Validation In progress).

The records that are in violation of this validation (column NULL values) will be assigned a process_flag = 'SVE' (Station data Validation Error) and the error code 4001 is to be written to the BCSTATNDATAERRORS table. The second step to ensuring that the records in the BCPLANKTONSTATNEDITS table are valid is to ensure that the plank_sample_key_value column value is unique within the dataset currently being processed. If the value stored in the plank_sample_key_value is deemed to be non-unique, then assign the process_flag = 'SVD' (Station Validation Duplicate) for the violating record and the error code 4043 is to be written to the BCSTATNDATAERRORS table.

These types of validation errors are termed ‘fatal validation errors’; no further processing is allowed to take place since the pseudo Primary Key structure is not valid. The Data Manager, who is running the validation, is alerted that errors exist in one or more records in the BCPLANKTONSTATNEDITS table. A report may be generated to display the record(s) that have a process_flag value of ‘SVE’ or ‘SVD’. The dataset that contains the records with process_flag = ‘SVE’ and/or ‘SVD’ may now be deleted from the BCPLANKTONSTATNEDITS table and the corresponding records may also be deleted from the BCPLANKTONDATAEDITS table (where the value of the plank_sample_key_value column from the BCPLANKTONDATAEDITS table equal the same column’s value in the BCPLANKTONSTATNEDITS table). This will enable a Data Manager to view the error report, fix the errors by providing data for the offending record(s) in the input ASCII file, then reload the data via SQL*Loader; thus ensuring that a ‘clean’ initial input ASCII file exists for archival. A Data Manager may also feel comfortable enough to edit the data using SQL*Plus in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS tables, update the process_flags to 'NR' for all records in the data set and then re-process the batch of data for initial validation errors. The user is given the choice to generate a report, or simply delete the records. If the report is generated, the user is prompted whether or not they would like to delete the records.

If the dataset passes the initial validation, a batch_seq number is assigned to all of the records in the dataset in order to categorise and group the data in the BCPLANKTONSTATNEDITS table to be processed as one batch or dataset (assign a job id). The sequence value that is generated by this action is then assigned to the batch_seq column in these two tables for all records that are identified as loaded on a specified date/time, and have a process_flag = ‘SVI’ (the BCPLANKTONDATAEDITS records must be related to a parent record from the BCPLANKTONSTATNEDITS table before the assigning of a batch_seq value).

Once a batch_seq is assigned, this value is used throughout the remaining validation process in the Edit Table System to manage data in various stages of processing. The Data Manager is also able to view a list of batch_seq values, on the BCBATCHES form, and batch names (Data Manager assigned) that are currently in the edit table system. The Data Manager will select the batch name, which represents the data to be processed and duplicate checked. The selection of an existing batch_seq value will be very useful in order to process data in more manageable datasets, especially for procedures such as Duplicate Maintenance (a procedure that performs a very detailed duplicate check of data, which is to be developed in the future).

The data is now prepared enough to be loaded to the normalized Edit Table System as defined in the column mappings in the tables below (beginning with the section named Load BCMISSIONEDITS From BCPLANKTONSTATNEDITS below).

Records, of a Data Manager specified batch_seq value, must all have a process_flag value of ‘SVI’ to be considered for loading into the edit table system from the BCPLANKTONSTATNEDITS table. A dataset with any record with a process_flag = ‘SVE’ and its corresponding records for the same batch_seq value must not be loaded into the edit table system, but as mentioned earlier must be deleted or edited and re-processed.

As the data is being loaded from the BCPLANKTONSTATNEDITS table to the BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, and BCPLANKTNHEDREDITS tables, the process_flag of the newly entered edit table system records, must be set to ‘ENR’ (Edit table New Record) while the process_flag of the record left in the BCPLANKTONSTATNEDITS table must remain as ‘SVI’ (Station Validation In progress). The records in the BCPLANKTONSTATNEDITS represented by the same batch_seq value and all having a process_flag = ‘SVI’ may be deleted only if all corresponding BCPLANKTONDATAEDITS records, of the same batch_seq value, have also been loaded to the edit tables (to the BCPLANKTNGENERLEDITS tables). And are assigned a similar process_flag = ‘DVI’ (Data Validation In progress) to signify that the data is now loaded into the Edit tables.

BCPLANKTONSTATNEDITS datasets may contain records without a related BCPLANKTONDATAEDITS record. In this case, load the record(s) from the BCPLANKTONSTATNEDITS into the BCMISSIONEDITS, BCEVENTEDITS, BCCOMMENTEDITS, and BCPLANKTNHEDREDITS tables then assign a process_flag = ‘SVI’ (in the BCPLANKTONSTATNEDITS). The records pertaining to a dataset with all records having a process_flag = ‘SVI’ must now be deleted especially if no corresponding records exist in the BCPLANKTONDATAEDITS table. If records do exist in the BCPLANKTONDATAEDITS table, then proceed to validate those records that correspond to the dataset being processed in the BCPLANKTONSTATNEDITS table.

Please Note: Any given dataset may only contain:

- ‘Mission’ and ‘Event’ data only.
- ‘Mission’, ‘Event’, and ‘Header’ data only.
- ‘Mission’, ‘Event’, ‘Header’, and ‘Data’ records.

* This data must be loaded to the appropriate Edit System tables.

Column Validation

All error validation in the Edit Table System is to be handled in the same manner. Records that pass the programmatic record level validation are assigned a process_flag = ‘ECN’ (Edit table record is CleaN and error free). Once a record is found that fails a specified programmatic column level validation (such as NOT NULL constraint violation), that record is assigned a process_flag = ‘ERR’ (edit table ERRor). The BCERRORS table must also be populated with the following values for each column in a record in error:

BCERRORS table description:

Column Name	Column NULL Option	Data Value
error_num_seq	NOT NULL	>call sequence generator<
edit_table_name	NOT NULL	Table name where validation error occurred.
record_num_seq	NOT NULL	Primary key value of the record where the validation error occurred.
column_name	NOT NULL	Column name where the validation error occurred.
error_code	NOT NULL	Error code assigned to the validation error. Assigned by the procedure and referenced from the BCERRORCODES table.
last_updated_by	NOT NULL	Defaults to the logged in Data Manager user name that is validating the data.
last_update_date	NOT NULL	Defaults to the sysdate of the record editing.

* See Column Validation section.

Once an invalid record is edited and corrected, the process_flag = ‘ECN’.

BCMISSIONEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONSTATNEDITS and the BCMISSIONEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONSTATNEDITS (OR OTHER)	BCMISSIONEDITS
BCMISSIONS_SEQ.nextval	Mission_edt_seq
NULL	Mission_seq
data_center_code	Data_center_code
mission_name	Name
mission_descriptor	Descriptor
mission_leader	Leader
mission_sdate	Sdate
mission_edate	Edate
mission_institute	Institute
mission_platform	Platform
mission_protocol	Protocol
mission_geographic_region	Geographic_region
mission_collector_comment1	Collector_comment
mission_data_manager_comment	Data_manager_comment
*See special considerations below	More_comment

NULL	Prod_created_date
created_by	Created_by
created_date	Created_date
created_by	Last_update_by
created_date	last_update_date
'ENR'	process_flag
batch_seq	batch_seq

* Special Considerations:

The BCMISSIONEDITS.more_comment column stores a 'Y' or 'N' to signify that a BCPLANKTONSTATNEDITS.mission_more_comment exists (that is to say that the BCPLANKTONSTATNEDITS.mission_more_comment is NOT NULL). This being the case, the value of this column (the text) is to be stored in the BCCOMMENTEDITS table and the BCMISSIONEDITS.more_comment = 'Y'.

When populating the BCMISSIONEDITS table, the validation procedures must ensure that no record is duplicated in the table when processing a 'batch' of data. Therefore, if there is an existing 'Mission' loaded in the BCMISSIONEDITS table, any subsequent 'Mission' data loaded must first ensure that the 'Mission' data does not already exist. If the record does not exist in the BCMISSIONEDITS table, then commit the new record to the table, otherwise, do not add the record (if it is exactly the same, that is to say that all column data for the specified 'Mission' record are the same).

- The first level of duplicate checking is to verify that the mission_descriptor value is different; if it is, then load the 'Mission' level data into the BCMISSIONEDITS table.If the mission_descriptor value is the same, then do not load the new 'Mission' level record.

- When a duplicate 'Mission' is found use the mission_edt_seq value of the existing 'Mission' level record and assign its value as the foreign key to the BCEVENTEDITS and/or the BCCOMMENTEDITS tables (if data exists in these two tables that is related to the Mission). It remains very important to preserve the PK/FK relationship of the data, therefore the linking of an existing 'Mission' record or newly entered 'Mission' record must be preserved to link to either the BCCOMMENTEDITS or the BCEVENTEDITS table records

Column Level Validations for the BCMISSIONEDITS table

These are the same validations as are described beginning with the previous section of the same name.

BCEVENTEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONSTATNEDITS and the BCCOMMENTEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONSTATNEDITS (OR OTHER)	**BCEVENTEDITS**
BCEVENTS_SEQ.nextval	event_edt_seq
NULL	event_seq
data_center_code	data_center_code
NULL	mission_seq
event_sdate	sdate
event_edate	edate
event_stime	stime
event_etime	etime
event_min_lat	min_lat

event_max_lat	max_lat
event_min_lon	min_lon
event_max_lon	max_lon
event_collector_stn_name	collector_station_name
event_collector_event_id	collector_event_id
event_UTC_offset	UTC_offset
event_collector_comment1	collector_comment
event_data_manager_comment	data_manager_comment
*See special considerations below	more_comment
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
'ENR'	process_flag
batch_seq	batch_seq
BCMISSIONEDITS.mission_edt_seq as a foreign key to the PK record.	mission_edt_seq

*Special Considerations:

The BCEVENTEDITS.more_comment column stores a 'Y' or 'N' to signify that a BCPLANKTONSTATNEDITS.event_more_comment exists (that is to say that the BCPLANKTONSTATNEDITS.event_more_comment is NOT NULL). This being the case, the value of this column is to be stored in the BCCOMMENTEDITS table and the BCEVENTEDITS.more_comment = 'Y'.

When populating the BCEVENTEDITS table, the validation procedures must ensure that no record is duplicated in the table when processing a 'batch' of data. Therefore, if there is an existing 'Event' loaded in the BCEVENTEDITS table, any subsequent 'Event' data loaded must first ensure that the 'Event' data does not already exist. If the record does not exist in the BCEVENTEDITS table, then commit the new record to the table, otherwise, do not add the record (if it is exactly the same, that is to say that all column data for the specified 'Event' record are the same).

- The first level of duplicate checking is to verify that the collector_event_id column values are different; if they are, then load the 'Event' level data into the BCEVENTEDITS table.If the collector_event_id values are the same, then check the other column values of the records that are being compared, to ensure that the records are truly identical. If the records are identical, do not load the new 'Event' level record.

- When a duplicate 'Event' is found use the event_edt_seq value of the existing 'Event' level record and assign its value as the foreign key to the BCACTIVITYEDITS and/or the BCCOMMENTEDITS tables (if data exists in these two tables that is related to the 'Event'). It remains very important to preserve the PK/FK relationship of the data, therefore the linking of an existing 'Event' record or newly entered 'Event' record must be preserved to link to either the BCACTIVITYEDITS or the BCCOMMENTEDITS table records.

Column Level Validations for the BCEVENTEDITS table

These are the same validations as are described beginning with the previous section of the same name.

BCCOMMENTEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONSTATNEDITS and the BCCOMMENTEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONSTATNEDITS (OR OTHER)	**BCCOMMENTEDITS**

BCCOMMENTS_SEQ.nextval	comment_edt_seq
NULL	comment_seq
Data_center_code	data_center_code
NULL	event_seq
NULL	mission_seq
Mission_more_comment and/or event_more_comment	edit_comment
*Ordinal number generator	comment_num
NULL	prod_created_date
Created_by	created_by
Created_date	created_date
Created_by	last_update_by
Created_date	last_update_date
'ENR'	process_flag
Batch_seq	batch_seq
BCMISSIONEDITS.mission_edt_seq as a foreign key to the PK record.	mission_edt_seq
BCEVENTEDITS.event_edt_seq as a foreign key to the PK record.	event_edt_seq

*Ordinal number generator:

- This table stores comments which are a continuation of comments in the BCMISSIONEDITS and the BCEVENTEDITS tables, since each of these tables may only store comments up to 2000 characters. Once a comment is longer than 2000 characters, that portion of the comment is written to the BCCOMMENTEDITS table. If a comment in the BCCOMMENTEDITS table is longer than 2000 characters, that comment may be continued in a new record in the BCCOMMENTEDITS table, referencing the parent table (BCMISSIONEDITS or BCEVENTEDITS) and numbered with an 'ordinal value' (internal table sequence) which will keep track of the order of the comments. The ordinal value will default to a value of 1 (one) for each new record which is a continuation from the comment columns in either the BCMISSIONEDITS or BCEVENTEDITS tables. The value of the ordinal will be incremented by one for each comment, which is a continuation of a comment within the BCCOMMENTEDITS table.

Column Level Validations for the BCCOMMENTEDITS table

These are the same validations as are described beginning with the previous section of the same name.

BCACTIVITYEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONSTATNEDITS and the BCACTIVITYEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_DISCRETE_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONSTATNEDITS (OR OTHER)	**BCACTIVITYEDITS**
BCACTIVITIES_SEQ.nextval	activity_edt_seq
NULL	activity_seq
NULL	event_seq
data_center_code	data_center_code
'PL'	data_pointer_code
'ECN'	process_flag
batch_seq	batch_seq
BCEVENTEDITS.event_edt_seq as a foreign key to the PK record.	event_edt_seq

BCPLANKTNHEDREDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONSTATNEDITS and the BCPLANKTNHEDREDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONSTATNEDITS (OR OTHER)	**BCPLANKTNHEDREDITS**
BCPLANKTNHEDRS_SEQ.NEXTVAL	pl_headr_edt_seq
NULL	plankton_seq
data_center_code	data_center_code
NULL	event_seq
NULL	activity_seq
pl_headr_gear_seq	gear_seq
pl_headr_sdate	Sdate
pl_headr_edate	Edate
pl_headr_stime	Stime
pl_headr_etime	Etime
pl_headr_phase_of_daylight	phase_of_daylight
pl_headr_slat	Slat
pl_headr_elat	Elat
pl_headr_slon	Slon
pl_headr_elon	Elon
pl_headr_time_qc_code	time_qc_code
pl_headr_position_qc_code	position_qc_code
pl_headr_start_depth	start_depth
pl_headr_end_depth	end_depth
pl_headr_sounding	Sounding
pl_headr_volume	Volume
pl_headr_volume_method_seq	volume_method_seq
pl_headr_lrg_plankton_removed	large_plankton_removed
pl_headr_mesh_size	mesh_size
pl_headr_collection_method_seq	collection_method_seq
pl_headr_collector_deplmt_id	collector_deployment_id
pl_headr_collector_sample_id	collector_sample_id
pl_headr_procedure_seq	procedure_seq
pl_headr_preservation_seq	preservation_seq
pl_headr_storage_seq	storage_seq
pl_headr_collector	Collector
pl_headr_collector_comment	collector_comment
pl_headr_meters_sqd_flag	meters_sqd_flag
pl_headr_data_manager_comment	data_manager_comment
pl_headr_responsible_group	responsible_group
pl_headr_shared_data	shared_data
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
'ENR'	process_flag
batch_seq	batch_seq
BCEVENTEDITS.event_edt_seq as a foreign key to the PK record.	event_edt_seq

BCACTIVITYEDITS.activity_edt_seq as a foreign key to the PK record.	activity_edt_seq

When populating the BCPLANKTNHEDREDITS table, the validation procedures must ensure that no record is duplicated in the table when processing a 'batch' of data. Therefore, if there is an existing 'Header' loaded in the BCPLANKTNHEDREDITS table, any subsequent 'Header' data loaded must first ensure that the 'Header' data does not already exist. If the record does not exist in the BCPLANKTNHEDREDITS table, then commit the new record to the table, otherwise, do not add the record (if it is exactly the same, that is to say that all column data for the specified 'Header' record are the same).

- The first level of duplicate checking is to verify that collector_sample_id an the preservation_seq column values in combination are different; if they are, then load the 'Header' level data into the BCPLANKTNHEDREDITS table.

- When a duplicate 'Header' is found use the pl_headr_edt_seq value of the existing 'Header' level record and assign its value as the foreign key to the BCPLANKTNHEDREDITS table (for detail data records). It remains very important to preserve the PK/FK relationship of the data; therefore the linking of an existing 'Header' record or newly entered 'Header' record must be preserved to link to the BCPLANKTNHEDREDITS table records.

Column Level Validations for the BCPLANKTNHEDREDITS table

Please Note: The Validation of the stime, etime, slat, slon, elat, and elon listed below do not take the assigned quality code value for the time_qc_code and position_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
pl_headr_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
plankton_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Check that the value exists in the BCDATACENTERS table. Column must be NOT NULL.	4001 – NOT NULL error 4002 – Invalid Data Center Code
event_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
activity_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
gear_seq	Check that the value exists in the BCGEARS table. Column must be NOT NULL.	4001 – NOT NULL error 4022 - Invalid gear sequence value.
sdate	Check that the date is a valid date (with a 4-digit year representation) and that the date is not in the future. Column must be NOT NULL.	4001 – NOT NULL error 4003 – Invalid date representation 4004 – Date is in the future. 4005 - sdate must be <= edate.
edate	Check that the date is a valid date (with a 4 digit year representation), that the date is not in the future, and that edate must be >= sdate. Column may be NULL.	4003 – Invalid date representation 4004 – Date is in the future. 4005 - sdate must be <= edate.

stime	Check that the time is a valid time (using 4 digits to represent it), that the time is between 0000 and 2359. Start time must be <= End time. Column may be NULL.	4010 – Invalid time. 4011 – Start time not before end time.
etime	Check that the time is a valid time (using 4 digits to represent it), that the time is between 0000 and 2359, and that etime must be >= stime. Column may be NULL.	4010 – Invalid time. 4012 – End time not after start time.
phase_of_daylight	Ensure that one of the following is the only accepted value for this column: day, night, twilight, unknown, unassigned. Column may be NULL.	4026 - Invalid phase_of_daylight value.
slat	Ensure that latitude range is -90.00 to 90.00. Column must be NOT NULL.	4001 – NOT NULL error 4013 – Invalid latitude range.
elat	Ensure that latitude range is -90.00 to 90.00. Column may be NULL.	4013 – Invalid latitude range.
slon	Ensure that longitude range is -180.00 to 180.00. Column must be NOT NULL.	4001 – NOT NULL error 4015 – Invalid longitude range.
elon	Ensure that longitude range is -180.00 to 180.00. Column may be NULL.	4015 – Invalid longitude range.
time_qc_code	Check that the value exists in the BCQUALCODES table. Column must be NOT NULL.	4001 – NOT NULL error 4018 – Invalid Quality Code
position_qc_code	Check that the value exists in the BCQUALCODES table. Column must be NOT NULL.	4001 – NOT NULL error 4018 – Invalid Quality Code
start_depth	Ensure that value is between 0 and 12000. Column must be NOT NULL.	4001 – NOT NULL error 4019 – Invalid depth range.
end_depth	Ensure that value is between 0 and 12000. Column must be NOT NULL.	4001 – NOT NULL error 4019 – Invalid depth range.
sounding	Ensure that value is between 0 and 12000.Column may be NULL.	4019 – Invalid depth range.
volume	Free format. No validation required.	
volume_method_seq	Check that the value exists in the BCVOLUMEMETHODS table. Column must be NOT NULL.	4001 – NOT NULL error 4028 - Invalid volume method sequence value.
large_plankton_removed	The large_plankton_removed must be either 'Y' for Yes or 'N' for No to represent that large plankton has been removed from the sample. Column may be NULL.	4027 - Invalid large_plankton_removed value.
mesh_size	Free format. No validation required.	
collection_method_seq	Check that the value exists in the BCCOLLECTIONMETHODS table. Column must be NOT NULL.	4001 – NOT NULL error 4029 - Invalid collection method sequence value.
collector_deployment_id	Free format. No validation required.	
collector_sample_id	Free format. No validation required.	
procedure_seq	Check that the value exists in the BCPROCEDURES table. Column must be NOT NULL.	4001 – NOT NULL error 4030 - Invalid procedure sequence value.
preservation_seq	Check that the value exists in the BCPRESERVATIONS table. Column must be NOT NULL.	4001 – NOT NULL error 4031 - Invalid preservation sequence value.
storage_seq	Check that the value exists in the BCSTORAGES table. Column must be NOT NULL.	4001 – NOT NULL error 4032 - Invalid storage sequence value.
collector	Free format. No validation required.	

collector_comment	Free format. No validation required.	
meters_sqd_flag	The meters_sqd_flag must be either 'Y' for Yes or 'N' for No to represent square meter measurements. Column may be NULL.	4033 - Invalid meters_sqd_flag value.
data_manager_comment	Free format. No validation required.	
responsible_group	List of values representing the various DFO sectors that will store data in the BIO-CHEM database system. No validation required.	
shared_data	Free format. No validation required.	
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
event_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCEVENTEDITS table. No validation required.	
activity_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCACTIVITYEDITS table. No validation required.	

The following information is a copy of the specifications (named "Convert Plankton DATA to General.doc") used to create the processing to handle the loading of the data from the BCPLANKTONDATAEDITS table (in the Data Manager Oracle Account) into the BCPLANKTNGENERLEDITS table. This information has been updated in this document in order to more accurately reflect the developed data validation processing as is currently developed (as of August 01, 2000).

The following spec pertains to the loading of data from the BCPLANKTONDATAEDITS table to the BCPLANKTNGENERLEDITS table.

All data records that are entered into the BCPLANKTONDATAEDITS table have a created_date column value of the date/timestamp of the data load (date and time). All data loaded at the same time will be assigned the same date/timestamp value. All records are also assigned a process_flag of 'NR' to signify a new record in the table.

All data must be managed by the plank_sample_key_value (pseudo Foreign Key) where the data in the dataset, which contains 'fatal errors', will be deleted based on the programmatically imposed PK/FK relationship

from the BCPLANKTONSTATNEDITS table record to the BCPLANKTONDATAEDITS table record(s), based on this column. If a BCPLANKTONSTATNEDITS table record is deemed to contain a fatal error, then those records (that caused the fatal error) are printed in a report. The Data Manager is notified of the fatal errors in the processing of the 'Station' record(s) then the dataset that contains the fatal error may be deleted from the 'Station' table. Likewise, all corresponding records that exist in the 'Data' table may also be deleted once they too have been assessed for fatal errors. The fatal errors report should encompass both the 'Station' and 'Data' records for a particular dataset to report the records that contain the fatal errors. Therefore the 'Station' records and the corresponding 'Data' records are validated for fatal errors before an error report is created or a dataset is deleted (deletion of PK values from the 'Station' table and the corresponding FK value from the 'Data' table).

Fatal Errors are those that consist of the following types of validations which result in error:

BCPLANKTONDATAEDITS		BCPLANKTONSTATNEDITS
mission_descriptor	=	mission_descriptor
event_collector_event_id	=	event_collector_event_id
event_collector_stn_name	=	event_collector_stn_name
plank_sample_key_value	=	plank_sample_key_value
data_center_code	=	data_center_code

If any of the above column validations fail, insert a record into the BCSTATNDATAERRORS table with the error_code value = 4042.

All of the validation conditions listed in the table above will generate a fatal error if the validation fails. When one of the above validations fail, the process_flag column must be assigned a value of 'DVE' (Data Validation Error). This will mark the record that is to be included in a report to the Data Manager to notify them of any fatal errors in the BCPLANKTONDATAEDITS table dataset. As the records are processed, assign the process_flag column a value of 'DVI' (Data Validation In progress) to denote that the record passed this level of validation. . If any "fatal validation errors" exist, once all records in the BCPLANKTONDATAEDITS dataset are processed (by plank_sample_key_value reference from the 'Station' table), notify the Data Manager via a message, the user may then generate an error report to display all of the fatal errors, and delete the dataset that contains the errors. The Data Manager must then fix all of the errors in the originating ASCII file, then reload the data.

The data loaded into this table via SQL*Loader must reference a corresponding record in the BCPLANKTONSTATNEDITS table via the plank_sample_key_value PK/FK relationship. If there is no record in the BCPLANKTONSTATNEDITS table for a record in the BCPLANKTONDATAEDITS table (creating an orphan data record), then the record must be marked as invalid by assigning a process_flag = 'DVO' (Data Validation Orphan record)) and insert a record into the BCSTATNDATAERRORS table with the error_code value = 4052..

Even if all 'Station' data is successfully validated, the 'Data' table may contain an invalid record marked as a fatal error that will cause the full dataset to be marked for deletion (records from both 'Station' and 'Data'). Therefore no records are deleted from the 'Station' or 'Data' tables until all records in a dataset from the BCPLANKTONSTATNEDITS table and the corresponding records from the BCPLANKTONDATAEDITS are processed and assigned a process_flag of 'SVE' or 'SVD' (for the 'Station' records), 'DVE' or 'DVO' (for the 'Data' records).

As stated above, once the data record passes the above listed validations, the process_flag is assigned a value of 'DVI' (Data Validation In progress). If a datasets process_flag record values all equal 'DVI', then the records are deemed as passing the first validation layer and are prepared to be loaded into the Edit Table System.

The table below outlines the mapping of the data from the BCPLANKTONDATAEDITS table to the BCPLANKTNGENERLEDITS table. All data, in a dataset containing all records having a process_flag = 'DVI', are to be loaded from the BCPLANKTONDATAEDITS table to the BCPLANKTNGENERLEDITS table.
As the records are loaded into the BCPLANKTNGENERLEDITS table, the originating data record in the BCPLANKTONDATAEDITS table has a process_flag = 'DVI' (Data Validation In progress) and duplicate checking must occur to identify duplicate data. Duplicate 'General' records must not exist in the BCPLANKTNGENERLEDITS table for any one BCPLANKTNHEDREDITS ('Header') record, therefore once a record is identified as being a duplicate, it should not be committed to the table.

BCPLANKTNGENERLEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONDATAEDITS and the BCPLANKTNGENERLEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONDATAEDITS (or Other)	**BCPLANKTNGENERLEDITS**
BCPLANKTNGENERLS_SEQ.nextval	pl_general_edt_seq
NULL	Plankton_general_seq
data_center_code	data_center_code
NULL	plankton_seq
pl_gen_national_taxonomic_seq	national_taxonomic_seq
pl_gen_collector_taxonomic_id	collector_taxonomic_id
pl_gen_life_history_seq	life_history_seq
pl_gen_trophic_seq	trophic_seq
pl_gen_min_sieve	min_sieve
pl_gen_max_sieve	max_sieve
pl_gen_split_fraction	split_fraction
pl_gen_sex_seq	sex_seq
pl_gen_counts	counts
pl_gen_count_pct	count_pct
pl_gen_wet_weight	wet_weight
pl_gen_ dry_weight	dry_weight
pl_gen_bio_volume	bio_volume
pl_gen_presence	presence
pl_gen_collector_comment	collector_comment
pl_gen_source	source
pl_gen_data_manager_comment	data_manager_comment
NULL	prod_created_date
created_by	created_by
created_date	created_date
Created_by	last_update_by
Created_date	last_update_date
'ENR'	process_flag
Batch_seq	batch_seq
BCPLANKTONHEDREDITS.pl_headr_edt_seq as a foreign key to the PK record.	pl_headr_edt_seq

Column Validation

All error validation in the Edit Table System is to be handled in the same manner. Records that pass the programmatic record level validation are assigned a process_flag = 'ECN' (Edit table record is CleaN and error free). Once a record is found that fails a specified programmatic column level validation (such as NOT NULL constraint violation), that record is assigned a process_flag = 'ERR' (edit table ERRor). The BCERRORS table must also be populated with the following values for each column in a record in error:

BCERRORS table definition:

Column Name	Column NULL Option	Data Value
error_num_seq	NOT NULL	>call sequence generator<
edit_table_name	NOT NULL	Table name where validation error occurred.
record_num_seq	NOT NULL	Primary key value of the record where the validation error occurred.
column_name	NOT NULL	Column name where the validation error occurred.
error_code	NOT NULL	Error code assigned to the validation error. Assigned by the procedure and referenced from the BCERRORCODES table.
last_updated_by	NOT NULL	Defaults to the logged in Data Manager user name that is validating the data.
last_update_date	NOT NULL	Defaults to the sysdate of the record editing.

Once an invalid record is edited and corrected, the process_flag = 'ECN'.

Column level Validations for the BCPLANKTNGENERLEDITS table.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
pl_general_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
plankton_general_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Verify that the column is NOT NULL. Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error. 4002 – Invalid Data Center Code.
plankton_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
national_taxonomic_seq	Verify that the column is NOT NULL. Check that the value exists in the BCNATNLTAXONCODES table.	4001 – NOT NULL error. 4034 - Invalid national taxonomic sequence value.
collector_taxonomic_id	Free format. No validation required.	
life_history_seq	Verify that the column is NOT NULL.Check that the value exists in the BCLIFEHISTORIES table.	4001 – NOT NULL error. 4035 - Invalid life history sequence value.
trophic_seq	Verify that the column is NOT NULL. Check that the value exists in the BCTROPHICDESCRIPTORS table.	4001 – NOT NULL error. 4036 - Invalid trophic sequence value.
min_sieve	Free format. No validation required.	
max_sieve	Free format. No validation required.	
split_fraction	Free format. No validation required.	
sex_seq	Verify that the column is NOT NULL. Check that the value exists in the BCSEXES table.	4001 – NOT NULL error. 4037 - Invalid sex sequence value.
counts	Free format. No validation required.	
count_pct	Free format. No validation required.	
wet_weight	Free format. No validation required.	

dry_weight	Free format. No validation required.	
bio_volume	Free format. No validation required.	
presence	Value must be either 'Y' for Yes or 'N' for No to represent the presence or absence of organisms in the sample. Column may be NULL.	4038 - Invalid value for the presence column.
collector_comment	Free format. No validation required.	
source	Free Format. Column must be NOT NULL.	4001 – NOT NULL error.
data_manager_comment	Free format. No validation required.	
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
pl_headr_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCPLANKTNHEDREDITS table. No validation required.	

BCPLANKTNFREQEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONDATAEDITS and the BCPLANKTNFREQEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONDATAEDITS (OR OTHER)	**BCPLANKTNFREQEDITS**
BCPLANKTNFREQS_SEQ	pl_freq_edt_seq
NULL	plankton_frequency_seq
data_center_code	data_center_code
pl_freq_data_type_seq	data_type_seq
Null	plankton_general_seq
pl_freq_upper_bin_size	upper_bin_size
pl_freq_lower_bin_size	lower_bin_size
pl_freq_bug_count	bug_count

pl_freq_bug_seq	bug_seq
pl_freq_data_value	data_value
pl_freq_data_qc_code	data_qc_code
pl_freq_detail_collector	detail_collector
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
'ENR'	process_flag
Batch_seq	batch_seq
BCPLANKTONGENERLEDITS.pl_general_edt_seq as a foreign key to the PK record.	pl_general_edt_seq

Once an invalid record is edited and corrected, the process_flag = 'ECN'.

Column level Validations for the BCPLANKTNFREQEDITS table.

Please Note: The Validation of the data_value column listed below does not take the assigned quality code value for the data_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
pl_freq_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
plankton_frequency_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Verify that the column is NOT NULL. Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error. 4002 – Invalid Data Center Code.
data_type_seq	Verify that the column is NOT NULL. Check that the value exists in the BCDATATYPES table.	4001 – NOT NULL error 4023 - Invalid Data Type value.
plankton_general_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
upper_bin_size	No validation required.	
lower_bin_size	No validation required.	
bug_count	Verify that the column is NOT NULL and NOT negative.	4001 – NOT NULL error. 4058 - Invalid Value. Must be greater than 0.
bug_seq	Verify that the column is NOT NULL and NOT negative.	4001 – NOT NULL error. 4058 - Invalid Value. Must be greater than 0.

data_value	Verify that the column is NOT NULL. Ensure that the data_value result falls between the ranges set for a specified datatype by the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_value columns for the specified datatype (If the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_values exist in the table).	4001 – NOT NULL error 4025 - Invalid data_value result.
data_qc_code	Verify that the column is NOT NULL. Check that the value exists in the BCQUALCODES table.	4001 – NOT NULL error 4018 - Invalid Quality Code value.
detail_collector	No Validation required.	
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
pl_general_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCPLANKTNHEDREDITS table. No validation required.	

BCPLANKTNDTAILEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONDATAEDITS and the BCPLANKTNDTAILEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONDATAEDITS (OR OTHER)	**BCPLANKTNDTAILEDITS**
BCPLANKTNDTAILS_SEQ.nextval	pl_detail_edt_seq
NULL	plankton_detail_seq
data_center_code	data_center_code
pl_detail_data_type_seq	data_type_seq
Null	plankton_general_seq
pl_detail_data_value	data_value

pl_detail_data_qc_code	data_qc_code
pl_detail_detail_collector	detail_collector
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
'ENR'	process_flag
Batch_seq	batch_seq
BCPLANKTONGENERLEDITS.pl_general_edt_seq as a foreign key to the PK record.	pl_general_edt_seq

Once an invalid record is edited and corrected, the process_flag = 'ECN'.

Column level Validations for the BCPLANKTNDTAILEDITS table.

Please Note: The Validation of the data_value column listed below does not take the assigned quality code value for the data_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
pl_detail_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
plankton_detail_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Verify that the column is NOT NULL. Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error. 4002 – Invalid Data Center Code.
data_type_seq	Verify that the column is NOT NULL. Check that the value exists in the BCDATATYPES table.	4001 – NOT NULL error 4023 - Invalid Data Type value.
plankton_general_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_value	Verify that the column is NOT NULL. Ensure that the data_value result falls between the ranges set for a specified datatype by the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_value columns for the specified datatype (If the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_values exist in the table).	4001 – NOT NULL error 4025 - Invalid data_value result.
data_qc_code	Verify that the column is NOT NULL. Check that the value exists in the BCQUALCODES table.	4001 – NOT NULL error 4018 - Invalid Quality Code value.
detail_collector	No Validation required.	

prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	
last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
pl_general_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCPLANKTNHEDREDITS table. No validation required.	

BCPLANKTNINDIVEDITS Column Mapping

The table in this section displays the column level mapping between the BCPLANKTONDATAEDITS and the BCPLANKTNINDIVEDITS table that is done in the Forms Library procedure PROCS.ASSIGN_BATCH_SEQ which calls the data manager stored procedure POPULATE_PLANKTON_EDITS_PKG.POPULATE_PLANKTON_EDITS to populate the data manager EDIT tables with the data stored in the BCPLANKTONSTATNEDITS and BCPLANKTONDATAEDITS data manager tables.

BCPLANKTONDATAEDITS (OR OTHER)	**BCPLANKTNINDIVEDITS**
BCPLANKTNINDIVDLS_SEQ.nextval	pl_indiv_edt_seq
NULL	plankton_individual_seq
data_center_code	data_center_code
pl_indiv_data_type_seq	data_type_seq
Null	plankton_general_seq
pl_indiv_bug_seq	bug_seq
pl_ indiv_data_value	data_value
pl_ indiv _data_qc_code	data_qc_code
pl_ indiv _data_collector	data_collector
NULL	prod_created_date
created_by	created_by
created_date	created_date
created_by	last_update_by
created_date	last_update_date
'ENR'	process_flag
Batch_seq	batch_seq
BCPLANKTONGENERLEDITS.pl_general_edt_seq as a foreign key to the PK record.	pl_general_edt_seq

Once an invalid record is edited and corrected, the process_flag = 'ECN'.

Column level Validations for the BCPLANKTNINDIVEDITS table.

Please Note: The Validation of the data_value column listed below does not take the assigned quality code value for the data_qc_code column values into account. The inclusion of the quality code reference in the column level validation is defined in the section named Quality Code Validations.

COLUMN NAME	COLUMN VALIDATION	ERROR CODE(S)
pl_detail_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. No validation required.	
plankton_detail_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
data_center_code	Verify that the column is NOT NULL. Check that the value exists in the BCDATACENTERS table.	4001 – NOT NULL error. 4002 – Invalid Data Center Code.
data_type_seq	Verify that the column is NOT NULL. Check that the value exists in the BCDATATYPES table.	4001 – NOT NULL error 4023 - Invalid Data Type value.
plankton_general_seq	System generated unique value created in the production table for data loaded from the edit table to the production table. No validation required.	
bug_seq		
data_value	Verify that the column is NOT NULL. Ensure that the data_value result falls between the ranges set for a specified datatype by the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_value columns for the specified datatype (If the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_values exist in the table).	4001 – NOT NULL error 4025 - Invalid data_value result.
data_qc_code	Verify that the column is NOT NULL. Check that the value exists in the BCQUALCODES table.	4001 – NOT NULL error 4018 - Invalid Quality Code value.
data_collector	No Validation required.	
prod_created_date	Defaults to the system date the record is loaded to the production tables from the edit tables after data is validated and duplicate checked in the edit table. No validation required. Data that is loaded from the production tables to the edit tables need not reassign this date.	
created_by	Defaults to the logged in data manager's user name. No validation required.	
created_date	Defaults to the system date the record is created in the edit table. No validation required.	
last_update_by	Defaults to the logged in data manager's user name who updated/edited the record. No validation required.	

last_update_date	Defaults to the system date the record is updated/edited in the edit table. No validation required.	
process_flag	Assigned programmatically by the application.	
batch_seq	Assigned programmatically by the application.	
pl_general_edt_seq	System generated unique value created in the edit table for new data and for data loaded from the production table. Foreign Key to the BCPLANKTNHEDREDITS table. No validation required.	

Quality Code Validations

Note: This Section (in its entirety) was initially prepared on June 20, 2000 as the specifications document for the development of the Quality Code Validations for time values, latitude and longitude values, and data values.

The purpose of this document is to define the specification for the Setting of Quality Code(s) for Data Validation. The currently deployed application validates all records in a batch for specific column value ranges (for columns that have specified acceptable data value ranges) regardless of the qc_code (quality code) value assigned. Time, position and data value columns are assigned a quality code in order to assign a measure of quality to the data. Once a Data Manager, data custodian, etc. assigns a quality code value to the data, the BIOCHEM Database Edit Application validation is to perform the validation according to the quality code and validation rules assigned for the time, position and data values.

System Overview

Table data used during the validation process (data value ranges and quality code values) are accessed from the following tables:

BIOCHEM on BANK:

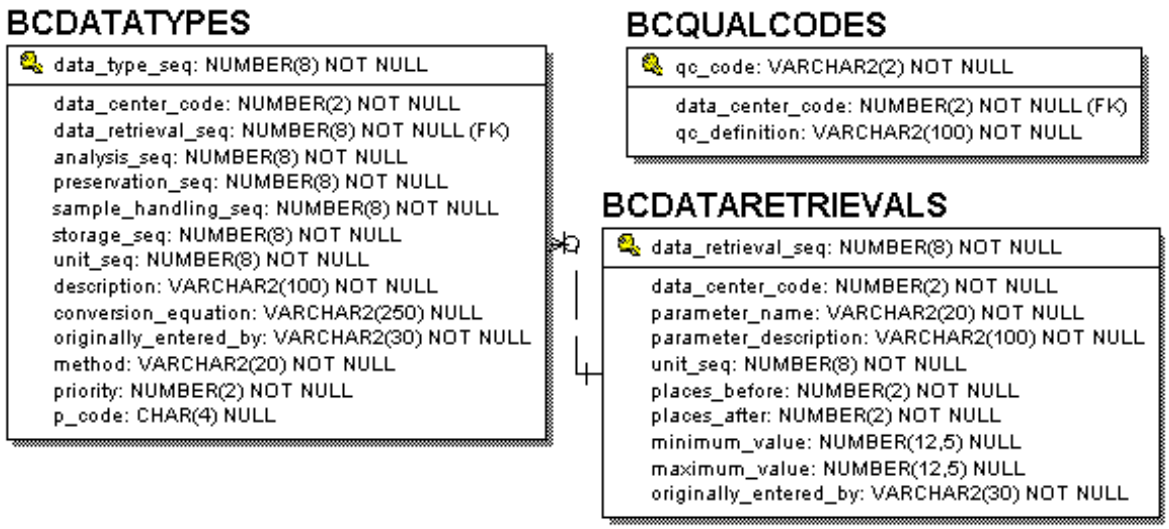


Figure 1

Columns referenced from the tables in Figure 1:

Note: BIOCHEM tables are referenced via synonyms already created in the Data Manager Oracle accounts. (The BCDATARETRIEVALS table stores the "Master" parameters and the DATATYPES table stores the sub-types of the "Master" parameters.)

BCDATATYPES.data_type_seq NUMBER(8) NOT NULL

- Column is PK for all data_type_seq columns found in the "Detail" tables (stores a reference to the property/parameter being measured).

BCDATARETRIEVALS.minimum_value NUMBER(12,5) NULL

- Stores the minimum_value allowable for a data_value for a type of datatypes (property/parameter). This value is referenced for the data_value column validation.

BCDATARETRIEVALS.maximum_value NUMBER(12,5) NULL

- Stores the maximum _value allowable for a data_value for a type of datatypes (property/parameter). This value is referenced for the data_value column validation.

Columns referenced from the Data Manger accounts on BANK:

BCDISCRETEHEDREDITS.time_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCDISCRETEHEDREDITS.stime NUMBER(4) NULL and BCDISCRETEHEDREDITS.etime NUMBER(4) NULL column values.

BCDISCRETEHEDREDITS.position_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCDISCRETEHEDREDITS.slat NUMBER(8,5) NULL, BCDISCRETEHEDREDITS.elat NUMBER(8,5) NULL, BCDISCRETEHEDREDITS.slon NUMBER(9,5) NULL, BCDISCRETEHEDREDITS.elon NUMBER(9,5) NULL column values.

BCDISCRETEDTAILEDITS.data_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCDISCRETEDTAILEDITS.data_value NUMBER(10,5) NULL column value.

BCDISREPLICATEDITS.data_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCDISREPLICATEDITS.data_value NUMBER(10,5) NULL column value.

BCPLANKTNHEDREDITS.time_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCPLANKTNHEDREDITS.stime NUMBER(4) NULL and BCPLANKTNHEDREDITS.etime NUMBER(4) NULL column values.

BCPLANKTNHEDREDITS.position_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCPLANKTNHEDREDITS.slat NUMBER(8,5) NULL, BCPLANKTNHEDREDITS.elat NUMBER(8,5) NULL,
  BCPLANKTNHEDREDITS.slon NUMBER(9,5) NULL,
  BCPLANKTNHEDREDITS.elon NUMBER(9,5) NULL column values.

BCPLANKTNFREQEDITS.data_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCPLANKTNFREQEDITS.data_value NUMBER(10,5) NULL column value.

BCPLANKTNDTAILEDITS.data_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCPLANKTNDTAILEDITS.data_value NUMBER(10,5) NULL column value.

BCPLANKTNINDIVEDITS.data_qc_code VARCHAR2(2) NULL

- Quality Code reference (FK to BIOCHEM.BCQUALCODES.qc_code) associated with the BCPLANKTNINDIVEDITS.data_value NUMBER(10,5) NULL column value.

Packaged Validation Procedures/Functions:

Note: The validation procedures already exist in the Data Manager Oracle accounts (as stored procedures).

ITEM_VALIDATION_PKG.VALIDATE_TIME_RANGE

- FUNCTION VALIDATE_TIME_RANGE(in_time_par IN NUMBER,
  table_name_par IN VARCHAR2,
  edit_seq_par IN NUMBER,
  column_name_par IN VARCHAR2,
  username_par IN VARCHAR2,
  batch_seq_par IN NUMBER) RETURN VARCHAR2

ITEM_VALIDATION_PKG.VALIDATE_LAT_RANGE

- FUNCTION VALIDATE_LAT_RANGE(in_lat_par IN NUMBER,
  table_name_par IN VARCHAR2,
  edit_seq_par IN NUMBER,
  column_name_par IN VARCHAR2,
  username_par IN VARCHAR2,
  batch_seq_par IN NUMBER) RETURN VARCHAR2

ITEM_VALIDATION_PKG.VALIDATE_LON_RANGE

- FUNCTION VALIDATE_LON_RANGE(in_lon_par IN NUMBER,
  table_name_par IN VARCHAR2,
  edit_seq_par IN NUMBER,
  column_name_par IN VARCHAR2,
  username_par IN VARCHAR2,
  batch_seq_par IN NUMBER) RETURN VARCHAR2

ITEM_VALIDATION_PKG.VALIDATE_DATA_VALUE

- FUNCTION VALIDATE_DATA_VALUE(in_data_value_par IN NUMBER,
  in_data_type_par IN NUMBER,
  in_averaged_data_par IN VARCHAR2,
  table_name_par IN VARCHAR2,
  edit_seq_par IN NUMBER,
  column_name_par IN VARCHAR2,
  username_par IN VARCHAR2,
  batch_seq_par IN NUMBER) RETURN VARCHAR2

Each of the above Functions currently validates the appropriate data based on predefined ranges that are either coded into the validations or referenced programmatically from table values:

- Valid time is between 0000 and 2359 (minutes cannot be greater than 59) (in code).
- Valid latitude values are between -90.00 and 90.00 (in code).
- Valid longitude values are between -180.00 and 180.00 (in code).
- Valid data values are between the BCDATARETRIEVALS.minimum_value and the BCDATARETRIEVALS.maximum_value (referenced dynamically from table values).

The above mentioned validation functions are executed during batch processing (when validating a new batch of data once loaded from the "Station/Data" tables, and by selecting a batch to revalidate from the Batch Validation processing form) and during the save routine to validate all columns in a record.

Specification

The business rules which govern the validation of "time" (stime, etime), position (slat, elat, slon, elon), and data_value column values are detailed in the points below. These validation specifications (listed below) are in addition to the existing validations (therefore this specification document will not address the already developed validation rules for time, position, or data values but will allude to them with the following notation: (normal validation)).

The validation rules are stated in a pseudo code-like style in order to show the flow of the validation procedures (and not miss any detail of the validation).

TIME: stime or etime (columns accept NULL values)

1. If stime or etime are NOT NULL then
    If the qc_code (quality code) IN ("4", "6", "9") then
        the time validation automatically fails. Error 4045 (time_qc_code error) and 4046 (time validation error based on invalid time_qc_code) are to be written to the BCERRORS table and the error flag variable is assigned a value of "Y" (error_flag_var := 'Y').
    Else range check the time values (normal validation).
  Else if stime and etime are NULL then
    If the qc_code IN ("1", "2", "3", "4", "5", "6") then
        Automatically fail the validation. Error 4047 (time_qc_code error) is written to the BCERRORS table and the error flag variable is assigned a value of "Y" (error_flag_var := 'Y').

POSITION: slat, elat, slon, or elon (slat/slon are NOT NULL columns)

2. If position is NOT NULL then
    If the qc_code (quality code) IN ("4", "6", "9") then
        the lat/lon validation automatically fails. Error 4048 (position_qc_code error) and 4049 (position validation error based on invalid position_qc_code) is to be written to the BCERRORS table and the error flag variable is assigned a value of "Y" (error_flag_var := 'Y').
    Else range check the position values (normal validation).

DATA VALUE: (data_value is a NOT NULL column)

3. If data value is NOT NULL then
    If qc_code (quality code) is "0", or "1" then
        Perform the data_value range check (normal validation).
        If the data_value is outside of the ranges (BCDATARETRIEVALS.minimum_value
            and the BCDATARETRIEVALS.maximum_value) then
            fail the data_value validation. The validation error (4025 (4039 if table name is BCDISCRETEDTAILEDITS and the averaged_data column = 'Y')) is to be written to the BCERRORS table and the error flag variable is assigned a value of "Y" (error_flag_var := 'Y').
        Else pass the validation.
    Else qc_code (quality code) IN ("2", "3", "4", "5", "6") then
        Do not perform the data_value range check. Pass the validation.
    Else qc_code (quality code) is "9" then
        The data_qc_code validation error (4050 quality code value not accepted as valid) is to be written to the BCERRORS table and the error flag variable is assigned a value of "Y" (error_flag_var := 'Y') ("9" may not exist as a qc_code value since the data_value column is NOT NULL).
  Else
    Error: NOT NULL error. Column is NOT NULL (included in (normal validation)).