Annex E
Functions and Web services

Table of Functions

Annex E
Functions and Web services

1. Introduction

The following sections explain the conventions and structured data storage utilised in the specification for each function.

1.1 Result Identifier

Each function returns a Result Identifier. This value will indicate whether the function succeeded or failed. A failure could be the result of a business decision (a failed check) or the result of an unanticipated exception error (a run time error). The convention used here is that zero (0) indicates failure and one (1) indicates success.

1.2 Transaction Object

The Transaction Object is used to store all the elements involved in a transaction required for processing. The Transaction Object is a parent class to the TransactionUnitBlock Object. The structure of the Transaction Object is as follows:

Structure TransactionObject
- TransactionIdentifier As String
- TransactionType As Integer
- SuppTransactionType As Integer
- TransactionStatus As Integer
- TransactionStatusDate As DateTime
- TransferringRegistryIdentifier As String
- TransferringRegistryAccountType As Integer
- TransferringRegistryAccountIdentifier As Long
- AcquiringRegistryIdentifier As String
- AcquiringRegistryAccountType As Integer
- AcquiringRegistryAccountIdentifier As Long
- NotificationIdentifier As Integer
- TransactionBlocks (array) As UnitBlock Object

1.3 UnitBlock Object

The functions below reference a UnitBlockObject to obtain the unit blocks associated with a transaction. An array of UnitBlockObjects is often called for because more than one block can be associated with a transaction. This UnitBlockObject is used in two ways. It is referenced as ITLUnitBlockObject when it holds data retrieved from the ITL Unit Block table. It is referenced as RegistryUnitBlock when it holds data submitted from a Registry through either a Proposal or notification. When the object is used for data from the ITL, the Block ID field will reference the unique identifier of the block's record in the Unit Block table. Data submitted by a registry will not contain Block ID. The structure of the UnitBlockObject is as follows:

Structure UnitBlockObject
    UnitSerialBlockStart As Long
    UnitSerialBlockEnd As Long
    OriginatingPartyCode As String
    UnitType As Integer
    SuppUnitType As Integer
    OriginalCommitPeriod As Integer
    ApplicableCommitPeriod As Integer
    LULUCFActivity As Integer
    ProjectIdentifier As Integer
    Track As Integer
    BlockRole As String
    AcquiringRegistryAccountType As Integer
    AcquiringRegistryAccountIdentifier As Long
    TransferringRegistryAccountType As Integer
    TransferringRegistryAccountIdentifier As Long
    YearInCommitmentPeriod As Integer
    InstallationIdentifier As Long
    ExpiryDate As DateTime
    BlockID As Number
    UnitStatusCode As Integer

1.4 MessageUnitBlock Object

The MessageUnitBlock Object is used to return only the attributes that uniquely identify it and is generally returned as part of a notification to a Registry in conjunction with a CheckResponse Object.

Structure MessageUnitBlockObject
    UnitSerialBlockStart As Long
    UnitSerialBlockEnd As Long
    OriginatingParty Code As String

1.5 CheckResponse Object

The CheckResponse Object is used to collect all checks that are applicable to a process. As each check is evaluated, the result identifier is modified to indicate a success or failure for the check. When the CheckResponse Object is returned, it only includes those response codes that have failed.

Structure CheckResponseObject
    Result Identifier As Integer
    Response Code As Integer

1.6 EvaluationResult Object

This object stores the specific CheckResponse Object along with the affected Unit Blocks.

Structure EvaluationResultObject
    CheckResponseObject
    UnitBlock (array) As RegistryUnitBlockObject

1.7 Reconciliation Object

The Reconciliation Object is used to describe a reconciliation action and its current status.

Structure ReconciliationObject
    ReconciliationIdentifier As String
    ReconciliationSnapshotDateTime As DateTime
    ReconciliationBeginDate As DateTime
    ReconciliationStatusCode As Integer
    ReconciliationEndDate As DateTime
    ReconciliationStatusDateTime As DateTime
    ReconciliationPhaseCode As Integer

1.8 Totals Object

The Totals Object is used by the reconciliation process to store the number of units held by a registry, account type, or account.

Structure TotalsObject
    ReconciliationIdentifer As String
    ReconciliationSnapshotDateTime As DateTime
    HoldingRegistry As String
    AccountType As Integer
    AccountCommitPeriod As Integer
    AccountIdentifier As Integer
    UnitType As Integer
    SupplementaryUnitType As Integer
    UnitCount As Integer

2. Specified Functions

The following functions will be implemented to carry out the duties of the ITL. They are listed in alphabetical order.

224 **Figure E1: AcceptNotification (Web service)**
225

Purpose
This is the HTTP SOAP request that registries use to send notifications regarding transactions in process. Once the request has passed preliminary checks and been written to the queue, an HTTP SOAP response of success or an HTTP SOAP fault of failure is returned. This Web service is hosted on both the ITL and the Registry.
Inputs
TransactionObject, RegistryUnitBlockObject, CheckResponseObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the preliminary checks before writing message to the message queue.
Outputs
Result_Identifier
Call(s)
Preliminary_Checks

226

227 **Figure E2: AcceptProposal (Web service)**
228

Purpose
This is the HTTP SOAP request that registries use to send proposals. Once the request has passed preliminary checks and is written to the message queue, a HTTP SOAP response of success is returned. This Web service is hosted on both the ITL and the Registry.
Inputs
TransactionObject, RegistryUnitBlockObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the preliminary checks before writing message to message queue.
Outputs
Result_Identifier
Call(s)
Preliminary_Checks

229

Figure E3: CER_Expired_Check (Function)

Purpose
Identify tCERs and lCERs that are to expire in 30 days and notify the registries that hold these units.
Inputs
Table(s)
Process
Every 24 hours, search for tCERs and lCERs that will expire within 30 days. Select Holding_Registry_Code, Originating_Party_Code, Start_Block, End_Block From Unit_Block Where Unit_Type = (6,7) and Expiration_Date – Current Date < 30 days If any records found, insert record into the Notification table for Impending Expiry of lCER or tCER (type 3) For each Holding_Registry_Code represented in the above result set, Insert record into Registry_Notification table. Call AcceptITLNotice on the Holding Registry with the Notification Type Code and the identifying information of each block found.
Outputs
Result_Identifier
Call(s)

Figure E4: Check_Reconciliation_Message (Function)

Purpose
The ITL will execute this function every time a reconciliation message is retrieved from the message queue. This function will in turn execute the checks for message age, reconciliation data integrity checks, and the reconciliation message sequence checks. If any of the checks fail, this message will send a notification of failure to the registry from which the message was received. If all the checks pass, the message will be routed to the next phase as appropriate.
Inputs
ReconciliationObject, TransactionObject
Table(s)
Process
Begin database transaction Call Retrieve_From_Message_Queue. Set Select for update database transaction. Call Write_To_Message_Log. Call Check_Registry_Status. If this function fails, stop processing message and call Reconciliation_Failure_Notification. Call Message_Age_Check. If the function fails, stop processing message and call Reconciliation_Failure_Notification. Call Reconciliation_Data_Integrity_Check, followed by Reconciliation_Message_Sequence_Check. If any check in either function fails, call Reconciliation_Failure_Notification after both functions have completed (so more than one error may be collected). Call Remove_From_Message_Queue so the message will not be processed again. Commit the database transaction. Determine message sender. If it was an STL, call Evaluate_STL_Reconciliation_Message. If it was any other registry, call Evaluate_Registry_Reconciliation_Message.
Outputs
ResultIdentifier ReconciliationObject TransactionObject

(cont.)

Figure E4: Check_Reconciliation_Message (Function) (cont.)

Call(s)
Retrieve_From_Message_Queue Message_Age_Check Write_To_Message_Log Check_Registry_Status Reconciliation_Data_Integrity_Check Reconciliation_Message_Sequence_Check Evaluate_Registry_Reconciliation_Message Evaluate_STL_Reconciliation_Message Reconciliation_Failure_Notification Remove_From_Message_Queue

235

Figure E5: Check_Registry_Status (Function)

Purpose
This function performs a lookup and evaluation of the status of a registry to determine if it is allowed to participate in a transaction or reconciliation action.
Inputs
ReconciliationObject, TransactionObject
Table(s)
Registry, Registry_Status_History
Process
Evaluate the status of the Initiating Registry by retrieving and executing status checks. Call Get_Checks to return the registry status checks. Call Execute_Checks to perform the check. If any check fails, return failure.
Outputs
CheckResponseObject
Call(s)

239 **Figure E6: Check_Version (Function)**
240

Purpose
This function compares the major and minor version number in the HTTP SOAP request. These values are checked against the current version of the ITL. Major version numbers must match or the request will be rejected. Messages with an old minor version are still processed, although a warning is issued.
Inputs
Major Version, Minor Version
Table(s)
This function does not interact with the database.
Process
Compare Major Version Number o If not a match, return message 1031. Return HTTP SOAP response of failure. o If a match, then compare Minor Version Number. If not a match, return message 1032.
Outputs
CheckResponseObject Result_Identifier
Call(s)

241

Figure E7: Close_Reconciliation_Action (Function)

Purpose
This function will update the Reconciliation_Log table with the end date of a reconciliation action.
Inputs
ReconciliationObject
Table(s)
Reconciliation_Log
Process
Update Reconciliation_Log table so that Recon_End_Date = input reconciliation end date.
Outputs
ResultIdentifier
Call(s)

Figure E8: Data_Integrity_Check (Function)

Purpose
This function will check the data in the TransactionObject to ensure that it meets the minimum requirements to begin processing the proposed transaction. See Section 5.4.5 for a list of all data integrity checks performed by this function.
Inputs
TransactionObject, RegistryUnitBlockObject, ITLUnitBlockObject
Table(s)
Check_Catalog, Response_Catalog
Process
For each check identified in Section 5.4.5, evaluate the TransactionObject against the check. Each check will compare data from the registry with data from the ITL. For each check, collect the appropriate response code returned and store in the CheckResponseObject. If there is a failure of a data integrity check, exit this function. If there is no failure, call the Message_Sequence_Check function to continue processing the transaction.
Outputs
CheckResponseObject TransactionObject
Call(s)
Message_Sequence_Check

248 **Figure E9: Delete_Inconsistent_Block (Function)**
249

Purpose
This function will delete records in the Inconsistent Block table as part of clearing blocks from a reconciliation process. It will also update the unit status in the Unit_Block table.
Inputs
ReconciliationObject
Table(s)
Inconsistent_Block, Unit_Block
Process
For each unit block in the UnitBlock array of the ReconciliationObject, delete record in the Inconsistent_Block table Update unit_status in Unit_Block table If the unit_status = 2 or 3 unit_status = unit_status – 2 Update Last_Action_Date in Unit_Block table with the current DateTime.
Outputs
Result_Identifier Inconsistent_Block_ID
Call(s)

Figure E10: Delineate_Units (Function)

Purpose
This function updates the status of a unit block so that the units are either available for transactions or are unavailable due to some ongoing transaction or reconciliation process. If the unit status is greater than zero, the unit cannot be traded. The unit status also keeps track of the reason the unit cannot be traded through a binary code system. When the unit is part of an ongoing transaction a value of 1 is added to the current unit status. When unit is part of an inconsistent block identified by ITL reconciliation a value of 2 is added to the current unit status. When the unit is part of an inconsistent block identified by STL reconciliation, a value of 4 is added to the current unit status. When the value of unit status is 1, 3, or 5, the unit is part of an ongoing transaction. When the value is 2 or 3, the unit is part of an inconsistent block identified by ITL reconciliation. When the value is 4 or 5, the unit is part of an inconsistent block identified by STL reconciliation. The unit status should reflect only one instance of a unit status, even if multiple processes have tied up the unit for the same reason.
Inputs
TransactionObject, ITLUnitBlockObject, UnitStatusCode, Engaged_flg
Table(s)
Unit_Block
Process
For each transaction block in the TransactionObject and associated ITLUnitBlockObjects, update the Unit_Status_Code in the Unit_Block table as described below. If the parameter "engaged_flg" is passed as true, and the existing unit status code is not equal to 1, then add 1 to the unit status code value. If parameter "engaged_flg" is passed as false, then subtract 1 from the unit status code value. Update the Unit_Status_DateTime fields with the system DateTime. For each transaction block in the TransactionObject and associated ITLUnitBlockObjects, update the unit_status in the Unit_Block table as described below.

(cont.)

Figure E10: Delineate_Units (Function) (cont.)

Process
When engaged_flg = 1 (ongoing transaction), If the unit_status <> 1,3,5 Unit_status = Unit_status + 1 When with engaged_flg = 0 (free to be traded), If the unit_status = 1,3,5 unit_status = unit_status - 1
Outputs
Call(s)

256

257 **Figure E11: Determine_Route_For_Proposal (Function)**
258

Purpose
After a proposed transaction has been checked by the ITL, this function will route the transaction to the next appropriate party based on the transaction status and transaction type.
Inputs
TransactionObject
Table(s)
This function does not interact with the database.
Process
Evaluate the Transaction Status If the Transaction Status = 3 ("Checked (Discrepancy)"), Call Send_Notification_To_Registry with Transferring Registry code to inform the Transferring Registry of the discrepancy. If the Transaction Type = 3 (External) Call Send_Notification_To_Registry with the Acquiring Registry code. If the Transaction Status = 2 ("Checked (No Discrepancy)"), Check to see if either party to the transaction is in a supplementary program. Call Trade_Scheme_Member for the Transferring Registry, and, if this is an External Transaction (3), the Acquiring Registry. If either function call returns success, call Send_To_STL. If the parties are not in a supplementary program, route the message as appropriate for the transaction type. For Transaction Type 3 (External Transfers), call Send_Proposal_To_Registry for the Acquiring Registry. For all other transaction types, call Send_Notification_To_Registry for the Transferring Registry.
Outputs
ResultIdentifier
Call(s)
Send_Notification_To_Registry Trade_Scheme_Member Send_To_STL Send_Proposal_To_Registry

Figure E12: Evaluate_Registry_Reconciliation_Message (Function)

Purpose
This function checks the reconciliation status of an incoming message from a registry and routes the message as appropriate.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Begin database transaction. Evaluate the reconciliation status: If "Initiated" (1), Call Validate_Totals If "Validated" (2), Call Forward_Totals_To_STL If "Totals Inconsistent" (3), Call Validate_Unit_Blocks If "Unit Blocks Inconsistent" (4) Call Accept_Audit_Trail If "STL Totals Inconsistent" (8), Call Forward_Unit_Blocks_To_STL If "STL Unit Blocks Inconsistent" (9), Call Forward_Audit_Trail_To_STL. When the process returns from one of the above functions, call Remove_From_Message_Queue and commit the database transaction.
Outputs
CheckResponseObject

(cont.)

Figure E12: Evaluate_Registry_Reconciliation_Message (Function) (cont.)

Call(s)
Validate_Totals Forward_Totals_To_STL Validate_Unit_Blocks Forward_Unit_Blocks_To_STL Forward_Audit_Trail_To_STL Accept_Audit_Trail Remove_From_Message_Queue

262 **Figure E13: Evaluate_ResponseObject (Function)**
263

Purpose
This function looks at the CheckResponseObject and determines if the transaction failed to meet the minimum checks required for further processing.
Inputs
CheckResponseObject
Table(s)
Response_Code
Process
Check the response codes collected in the CheckResponseObject. If any codes indicate a failure (Response_Code.Response_Type_Code = 0), then return HTTP SOAP response of failure and include response codes in CheckResponseObject indicating nature of failure. If there are no failures in the CheckResponseObject, then call the Evaluate_Transaction function to determine the next level of checks to perform.
Outputs
CheckResponseObject
Call(s)
Evaluate_Transaction

264

265 **Figure E14: Evaluate_STL_Reconciliation_Notice (Function)**
266

Purpose
This function checks the reconciliation status on incoming messages from the STL and processes them or routes them as appropriate.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Begin database transaction. Update the reconciliation status on the ITL with the status passed from the STL. Then evaluate the reconciliation status: If Initiated (1) Call Start_Reconciliation If "STL Totals Inconsistent" (8) Call Write_To_Reconciliation_Status Call Request_Unit_Blocks_From_Registry If "STL Unit Blocks Inconsistent" (9) Call Insert_Inconsistent_Block for each block Call Write_To_Reconciliation_Status Call Request_Audit_Trail_From_Registry If "STL Validated" (10) Call Write_To_Reconciliation_Status Call Send_Reconciliation_Notification_To_Registry Call Delete_Inconsistent_Block to free any blocks previously frozen If "STL Complete with Manual Intervention" (11) Call Write_To_Reconciliation_Status Call Send_Reconciliation_Notification_To_Registry. When the process returns from one of the above functions, commit the database transaction.

(cont.)

Figure E14: Evaluate_STL_Reconciliation_Notice (Function) (cont.)

Outputs
CheckResponseObject
Call(s)
Start_Reconciliation Request_Unit_Blocks_From_Registry Request_Audit_Trail_From_Registry Send_Reconciliation_Notification_To_Registry Delete_Inconsistent_Block Start_Reconciliation Write_To_Reconciliation_Status Close_Reconciliation_Action

Figure E15: Evaluate_Transaction (Function)

Purpose
This function determines what the next action should be in processing the transaction. This is accomplished by checking both the transaction status and the transaction type.
Inputs
TransactionObject
Table(s)
Process
Evaluate the transaction status code. If the status = "Proposed", then evaluate the transaction type. All transaction types are processed by the ITL (Validate_Proposal). After control is passed back to this function, determine where the TransactionObject is to be routed next (Determine_Route_For_Proposal). If the transaction is anything other than "Proposed", then the transaction is "in process." If the transaction type code = 3 (External) then initiate external processing (Process_External_Notification), else, all other transaction types (1, 2, 4, 5, 6, 7, 8 and 10) are processed (Process_Non-external_Notification).
Outputs
TransactionObject
Call(s)
Validate_Proposal Process_Non-external_Notification Process_External_Notification Determine_Route_For_Proposal

Figure E16: Excess Issuance for CDM Project (Function)

Purpose
This function will initiate a Notification that will inform an operational entity that excess CERs have been issued for a CDM Project and that the operational entity is required to cancel a specified quantity of units.
Inputs
Registry Code, Quantity of Units to be Cancelled
Table(s)
Notification
Process
Insert record into the Notification table for "Excess Issuance for CDM Project" (type 6) Inform operational entity via email of the Notification Type Code and the quantity of units that must be cancelled.
Outputs
Result_Identifier
Call(s)

Figure E17: Execute_Checks (Function)

Purpose
This function will execute each check contained in the CheckResponseObject. It will update the result of each check in the ResponseObject.
Inputs
CheckResponseObject, TransactionObject, RegistryUnitBlockObject, ITLUnitBlockObject
Table(s)
Process
For each check in the CheckResponseObject, perform check against the data in the RegistryUnitBlockObject. If a check returns a failure, record the failure of that check in the CheckResponseObject and call the Write_Block_History function.
Outputs
CheckResponseObject
Call(s)
Write_Block_History

276 **Figure E18: Finalise_Transaction (Function)**
277

Purpose
When a transaction is complete, this function is called to update the unit holding records of the ITL.
Inputs
TransactionObject
Table(s)
This function does not interact with the database.
Process
This function evaluates the transaction type in order to know how to update the Unit_Block table. If the Transaction Type Code = 1 (Issuance), the Insert_Unit_Block function is then called for every unit block involved in the transaction to add a new record to the Unit_Block table. If the Transaction Type Code = 2, 7, or 8 (Conversion, Carry-over, or Expiry Date Change), Delineate_Units is called with 0 for every block in the transaction to indicate that the blocks are free to be used in a trade. The Update_Unit_Block function is then called to change the unit type (for Conversion), the applicable commitment period (for Carry-over), or the expiry date (for Expiry Date Change). If the Transaction Type Code = 3, or 4 (External, Cancellation), Delineate_Units is called with 0 for every block in a transaction to indicate that the blocks are free to be used in a trade. The Update_Block_Ownership_Or_Account_Type function is then called to transfer ownership of the units to the acquiring party. If the Transaction Type Code = 5 (Retirement) and the Supplementary Transaction Type Code is blank or = 00, Delineate_Units is called with 0 for every block in a transaction to indicate that the blocks are free to be used in a trade. The Update_Block_Ownership_Or_Account_Type function is then called to transfer ownership of the units to the retirement account. If the Transaction Type Code = 5 (Retirement) and the Supplementary Transaction Type Code = 01 (EC Retirement), Delineate_Units is called with 0 for every block in the transaction to indicate that the blocks are free to be used in a trade. This transaction contains two types of blocks: 1) EC allowances to be "converted" back to AAUs on behalf of the EC; 2) Kyoto Units to be retired by transferring them to a retirement account. The EC allowances are identified by the lack of an Acquiring Account Type Code. For each of these blocks the Update_Unit_Block is called to modify the supplementary unit type. For each block of Kyoto units to be retired, the Update_Block_Ownership_Or_Account_Type function is called to transfer the units to the retirement account.

(cont.)

Figure E18: Finalise_Transaction (Function) (cont.)

Process
If the Transaction Type Code = 6 (Replacement), Delineate_Units is called with 0 for every block in the transaction to indicate that the blocks are free to be used in a trade. Next call Update_Block_Ownership_Or_Account_Type to complete the transfer to the replacement account. Finally, call Replace_Unit_Block to insert a record in the Replacement_Unit_Block table and establish the relationship between the two types of blocks. If the Transaction Type Code = 10 (Internal Transfer) and the Supplementary Transaction Type Code is blank or = 00, 02, (Allowance Surrender), or 53 (Allowance Allocation), Delineate_Units is called with 0 for every block in the transaction to check that the blocks are free to be used in a trade. Next call Update_Block_Ownership_Account_Type to update the account type. If the Transaction Type Code = 10 (Internal Transfer) and the Supplementary Transaction Type Code = 52 (Allowance Issue) or 55 (Allowance Correction), call Update_Unit_Block to update the Supplementary Unit Type code. If the Transaction Type Code = 10 (Internal Transfer) and the Supplementary Unit Type Code = 41 (Cancellation and Replacement) the transaction contains two types of blocks: 1) Kyoto units to be cancelled and 2) Kyoto Units to be transferred from an EC national holding account into an operating holding account. The units to be cancelled are identified by an Acquiring Account Type Code = 230 (Voluntary Cancellation Account). Update_Unit_Block_Ownership_or_Account_Type is called for each block to transfer ownership to the cancellation account. The blocks to be transferred back to the holding account are identified by a Transferring Account Type Code = 100 (Holding Account). For each of these blocks, Update_Unit_Block_Ownership_or_Account_Type is called to transfer ownership. For all transactions, Call Process_Running_Totals.
Outputs
Result Identifier
Call(s)
Delineate_Units Insert_Unit_Block Update_Unit_Block Update_Block_Ownership_Or_Account_Type Replace_Unit_Block Process_Running_Totals

278

279 **Figure E19: Forward_Audit_Trail_To_STL (Function)**
280

Purpose
This function writes to the message log and then calls the ReceiveAuditTrail Web service on the STL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ReceiveAuditTrail Web service on the STL.
Outputs
CheckResponseObject
Call(s)
Write_To_Message_Log

281

Figure E20: Forward_Totals_To_STL (Function)

Purpose
This function writes to the message log and then calls the ReceiveTotals Web service on the STL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ReceiveTotals Web service on the STL.
Outputs
CheckResponseObject
Call(s)
Write_To_Message_Log Receive_Totals

285 **Figure E21: Forward_Unit_Blocks_To_STL (Function)**
286

Purpose
This function writes to the message log and then calls the ReceiveUnitBlocks Web service on the STL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ReceiveUnitBlocks Web service on the STL.
Outputs
CheckResponseObject
Call(s)
Write_To_Message_Log Receive_Unit_Blocks

287

288 **Figure E22: Get_Block_ID (Function)**
289

Purpose

This function will search the unit block table for the serial numbers passed. If the entire range of serial numbers are found, this function returns true and the block ID(s) in which the range of numbers was found is also returned. If all the serial numbers are not found, this function returns false. The block ID(s) of any overlapping blocks are still returned. This is used by the reconciliation process for identifying inconsistent blocks.

Inputs

Unit Block ID

Table(s)

Unit_Block

Process

Step 1: Search for exact match

```
Select Block_ID
From Unit_Block
Where Originating_Party_Code = input originating party code
        and Start_Block = input start block
        and End_Block = input end block

If found
        add Block_ID to output Block ID's
        set result to True
else
        proceed to step 2.
```

Step 2: Search for single block that completely encompasses input block

```
Select Block_ID
From Unit_Block
Where Originating_Party_Code = input originating party code
        and Start_Block <= input start block
        and End_Block >= input end block

If found
        add Block_ID to output Block ID's
        set result to True
else
        proceed to step 3.
```

(cont.)

Figure E22: Get_Block_ID (Function) (cont.)

Process
Step 3: Search for all unit blocks that overlap with the input serial numbers Search Unit_Block for any block containing either the starting number or ending number. Order by serial number. Select Block_ID From Unit_Block Where Originating_Party_Code = input originating party code And ((Start_Block >= input start block and Start_Block <= input end block) or (End_Block <= input start block and End_Block <= input end block)) Order By Start_Block Check to see if the blocks returned are contiguous Initialize v_prev_end_block to Start_Block of first record returned + 1 For every record found, Add block ID to output block IDs If Start_Block <> v_prev_end_block + 1, the blocks are not contiguous set result to False and continue searching set v_prev_end_block = End_Block Skip to next record If the records returned are all contiguous, Return True else Return False
Outputs
Result Identifier, array of Block IDs
Call(s)
None

290

Figure E23: Get_Checks (Function)

Purpose
This function returns all the general checks and all of the checks associated with the transaction type which are associated with a check category. The checks are returned in the CheckResponseObject.
Inputs
TransactionObject, Check_Category_Code
Table(s)
Response_Catalog
Process
Retrieve from the Response_Catalog table all general checks and transaction-specific checks where Check_Catalog_Code matches input Check_Category_Code. Store checks to perform in CheckResponseObject.
Outputs
CheckResponseObject
Call(s)

294 **Figure E24: InitiateReconciliation (Web service)**
295

Purpose
This is the HTTP SOAP request that an STL will use to request the ITL to begin a reconciliation action for a specified registry and snapshot DateTime. The ITL will also use this request to inform the STL that the ITL has initiated a reconciliation for a specified registry and snapshot time on its own accord. This will allow the STL to create its own snapshot at the appropriate time. Once the request has passed preliminary checks and been written to the queue, an HTTP SOAP response of success or an HTTP SOAP fault of failure is returned. This Web service is hosted on both the ITL and the STL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the preliminary checks before writing message to the message queue.
Outputs
Result_Identifier
Call(s)
Preliminary_Checks Write_To_Message_Queue

296

297 **Figure E25: Insert_Inconsistent_Block (Function)**
298

Purpose
The ITL will call this function to insert a unit block to the Inconsistent_Block table for the registry. The Unit_Status and Unit_Status_DateTime will also be updated in the Unit_Block table. If the unit status is greater than zero, the unit cannot be traded. The unit status also keeps track of the reason the unit cannot be traded through a binary code system. When the unit is part of an ongoing transaction a value of 1 is added to the current unit status. When unit is part of an inconsistent block identified by ITL reconciliation a value of 2 is added to the current unit status. When the unit is part of an inconsistent block identified by STL reconciliation, a value of 4 is added to the current unit status. So, when the value of unit status is 1, 3, or 5, the unit is part of an ongoing transaction. When the value is 2 or 3, the unit is part of an inconsistent block identified by ITL reconciliation. When the value is 4 or 5, the unit is part of an inconsistent block identified by STL reconciliation. The unit status should reflect only one instance of a unit status, even if multiple processes have tied up the unit for the same reason.
Inputs
ReconciliationObject
Table(s)
Process
For each unit block in the UnitBlock array of the ReconciliationObject, insert a record in the Inconsistent_Block table Update unit_status in Unit_Block table If the unit_status <> 2 or 3, unit_status = unit_status + 2 Update Last_Action_Date in Unit_Block table with current DateTime.

(cont.)

Figure E25: Insert_Inconsistent_Block (Function) (cont.)

Outputs
ResultIdentifier
Call(s)

299 **Figure E26: Insert_Unit_Block (Function)**
300

Purpose
This function will insert a row into the Unit_Block table when a new unit block is created.
Inputs
TransactionObject, ITLUnitBlockObject
Table(s)
Unit_Block
Process
Append a record into the Unit_Block table for the unit block in the ITLUnitBlockObject associated with the TransactionObject.
Outputs
Result_Identifier
Call(s)

301

Figure E27: Lack_of_Certification_Report (Function)

Purpose
This function will set the freeze flag for all units associated with a project that has failed to submit a certification report. Once the freeze flag is set the units can no longer be traded. This function will also inform all affected registries about the unit freeze.
Inputs
Project ID
Table(s)
Process
Set freeze flag for all units associated with the specified project. Freeze all lCERs for that project Update Unit_Block Set Project_Freeze_Flag = 1 Where Project_ID = input Project ID Insert record into the Notification table for "Non-submission of Certification Report for CDM Project" (type 5). For each Holding_Registry_Code represented in the above result set, Insert record into Registry_Notification table. Call AcceptITLNotice on Holding Registry with the Notification Type Code and the identifying information of each block.
Outputs
Result_Identifier
Call(s)

Figure E28: Message_Age_Check (Function)

Purpose
This function will evaluate whether the message retrieved from the queue was received more than 24 hours earlier. If so, a response code is returned and the processing is terminated.
Inputs
Message DateTime
Table(s)
No interaction with the database
Process
If current time – Message DateTime > 24 hours, return response code 1301
Outputs
Result_Identifier
Call(s)

Figure E29: Message_Sequence_Check (Function)

Purpose
This function will check the data in the TransactionObject to ensure that the transaction status identified in the TransactionObject is in the expected sequence order. See Sections 5.4.6 and 5.4.7 for specifics on the sequence checks.
Inputs
TransactionObject
Table(s)
Response_Catalog, Transaction_Log, Transaction_Log_History
Process
For each check identified in Sections 5.4.6 and 5.4.7 (depending on whether the message is from a registry or an STL), evaluate the TransactionObject against the check. Update ResultIdentifier and ResponseCode in the CheckResponseObject with the result of the check.
Outputs
CheckResponseObject TransactionObject
Call(s)

Figure E30: Net_Source_Cancellation (Function)

Purpose
This function will initiate a Notification that will inform a party that LULUCF activities of a Party have resulted in a net source of emissions and that the Party is required to cancel a specified quantity of units..
Inputs
Registry Code, Quantity of Units to be Cancelled
Table(s)
Notification, Registry_Notification
Process
Insert record into the Notification table for "Net Source Cancellation" (type 1). Insert record into Registry_Notification table for the input registry indicating the quantity of units that must be cancelled. Call AcceptITLNotice on Input Registry with the Notification Type Code and the quantity of units that must be cancelled.
Outputs
Result_Identifier
Call(s)

Figure E31: Non-compliance Cancellation (Function)

Purpose
This function will initiate a Notification that will inform a party that it is in non-compliance with its emissions target and that the Party is required to cancel a specified quantity of units.
Inputs
Registry Code, Quantity of Units to be Cancelled
Table(s)
Notification, Registry_Notification
Process
Insert record into the Notification table for "Non-compliance Cancellation" (type 2). Insert record into Registry_Notification table for the input registry indicating the quantity of units that must be cancelled. Call AcceptITLNotice on input Registry with the Notification Type Code and the quantity of units that must be cancelled.
Outputs
Result_Identifier
Call(s)

316 **Figure E32: Notification_Requirement_Attainment_Check (Function)**
317

Purpose
This function will evaluate registries compliance with outstanding notification requirements. If a registry is found to have fulfilled the requirement, the notification status will be updated and a Notification Update will be sent to the registry indicating the requirement was met. If the registry has not fulfilled the requirement a Notification Update may be sent to remind the registry of the remaining requirement and by when the registry must fulfill the requirement.
Inputs
Table(s)
Notification, Registry_Notification
Process
1. Check for open Notifications in the Registry_Notification table. Select Notification ID, Notification Type, Registry Code, Project_Log_ID, Total Units as Requirement Level From Registry_Notification, Notification Where Registry_Notification.Notification_Status <> 3 (Completed) and Notification_Type_Code <> 9 (Notification Update) For each record found, 2. Get Notification Requirement Attainment Level. Select sum (Total Units) as Attainment Level From Transaction_Notification, Transaction_Log_History Where Registry Code = input Registry Code and Notification ID = input Notification ID and Transaction_Status_Code = 4 (Completed) 3. Compare Requirement Level with Attainment Level. If Attainment Level >= Requirement Level, Update Notification_Status to 3 (Completed). Call Notification_Update with Notification ID, Quantity of Units = 0, and a Comment indicating the notification requirement has been fulfilled.

(cont.)

Figure E32: Notification_Requirement_Attainment_Check (Function) (cont.)

Process
If Notification Type Code = 4 (Reversal of Storage for CDM Project), The remaining lCERs held by that Registry need to be unfrozen. Select Project_ID From Project Action Log Where Project_Log_ID = input Project Log ID Update Unit_Block Set Project_Freeze_Flag = 0 Where Project ID = input Project ID And Holding Registry Code = input Party Code If Attainment Level < Requirement Level, Call Notification_Update with Notification ID, Quantity of Units = (Requirement Level – Attainment Level), Action Due Date, and a Comment describing the outstanding requirement.
Outputs
Result_Identifier
Call(s)
Notification_Update

Figure E33: Notification_Update (Function)

Purpose
This function will send a Notification Update to a registry in order to inform the registry that the requirements of a Notification have been fulfilled or to update a registry's progress towards meeting the requirement. When a Notification Update is sent, a record is added to the Notification Table and therefore a new Notification ID is generated. When the record is added, however, the Parent_Notification_ID field will reference the original Notification ID. When the Notification Update is sent, the original Notification ID is sent, not the Notification ID generated for the Notification Update.
Inputs
Parent Notification ID, Party Code, Comment, Quantity of Units, Action Due Date, Project Log ID
Table(s)
Notification, Registry_Notification
Process
Append record to Notification table as follows: Parent_Notification_ID = input Parent Notification ID Notification Date = system DateTime Notification_Content = Comment Total_Units = input Quantity of Units Notification_Type_Code = 9 (Notification Update) Project_Log_ID = input Project Log ID Notification_Status_Code = 3 (Completed) Capture the Notification ID for new record. Append record to Registry_Notification table as follows: Registry_Code = input Registry Code Notification ID = New Notification ID Registry_Action_Due_Date = input Action Due Date Registry_Notification_Date = system DateTime Total_Units = input Quantity of Units Notification Status Code = 3 (Completed) Call AcceptITLNotice of the input Party with Comment System DateTime Notification Type Code = 9 Parent Notification ID Input Quantity of Units Input Action Due Date

(cont.)

Figure E33: Notification_Update (Function) (cont.)

Outputs
Result_Identifier
Call(s)

320

Figure E34: Outstanding_Unit_Cleanup (Function)

Purpose
Identify units still in holding accounts for a Commitment Period after the true-up period has expired. Units should be cancelled, retired, carried-over or used for replacement.
Inputs
Commitment Period ID
Table(s)
Process
Search the Unit_Block table for registries holding units from the specified Commitment Period. Select Holding_Registry_Code, Originating_Party_Code, Start_Block, End_Block From Unit_Block Where Applicable_Period_Code = input Commitment Period And Account_Type_Code = 100, 110, 120, or 121 If any records found, append record to the Notification table Unit Carry-over (type 8) For each Holding_Registry_Code represented in the above result set, Append record to Registry_Notification table. Call AcceptITLNotice on the Holding Registry with the Notification Type Code and the identifying information of each block.
Outputs
ResultIdentifier
Call(s)

Figure E35: Preliminary_Checks (Function)

Purpose
The ITL calls this function when a message is received through one of its Web services. This function will authenticate the sender of the message and check the version number. If the message contains a transaction, it will call another function to write the message to a file. If there are no problems, the message will be added to the message queue.
Inputs
TransactionObject, ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Registry_Authentication. Call Check_Version. If the version check returns failure, this function immediately returns failure. If this message contains a transaction, call Write_To_File to record the contents of the message. If there is no problem in the above process, call the Write_To_Message_Queue function.
Outputs
Result_Identifier
Call(s)
Registry_Authentication Check_Version Write_To_File Write_To_Message_Queue

Figure E36: Process_External_Notification (Function)

Purpose
This function processes a transaction after an update has been received regarding an External Transaction (transaction type 3). This function will update the status at the ITL, and route the transaction to the next party based on the transaction status.
Inputs
TransactionObject
Table(s)
Process
The first step of this function is to update the transaction status at the ITL by calling the Write_Transaction_Status function. Once complete, this function evaluates the transaction status. If the Transaction Status = 4 ("Completed"), the transaction was completed at the Transferring Registry and the ITL calls Finalise_Transaction to update its own records. If either party to the transaction, as determined by calling Trade_Scheme_Member, is in a supplementary program, Send_To_STL is also called. If the Transaction Status = 5 ("Terminated"), this function simply calls Delineate_Units with 0 to indicate that the units involved in the transaction are free to be used in another transaction. In the same manner as above, Send_To_STL is also called if either party to the transaction is in a supplementary program. If the Transaction Status = 6 ("Rejected"), the Acquiring Registry refused the transaction. Send_Notification_To_Registry for the Transferring Registry is called to inform that party of the rejection. If either party is in a supplementary program, Send_To_STL is also called. If the Transaction Status = 8 ("Accepted"), the Acquiring Registry approved the transaction. Call Send_Notification_To_Registry to inform the Transferring Registry of the Acquiring Registry's decision.
Outputs
CheckResponseObject

(cont.)

Figure E36: Process_External_Notification (Function) (cont.)

Call(s)
Write_Transaction_Status Send_Notification_To_Registry Trade_Scheme_Member Send_To_STL Finalise_Transaction Delineate_Units

Figure E37: Process_Non-external_Notification (Function)

Purpose
After the ITL receives an update regarding any transaction except an External Transaction (transaction type 3), this function will update the status at the ITL, and route the transaction to the next party based on the transaction status.
Inputs
TransactionObject
Table(s)
Process
The first step of this function is to update the transaction status at the ITL by calling the Write_Transaction_Status function. Once complete, this function evaluates the transaction status. If the Transaction Status = 4 ("Completed"), the transaction was completed at the Transferring Registry, and the ITL calls Finalise_Transaction to update its own records. If the transferring registry, as determined by calling Trade_Scheme_Member, is in a supplemental program, Send_To_STL is also called. If the Transaction Status = 5 ("Terminated"), this function simply calls Delineate_Units with 0 to indicate that the units involved in the transaction are free to be used in another transaction. In the same manner as above, Send_To_STL is also called if the Transferring Registry is in a supplemental program.
Outputs
CheckResponseObject
Call(s)
Trade_Scheme_Member Send_To_STL Finalise_Transaction Delineate_Units Replace_Unit_Block Write_Transaction_Status

Figure E38: Process_Running_Totals (Function)

Purpose
This function will adjust all appropriate running totals whenever a transaction is finalised.
Inputs
TransactionObject
Table(s)
Notification, Registry Notification, Registry Unit Sum, Unit Block
Process
If Transaction Type Code = 1 (Issuance), 1. Calculate the number of units to be issued. 2. Update the Commitment Period holdings. Call Update_Running_Total with number of units to be issued Computation Type Code = 2 (Commitment Period Holdings) Registry Code = Acquiring Registry Code Unit Type = issued unit type Time Period = CMP Time Period Value = current Commitment Period 3. Update the Unit Type Issued Level. If Unit Type Code = 1 (AAU) or 2 (RMU), Call Update_Running_Total with number of units to be issued Computation Type Code = 1 (Unit Type Issuance Level) Registry Code = acquiring registry code Unit Type = issued unit type Time Period = CMP Time Period Value = current Commitment Period

(cont.)

Figure E38: Process_Running_Totals (Function) (cont.)

Process
4. Update Project Issued Level. If Unit Type Code = 5 (CER), 6 (tCER), or 7 (lCER), Get Project Log ID for issuance. There could be more than one issuance in the Project Log. Associate this issuance with the most recent. Select Max(Project Log ID) From Project Action Log Where Project ID = input Project ID And Project Action Code = 4 (Issuance) Call Update_Running_Total with number of units to be issued Computation Type Code = 9 (Project Issuance Level) registry code = acquiring registry code Project Log ID = input Project Log ID If Transaction Type Code = 2 (Conversion), Update the Track 2 ERU Conversion Level. 1. Calculate the number of Track 2 ERUs to be converted. 2. Update Conversion Level running total. Call Update_Running_Total with number of Track 2 ERUs to be converted Computation Type Code = 7 (Converted Level) Registry Code = Transferring Registry Code Unit Type Code = input unit type Time Period = CMP Time Period Value = current Commitment Period If Transaction Type Code = 3 (External), 1. Calculate the number of units to be transferred by unit type. 2. Update the Commitment Period Holdings.

(cont.)

Figure E38: Process_Running_Totals (Function) (cont.)

Process
For each unit type total, a. Modify the CP Holdings Level for the transferring registry. Call Update_Running_Total with number of units to be transferred Computation Type Code = 2 (Commitment Period Holdings) Registry Code = Transferring Registry Code Unit Type = input unit type Time Period = CMP Time Period Value = current Commitment Period b. Modify the CP Holdings Level for the acquiring registry. Call Update_Running_Total with number of units to be transferred Computation Type Code = 2 (Commitment Period Holdings) Registry Code = Acquiring Registry Code Unit Type = input unit type Time Period = CMP Time Period Value = current Commitment Period 3. Update ERU Track 2 First Time Transfer Level. If unit type = 3 (ERU) or 4 (ERU converted from RMU) and Track = 2 and Transfer Flag = F, Call Update_Running_Total with number of units to be transferred Computation Type Code = 11 (Track 2 ERU First-time transfer level) Registry Code = Transferring Initiating Registry Code Time Period = CMP Time Period Value = current Commitment Period 4. Update Flag for ERU Track 2 First Time Transfer. If unit type = 3 (ERU) or 4 (ERU converted from RMU) and Track = 2 and Transfer Flag = F, Update Unit Block Set Transfer Flag = T

(cont.)

Figure E38: Process_Running_Totals (Function) (cont.)

Process
If Transaction Type Code = 4 (Cancellation), 1. Update the Commitment Period Holdings. Calculate the number of units to be transferred by unit type. For each unit type total, Modify the CP Holdings Level for the transferring registry Call Update_Running_Total with number of units to be transferred Computation Type Code = 2 (Commitment Period Holdings) Registry Code = transferring registry code Unit Type = input unit type Time Period = CMP Time Period Value = current Commitment Period If Transaction Type Code = 5 (Retirement), 1. If Unit Type Code 6 (tCER) or 7 (lCER), Update tCER/lCER Retirement Level. Call Update_Running_Total with number of units to be retired Registry Code = Transferring Registry Code Computation Type Code = 10 (Retirement level) Unit Type = retired unit type Time Period = CMP Time Period Value = current Commitment Period If Transaction Type Code = 7 (Carry-over), 1. Update Carry-over Level for each unit type. Calculate the number of units to be carried-over by unit type. For each unit type total, Modify the Carry-over Level for the transferring registry. Call Update_Running_Total with number of units to be carried-over Registry Code = Transferring Registry Code Computation Type Code = 8 (Carry-over Level) Unit Type = input unit type Time Period = CMP Time Period Value = current Commitment Period

(cont.)

Figure E38: Process_Running_Totals (Function) (cont.)

Outputs
Result_Identifier
Call(s)
Update_Running_Total

335 **Figure E39: ProvideAuditTrail (Web service)**
336

Purpose
This is the HTTP SOAP request that the STL uses to initiate the request for the ITL to request audit trail data from a registry. If the SOAP request is not accepted by the ITL, a Registry HTTP SOAP fault is returned. Once the request has passed reconciliation message checks, an HTTP SOAP response is returned. The ITL will relay the request to the appropriate registry. This Web service is hosted by the ITL and the Registry.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the Preliminary_Checks function to validate the incoming message and add the message to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

337

338 **Figure E40: ProvideTotals (Web service)**
339

Purpose
This is the HTTP SOAP request that the STL uses to initiate a request for total number of units held by each account at a registry. If the SOAP request is not accepted, an HTTP SOAP fault is returned. Once the request has passed reconciliation message checks and been written to the queue, an HTTP SOAP response is returned. The ITL relays the request to the appropriate registry. This Web service is hosted on the ITL and the Registry.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the Preliminary_Checks function to validate the incoming message and add the message to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

340

Figure E41: ProvideUnitBlocks (Web service)

Purpose
This is the HTTP SOAP request that the STL uses to initiate a request for the unit blocks held by an account at a registry. If the SOAP request is not accepted, an HTTP SOAP fault is returned. Once the request has passed reconciliation message checks and been written to the queue, an HTTP SOAP response is returned. The ITL relays the request to the appropriate registry. This Web service is hosted by the ITL and by the Registry.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the Preliminary_Checks function to validate the incoming message and add the message to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

344 **Figure E42: ReceiveAuditTrail (Web service)**
345

Purpose
This is the HTTP SOAP request that registries use to send a transaction history for inconsistent blocks. If the SOAP request is not accepted, an HTTP SOAP fault is returned. Once the request has passed reconciliation message checks and been written to the queue, an HTTP SOAP response is returned. This Web service is hosted by the ITL.
Inputs
ReconciliationObject, TransactionObject
Table(s)
This function does not interact with the database.
Process
This function calls the Preliminary_Checks function to validate the incoming message and write it to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

346

Figure E43: ReceiveReconciliationResult (Web service)

Purpose
This is the HTTP SOAP request that the STL uses to inform the ITL that it has finished processing a reconciliation action. If the SOAP request is not accepted, an HTTP SOAP fault is returned. Once the request has passed reconciliation message checks and been written to the queue, an HTTP SOAP response is returned. If necessary, the ITL relays the message to the appropriate registry. This Web service is hosted on the ITL and the Registry.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This Web service calls the Preliminary_Checks function to validate the incoming message and add the message to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

350 **Figure E44: ReceiveTotals (Web service)**
351

Purpose
This is the HTTP SOAP request that registries use to send the ITL the total number of units held by each account or account type at that registry. If the SOAP request is not accepted, an HTTP SOAP fault is returned. Once the request has passed reconciliation message checks and been written to the queue, an HTTP SOAP response is returned. This Web service is hosted on the ITL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This function calls the Preliminary_Checks function to validate the incoming message and write it to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

352

353 **Figure E45: ReceiveUnitBlocks (Web service)**
354

Purpose
This is the HTTP SOAP request that registries use to send the total unit blocks held by each account or account type at that registry to the ITL. If the SOAP request is not accepted, an HTTP SOAP fault is returned. Once the request has passed reconciliation message checks and been written to the queue, an HTTP SOAP response is returned. This Web service is hosted on the ITL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
This function calls the Preliminary_Checks function to validate the message and add it to the queue.
Outputs
CheckResponseObject
Call(s)
Preliminary_Checks

355

356 **Figure E46: Reconciliation_Data_Integrity_Check (Function)**
357

Purpose
This function will check the data in the ReconciliationObject and TransactionObject as appropriate to assure that it meets the minimum requirements to begin processing the incoming message. See Section 6.2.4 for a list of all data integrity checks performed by this function.
Inputs
ReconciliationObject, TransactionObject
Table(s)
Response_Catalog, Response_Log
Process
For each check identified in Section 6.2.4 evaluate the ReconciliationObject against the check. For each check, collect the appropriate response code returned and store in the CheckResponseObject.
Outputs
CheckResponseObject
Call(s)

358

359 **Figure E47: Reconciliation_Failure_Notification (Function)**
360

Purpose
The ITL calls this function to notify a registry or the STL that a reconciliation message sent by that party failed one or more message validation check.
Inputs
ReconciliationObject, CheckResponseObject
Table(s)
Process
Evaluate from where the message originated. If it was an STL, call Send_Reconciliation_Notification_To_STL. If it was a registry, call Send_Reconciliation_Notification_To_Registry.
Outputs
ResultIdentifier ReconciliationObject CheckResponseObject
Call(s)
Send_Reconciliation_Notification_To_STL Send_Reconciliation_Notification_To_Registry

361

Figure E48: Reconciliation_Message_Sequence_Check (Function)

Purpose
This function will check the data in the ReconciliationObject to assure that the reconciliation status identified is the expected sequence order. See Sections 6.2.5 and 6.2.6 for specifics on the sequence checks.
Inputs
ReconciliationObject
Table(s)
Response_Catalog, Response_Log
Process
For each check identified in Sections 6.2.5 and 6.2.6 evaluate the ReconciliationObject against the check. Modify the result indicator in the CheckResponseObject to indicate the success or failure of a check.
Outputs
CheckResponseObject
Call(s)

365 **Figure E49: Remove_From_Message_Queue (Function)**
366

Purpose
This function will remove a message from the processing queue when processing of the original message has completed without error.
Inputs
Table(s)
This function does not interact with the database.
Process
For the message being processed, remove message from the processing queue.
Outputs
Result_Identifier
Call(s)

367

368 **Figure E50: Replace_Unit_Block (Function)**
369

Purpose
This function will insert a new record in the Replacement_Unit_Block table in order to establish a relationship between the units being replaced and the replacing units.
Inputs
TransactionObject
Table(s)
Replacement_Unit_Block
Process
Retrieve the Replaced Block_ID for the block(s) to be replaced. Note that for a replacement transaction, it should not be necessary to split this block. Select Block_ID from the Unit_Block table where: • Holding_Registry_Code = replaced units registry code • Account_Type_Code = replaced units account type • Start_Block = replaced units start block • End_Block = replaced units end block Retrieve the Replacing Block_ID for the block(s) that replaces the old block. Select Block_ID from the Unit_Block table where: • Holding_Registry_Code = replacing units registry code • Account_Type_Code = replacing units account type • Start_Block = replacing units start block • End_Block = replacing units end block Append to the Replacement_Unit_Block table as follows: • Block_ID = Replaced Block_ID • Replacement_Block_ID = Replacing Block_ID • Replacement_DateTime = current time
Outputs
Result_Identifier
Call(s)

370 **Figure E51: Request_Audit_Trail (Function)**
371

Purpose
This function will call the ProvideAuditTrail Web service to request a transaction history for specified blocks.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ProvideAuditTrail Web service of the registry with parameters from the ReconciliationObject.
Outputs
ReconciliationObject
Call(s)
Write_To_Message_Log ProvideAuditTrail

372

373 **Figure E52: Request_Totals_From_Registry (Function)**
374

Purpose
This function will call the registry's Web service to request unit/type account type totals for reconciliation. If the "by Account" parameter is passed, then data are to be grouped by AccountIdentifier.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ProvideTotals Web service of the registry with parameters set in the ReconciliationObject to request the unit holding count by account type and unit type.
Outputs
ReconciliationObject ResponseObject
Call(s)
Write_To_Message_Log ProvideTotals

375

376 **Figure E53: Request_Unit_Blocks_From_Registry (Function)**
377

Purpose
This function will call the registry's Web service to request unit blocks for reconciliation for an account type and unit type.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ProvideUnitBlocks Web service of the registry with parameters set in the ReconciliationObject.
Outputs
CheckResponseObject
Call(s)
Write_To_Message_Log ProvideUnitBlocks

378

Figure E54: Retrieve_Message_From_Queue (Function)

Purpose
This function retrieves the first message in the queue to begin processing the next set of checks.
Inputs
Table(s)
Process
Retrieves the oldest time stamped item in the queue and stores it to a transaction object in preparation for further checks. The date and time that the message is placed in the message queue is the official date and time of record for the transaction or reconciliation message or status.
Outputs
TransactionObject ReconciliationObject
Call(s)
Write_To_Message_Log

382 **Figure E55: Reversal_of_Storage (Function)**
383

Purpose
Upon receiving instructions from the CDM Executive Board, this function will freeze all units associated with a project. It will then calculate the number of units each registry must retire due to a reversal of greenhouse gas storage for a project. This function will then inform each affected registry of the number of units that must be cancelled or replaced.
Inputs
Project ID, Percentage
Table(s)
Unit_block, Notification, Registry_Notification, Project
Process
Set freeze flag for all units associated with the project. Freeze all lCERs for that project Update Unit_Block Set Project_Freeze_Flag = 1 Where Project_ID = input project ID If any records found, insert record into the Notification table for "Reversal in Storage for CDM Project Notification" (type 4). Identify the registries holding affected units and calculate the number of units each registry must replace or cancel. Select Holding_Reg_Code, sum(Block_End - Block_Start + 1) as RegistrylCERTotal From Unit_Block Where Project_ID = input Project ID Group By Holding_Registry_Code For each record returned in the above resultset, calculate the number of lCERs to unfreeze at each registry. Number of lCERs to cancel or replace = input Percentage * RegistrylCERTotal If number of lCERs to unfreeze not an integer, round up to next integer. Append record to Registry_Notification table with the number of lCERs to cancel or replace.

(cont.)

Figure E55: Reversal_of_Storage (Function) (cont.)

Process
Call AcceptITLNotice on Holding Registry with the Notification Type Code and the number of units that must be cancelled or replaced.
Outputs
Result_Identifier
Call(s)

384

385 **tiiiFigure E56: Send_Notification_To_Registry (Function)**
386

Purpose
This function sends a HTTP SOAP request with the contents of the TransactionObject and the ResponseObject to a registry. This function is used to send notification to the Transferring Registry that either the proposal could not be processed due to discrepancies or no discrepancies were found. It is also used to notify the Acquiring Registry in an External transaction when a discrepancy is found in the Initiating Registry's proposal.
Inputs
TransactionObject, CheckResponseObject, RegistryCode
Table(s)
Process
Invoke the AcceptNotification Web service at the appropriate registry with the contents of the TransactionObject and the CheckResponseObject. This function waits for a HTTP SOAP response from the registry indicating the message was received successfully, at which time the request is logged to the Message_Log table.
Outputs
ResultIdentifier
Call(s)
Write_To_Message_Log

387

388 **Figure E57: Send_Proposal_To_Registry (Function)**
389

Purpose
This function will call the AcceptProposal Web service on the Acquiring Registry and add an entry in the message log.
Inputs
TransactionObject
Table(s)
Invoke the AcceptProposal Web service at the appropriate registry with the contents of the TransactionObject. This function waits for a HTTP SOAP response from the registry indicating the message was received successfully, at which time the request is logged to the Message_Log table.
Outputs
Result_Identifier
Call(s)
Write_To_Message_Log

390

391 **Figure E58: Send_Reconciliation_Notification_To_Registry (Function)**
392

Purpose
This function will call the registry's Web service to receive updates about an ongoing reconciliation action.
Inputs
ReconciliationObject CheckResponseObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ReceiveReconciliationResult Web service on the registry.
Outputs
ReconciliationObject CheckResponseObject
Call(s)
Write_To_Message_Log

393

394 **Figure E59: Send_Reconciliation_Notification_To_STL (Function)**
395

Purpose
This function calls the ReceiveReconciliationResult Web service on the STL.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call ReceiveReconciliationResult Web service on an STL.
Outputs
ReconciliationObject CheckResponseObject
Call(s)
Write_To_Message_Log

396

397 **Figure E60: Send_Reconciliation_Snapshot_DateTime (Function)**
398

Purpose
This function will call the InitiateReconciliation Web service on an STL in order to inform the STL of the snapshot DateTime of an upcoming reconciliation for a member registry.
Inputs
ReconciliationObject
Table(s)
This function does not interact with the database.
Process
Call Write_To_Message_Log Call InitiateReconciliation Web service method on destination STL.
Outputs
Result_Identifier
Call(s)
Write_To_Message_Log

399

400 **Figure E61: Send_To_STL (Function)**
401

Purpose
This function sends an HTTP SOAP request with the contents of the TransactionObject to an STL and logs the action to the Routing_Log.
Inputs
TransactionObject
Table(s)
Routing_Log
Process
This function will determine which Web service to invoke on the STL. If the transaction status = "Proposed" then the AcceptProposal will be called, otherwise the AcceptNotification will be called. This function waits for an HTTP SOAP response from an STL after which it writes a record to the Routing_Log indicating a transaction was forwarded to an STL for further processing.
Outputs
Result_Identifier
Call(s)
Write_To_Routing_Log

402

403 **Figure E62: Snapshot_Registry_Data (Function)**
404

Purpose
This function will take a "snapshot" of the holdings in the unit_block table for a specified registry. This function will be called precisely at the snapshot time and will retrieve all the unit blocks held at a specified registry. In addition to the identifying information for the unit block, the account type and unit type are also retrieved. The ITL will perform reconciliation validation operations on the resultant dataset.
Inputs
ReconciliationObject
Table(s)
Unit_Block
Process
Select Holding_Registry_Code, Account_Type_Code, AccountCommitmentPeriod, Unit_Type_Code, Originating_Party, Period_Code, Start_Block, End_Block From Unit_Block, Government_Account Where Holding_Registry_Code = code for registry subject to reconciliation The returned dataset will be inserted into a snapshot unit table for reconciliation validation processes.
Outputs
Result_Identifier
Call(s)

405

406 **Figure E63: Split_Block (Function)**
407

Purpose
This function will search the Unit_Block table for a block involved in a transfer, and split the existing block if necessary so that only the units involved in a transaction will be affected. The new blocks created will have the same characteristics (account type, block type, etc.) as the parent block. There are four scenarios to account for: • Block to be transferred matches serial numbers exactly • Begin numbers match • End numbers match • Neither numbers match, but new block needs to be created.
Inputs
RegistryUnitBlockObject
Table(s)
Unit_Block
Process
Step 1: Search the Unit_Block table for records that exactly match the block to be transferred. Search for: • Originating_Party_Code = input originating party code • Start_Block = input start block • End_Block = input end block If found, there is no need to split the block. Store Block_ID to array of Block ID. Function ends. If not found, proceed to Step 2 Step 2: Search the Unit_Block table for a single record where the starting serial numbers match, and the end number in the Unit_Block table is greater than the end number of the block to be transferred. Search for: • Originating_Party_Code = input originating party code • Start_Block = input start block • End_Block > input end block

(cont.)

Figure E63: Split_Block (Function) (cont.)

Process
If found, split the block into two blocks by modifying the beginning number of existing block in the Unit_Block table (Set Start_Block = input end block + 1) insert record in Unit_Block table for the block to be transferred (Call Split_Related_Blocks) Store Block IDs to array of Block ID If not found, proceed to Step 3 Step 3: Search the Unit_Block table for a single row where the end serial number matches the end number of the block to be traded, and the Unit_Block start number is less than the start number of the block to be traded. Search for: • Originating_Party_Code = input party code • Start_Block < input start block • End_Block = input end block If found, split the block into two blocks by modifying the beginning number of existing block in the Unit_Block table (Set End_Block = input start block - 1) insert record in Unit_Block table for the new block (Call Split_Related_Blocks) Store Block_IDs to array of Block ID If not found, proceed to Step 4 Step 4: Search Unit_Block table for a single row where the block to be transferred is completely contained by an existing block. Search for: • Originating_Party_Code = input originating party code • Start_Block < input start block • End_Block > input end block

(cont.)

Figure E63: Split_Block (Function) (cont.)

Process
If found, split the block into three blocks by modifying the ending number of existing block in the Unit_Block table (Set End_Block = input Start Block - 1) Insert record in Unit_Block table for the new block to be transferred Call Insert_Unit_Block with input Start_Block and input End_Block. Insert record in Unit_Block table where Start_Block = input End_Block + 1 and End_Block = original End_Block Call Split_Related_Blocks Store Block_IDs to array of Block IDs If not found, continue to Step 5 Step 5: Search the Unit Block table for the block that contains the following: Create record set of Unit_Block where • Originating_Party_Code = input originating party code • Maximum (Start_Block) ≤ input start block This is Block A. Split Block A into two blocks by: Modifying the ending number of existing block in the Unit_Block table (set End_Block = input start block - 1) Insert record into Unit_Block table for the part of the block to be transferred Call Insert Unit Block with input start block and Block A's original end number Call Split_Related_Blocks Add new block to Block A record set and remove original Block_ID Create record set of Unit_Block where • Originating_Party_Code = input originating party code • Minimum (End_Block) ≥ input end block This is Block B. Split the block into two blocks by: Modifying the starting number of existing block in the Unit_Block table (set Start_Block = input end block + 1) Insert record into Unit_Block table for the part of the block to be transferred Call Split_Related_Block Add new block to Block B record set and remove original Block_ID Create record set Block C where Block_IDs are between record set A and B. Create instance of ITLUnitBlockObject for every unique Block_ID in Block A + B + C record sets

(cont.)

Figure E63: Split_Block (Function) (cont.)

Outputs
Result Identifier, ITLUnitBlockObject
Call(s)
Write_To_Inconsistent_Block Split_Related_Blocks

408

409 **Figure E64: Split_Related_Blocks (Function)**
410

Purpose
This function splits related blocks in the Replacement_Unit_Block table and the Inconsistent_Block table when a split has been made in the Unit_Block table.
Inputs
Original_Block_ID, New_Block_ID
Table(s)
Inconsistent_Block Replacement_Unit_Block
Process
Select from Inconsistent_Block table where Block_ID = Original_Block_ID. If found, insert new record where • Recon_Log_ID = Recon_Log_ID • Initiating_Party_Code = Initiating_Party Code • Block_ID = New_Block_ID Select from Replacement_Unit_Block table where Replacement_Block_ID = Original_Block_ID If found, insert new record where • Replacement_Block_ID = New_Block_ID • Block ID = Block_ID • Replacement_DateTime = Replacement_DateTime Note that the Block_ID does not split as the Replacement_Block_ID has. The total units of all the Replacement_Block_IDs associated with the one Block_ID should be equal.

411

412 **Figure E65: Start_Reconciliation (Function)**
413

Purpose
This function opens a reconciliation action and requests the registry subject to reconciliation to send data to the ITL. The data requested will depend on the phase for which reconciliation was requested. Prior to opening a new reconciliation, this function will check if there is an ongoing reconciliation action for the specified registry. If an ongoing reconciliation action is found, the new reconciliation action will be denied and the Party that requested the reconciliation will be notified.
Inputs
ReconciliationObject Phase
Table(s)
Reconciliation_Log
Process
Call Write_To_Reconciliation_Log to record the new reconciliation action. Call the Ongoing Reconciliation check to verify that another reconciliation action for this registry is not ongoing. Call the Snapshot DateTime Validity check to verify that snapshot time is not in the future. If any check fails, Call Write_To_Reconciliation_Status to update the status to 9 ("Reconciliation Start Denied"). Call Close_Reconciliation_Action to update the end date in the Reconciliation Log table. The Response Code is 4001. If the source of the reconciliation request was an STL, call Send_Reconciliation_Notification_To_STL with the response code. Call Trade_Scheme_Member to determine if the registry is part of an STL. If the registry is part of an STL, Call Send_Reconciliation_Snapshot_DateTime. If all checks pass, Call Write_To_Reconciliation_Status with status of 1 ("Initiated"). If requested phase = 1, Call Request_Totals_From_Registry.

(cont.)

Figure E65: Start_Reconciliation (Function) (cont.)

Process
If requested phase = 2, Call Request_Unit_Blocks_From_Registry. If requested phase = 3, Call Request_Audit_Trail_From_Registry.
Outputs
CheckResponseObject
Call(s)
Write_To_Reconciliation_Log Write_To_Reconciliation_Status Request_Totals_From_Registry Request_Unit_Blocks_From_Registry Request_Audit_Trail_From_Registry Send_Reconciliation_Notification_To_STL Close_Reconciliation_Action Trade_Scheme_Member

414

Figure E66: Time_Sync (Function)

Purpose
This function will run on a periodic basis to check the system time of the registries that are a part of the trading network.
Inputs
RegistryCode
Table(s)
System_Log
Process
Call ProvideTime web service method on the input registry. Insert new record in the System_Log table with • Registry Code • Registry Date and Time • ITL Date and Time
Outputs
Result_Identifier
Call(s)

418 **Figure E67: Trade_Scheme_Member (Function)**
419

Purpose
This function identifies whether the registry is a member of a specified trading scheme.
Inputs
Registry_Code, Trade_Scheme
Table(s)
Registry_Trading_Scheme
Process
Query the Registry_Trading_Scheme table to see if the Registry_Code and Trade_Scheme exist. If so, return true (1).
Outputs
Result_Identifier
Call(s)

420

421 **Figure E68: Transaction_Cleanup (Function)**
422

Purpose
This function runs every hour and checks for transactions that have not completed and for which the last message was initiated more than 24 hours ago. If found, the transaction is Cancelled.
Inputs
Table(s)
Transaction_Log, Transaction_Log_History
Process
In progress transactions will have a status of: • 1 - Proposed • 2 - Checked (No Discrepancy) • 3 - Checked (Discrepancy) • 6 - Rejected • 8 - Accepted • 9 - STL Checked (No Discrepancy) • 10 - STL Checked (Discrepancy) Every Hour: Select Transaction_ID, Transaction_Status_ID From Transaction_Log, Transaction_Log_History Where Transaction_Status_ID = (1,2,3,6,8,9,10) Transaction_Status_DateTime = most recent status for that Transaction_ID and Current Time – Transaction_Status_DateTime > 24 hours For each record found, Set the Response Code to 4016 – A transaction must be completed within 24 hours of the last action. This transaction has exceeded that limit and has been cancelled. Call Write_Transaction_Status with Transaction_ID and "Cancelled" (7) For each Unit Block involved in the transaction, call Delineate_Units with 0 so the units may be involved in another transaction.

(cont.)

Figure E68: Transaction_Cleanup (Function) (cont.)

Process
Call Send_Notification_To_Registry for the Initiating Registry. If Transaction Type is External (type 3), Call Send_Notification_To_Registry for the Acquiring Registry. Call Trade_Scheme_Member for the Initiating Registry and the Acquiring Registry (if applicable). If either registry is part of an STL, call Send_To_STL for the identified STL.
Outputs
Result_Identifier
Call(s)
Write_Transaction_Status Delineate_Units Send_Notification_To_Registry Send_To_STL

423

Figure E69: Update_Block_Ownership_Or_Account_Type (Function)

Purpose
This function will update the Holding_Registry_Code and Account_Type attributes of the Unit_Block table in order to complete a unit block transfer.
Inputs
TransactionObject, ITLUnitBlockObject
Table(s)
Unit_Block
Process
For each Block ID in the ITLUnitBlockObject, update the following: • Holding_Registry_Code = acquiring registry code • Account_Type_Code = acquiring account type
Outputs
Result_Identifier
Call(s)

427
428

Figure E70: Update_Running Total (Function)

Purpose
This function will update the Running Total and will insert a record in the running total history table.
Inputs
TransactionObject
Table(s)
Registry_Unit_Sum, Registry_Unit_Total_History
Process
Search for Existing Running Total. Select Registry_Unit_Sum_ID, Total Units as Previous Total From Registry Unit Sum Where Computation Type Code = input Computation Type Code and Registry Code = input Registry Code (apply following limiting criteria only if input values are provided) Unit Type = input Unit Type LULUCF Code = input LULUCF Code Time Period = input Time Period Time Period Value = input Time Period Value Notification ID = input Notification ID Project Log ID = input Project Log ID Data Source = input Data Source If not found, Append record to the Registry Unit Sum Table in accordance with the values passed into this function. Append record into the Registry Unit Total History If found, Update Registry Unit Sum Table Set Total Units = Total Units + number of units to be issued Last Action Date = current DateTime Where Computation Type Code = input Computation Type Code and Registry Code = input Acquiring Registry Code (apply following limiting criteria only if input values are provided) Unit Type = input Unit Type LULUCF Code = input LULUCF Code

(cont.)

Figure E70: Update_Running Total (Function) (cont.)

Process
Time Period = input Time Period Time Period Value = input Time Period Value Notification ID = input Notification ID Project Log ID = input Project Log ID Data Source = input Data Source Append record into the Registry Unit Total History
Outputs
Result_Identifier
Call(s)

429
430
431

Figure E71: Update_Unit_Block (Function)

Purpose
This function will update a record in the Unit_Block table. The only fields that this function updates are the Holding_Registry_Code, Account_Type_Code, Unit_Type_Code, the Applicable_Commitment_Period, and Supplementary_Unit_Type.
Inputs
ITLUnitBlockObject, RegistryUnitBlockObject
Table(s)
Unit_Block
Process
If the transaction type is External (3), update the Holding_Registry_Code and Account_Type_Code to the registry and account type of the acquiring party. If transaction type is Carry-over (4), update the Applicable_Commitment_Period to the current commitment period. If the transaction type is Conversion (2), and the unit was not previously (2) RMU, update the Unit_Type to (3) for ERU. Update the Project Identifier. If the transaction type is Conversion (2), and the previous unit type was (2) RMU, update the Unit_Type to (4) for ERU converted from RMU. Update the Project Identifier. If the transaction type is Cancellation (4) or Retirement (5), update the Account_Type to the account type of the acquiring party. If the transaction type is Expiry Date Change (8), update the Expiry Date. If the transaction type is Internal/Supplementary (10) and the Supplementary Transaction Code=52 then the Supplementary_Unit_Type is updated to the appropriate code for the new Supplementary Transaction Type Code.
Outputs
Result_Identifier
Call(s)

435 **Figure E72: Validate_Proposal (Function)**
436

Purpose
This function will enter a transaction in the ITL, evaluate the transaction based on its transaction type, accumulate all the appropriate responses, update the transaction status, and freeze the units involved in the transaction.
Inputs
TransactionObject
Table(s)
This function does not interact with the database.
Process
Set Select for update on database transaction. Call Delineate_Units with 1 to indicate that the unit blocks are involved in a transaction. Commit database transaction. Set Select for Update on database transaction. Record the transaction by calling the Write_Transaction procedure followed by Write_Transaction_Block for each unit block involved in the transaction. Set the transaction status to "Proposed" (1) by calling Write_Transaction_Status. If Notification ID not null, append record to the Transaction_Notification table with • Transaction ID • Notification ID • Party Code of the registry responding to a Notification • Number of units processed in response to Notification Once recorded, validate the transaction against the business rules. Call Get_Checks, followed by Execute_Checks to retrieve and execute the business checks for this transaction. Get_Checks is first called to return checks that are general and apply to all transactions, and then again to return checks that are specific to the current transaction type. As each unit block is evaluated, call Write_Block_History to record the problem if a check is failed. If any of the checks fail, call Write_Transaction_Status with 3 for "Checked (Discrepancy)." If no checks fail, call that same function with 2 for "Checked (No Discrepancy)."

(cont.)

Figure E72: Validate_Proposal (Function) (cont.)

Process
Finally, no matter what the Transaction Status, call Delineate_Units with 1, to set the Unit_Status in the Transaction_Block table and prevent the units from being involved in another transaction until this transaction is complete or the discrepancy is resolved. Commit database transaction.
Outputs
Result_Identifier, CheckResponseObject
Call(s)
Write_Transaction Write_Transaction_Block Write_Transaction_Status Get_Checks Execute_Checks Write_Block_History Delineate_Units

437

438 **Figure E73: Validate_Totals (Function)**
439

Purpose
This function will compare each unit block sent by the registry against the snapshots of the ITL records previously taken. If blocks do not match, they will be marked as inconsistent.
Inputs
Table(s)
Process
Call Account Type/Unit Type Totals check (6410). If result = failure, Call Write_To_Reconciliation_Status with a status of 3 ("ITL Totals Inconsistent"). Continue to next step of reconciliation. Call Request_Unit_Blocks_From_Registry If result = pass, Call Write_To_Reconciliation_Status with a status of 2 ("ITL Validated"). Call Trade_Scheme_Member to determine if registry is part of a supplementary program. If part of a supplementary program, Call Request_Totals_By_Account. else Notify registry of successful reconciliation. Call Send_Reconciliation_Notification_To_Registry. Clean up inconsistent blocks, if any, now that a reconciliation has been performed. Select Inconsistent_Block_ID From Inconsistent_Block Where Originating_Party_Code = code for registry subject to this reconciliation action. For each item found, Call Delete_Inconsistent_Block.

(cont.)

Figure E73: Validate Totals (Function) (cont.)

Outputs
Call(s)
Write_To_Reconciliation_Status Request_Unit_Blocks_From_Registry Trade_Scheme_Member Request_Totals_By_Account Delete_Inconsistent_Block Send_Reconciliation_Notification_To_Registry

440
441

442 **Figure E74: Validate_Unit_Blocks (Function)**
443

Purpose
This function will compare each unit block sent by the registry against ITL records. If blocks do not match, they will be marked as inconsistent.
Inputs
ReconciliationObject
Table(s)
Unit_Block
Process
To simplify the comparison, this function will first analyze the UnitBlockObject array within the ReconciliationObject and recombine all unit blocks with adjacent serial numbers and identical attributes. The reduced unit blocks will be stored in a new, Combined UnitBlockObject array. The registry, account type, unit type and applicable Commitment Period are listed for each unit block. Likewise, this function will analyze the Reconciliation Snapshot table and combine unit blocks with adjacent serial numbers that have identical attributes, and are held in the same account type at the registry in question. These unit blocks will be stored in the Master UnitBlockObject array. The Master UnitBlockObject will maintain a record of the original block ID of the unit. If multiple blocks were combined to create the record, the ID of each block combined will be stored. Create a new field to serve as a key field. This field should be the account type combined with the unit type and the block start number. Order each array by the key field. Loop through the arrays, keeping them in sync, and compare the corresponding records of each file until the end of both arrays is reached. In normal circumstances, the two reduced unit block arrays will be identical and no inconsistent blocks will be identified. Loop until the ends of both the Combined and Master arrays are reached. Call Compare Unit Blocks check with Combined Key Field and Master Key Field If result = pass, Move to next record in Combined array. Move to next record in Master array. If result = failure, If Combined Key Field < Master Key Field OR End of Master array, The Combined Array has a key that is not in the Master array. Call Get_Block_ID to return any overlapping blocks in the Unit_Block table.

(cont.)

444 **Figure E74: Validate_Unit_Blocks (Function) (cont.)**
445

Process
For each Block_ID returned, Call Insert_Inconsistent_Block. Move to next record in Combined array. Do not move to next record in Master Array. If Combined Key Field > Master Key Field or End of Combined array, The Master array has a key that is not in the Combined array. Call Insert_Inconsistent_Block for each block ID that made up the record in the Master array unit block. Move to next record in the Master array. Do not move to next record in Combined array. If any inconsistent blocks are found, call Write_To_Reconciliation_Status to update the status to 4 ("Unit Blocks Inconsistent"). Call Request_Audit_Trail to have the registry send a transaction history for each inconsistent block since the last successful reconciliation. If no inconsistent blocks are found, the reconciliation enters the manual intervention phase. This phase determines why the unit total counts did not match in the first step of reconciliation.
Outputs
Result_Identifier ResponseObject
Call(s)
Insert_Inconsistent_Block Request_Audit_Trail Write_To_Reconciliation_Status Get_Block_ID

446

447 **Figure E75: Write_Audit_Trail_To_File (Function)**
448

Purpose
This function will create a new text file and transfer the transaction history presented in the incoming message to the newly created file. The transaction history will be stored in XML format.
Inputs
TransactionObject
Table(s)
Process
Create and open new text file. For each transaction in the TransactionObject array, write contents to text file.
Outputs
Result_Identifier File Name and Path
Call(s)

449

Figure E76: Write_Block_History (Function)

Purpose
This function will insert a record in the Transaction_Block_History table.
Inputs
Transaction_Block_ID, Response_Code
Table(s)
Transaction_Block_History
Process
Append to the Transaction_Block_History table the DateTime and response code passed for a Transaction_Block_ID.
Outputs
Result_Identifier
Call(s)

453 **Figure E77: Write_To_File (Function)**
454

Purpose
This function will create a text file, write the contents of the HTTP SOAP Request to the file, then add the file to a master Zip file.
Inputs
Web_Service_URL, Transaction_Type, XML Message Content, File_name
Table(s)
This function does not interact with the database.
Process
Retrieve the to and from elements in the contents of the SOAP request. Retrieve the Registry Code and Transaction Identifier values. Generate the file name by concatenating these two values along with a random number. Write contents of XML body to text file and store in the Zip file. If the file fails to write to file, return HTTP SOAP response error. Record file name and the Zip file name in which the file is compressed to message queue to be recorded in Message_Log.
Outputs
Result_Identifier
Call(s)

455

456 **Figure E78: Write_To_Message_Log (Function)**
457

Purpose
This function will append a record to the Message_Log. The Message_Log table records the history of all ITL message exchange. The contents of an HTTP SOAP Request received from either a registry, CDM or the STL are parsed and constructed as text files that are then stored in a larger Zip file. The Message_Log tracks the time when the file was created, the name of the file and the name of the Zip file in which it has been compressed.
Inputs
Filename, Master_File_Name, Registry_Code, Reconciliation ID, Transaction ID, Web service
Table(s)
Message_Log
Process
Append to Message_Log, Registry_Code, File_Name, Master_File_Name, SystemTime( ) Reconciliation ID or Transaction ID, Web service to Registry_Code, File_Name, File_Path, Submission_Date, Recon_ID, Transaction_ID, Web_Service.
Outputs
Result_Identifier
Call(s)

458

459 **Figure E79: Write_To_Message_Queue (Function)**
460

Purpose
This function adds the HTTP SOAP request information to a message queue.
Inputs
Include all incoming HTTP SOAP request information.
Table(s)
This function does not interact with the database.
Process
This function adds the contents of all incoming requests to the appropriate message queue. In addition, the URL of the Initiating Registry and the date and time of the request is added to the queue. Note that the date and time a message is added to the queue becomes the official date and time of record for the transaction.
Outputs
Result_Identifier
Call(s)

461

462 **Figure E80: Write_To_Reconciliation_Log (Function)**
463

Purpose
This function inserts a new record into the Reconciliation_Log table.
Inputs
ReconciliationObject
Table(s)
Reconciliation_Log
Process
Append to Reconciliation_Log table as follows: • Recon_ID = input reconciliation ID • Recon_Action BeginDateTime = input reconciliation begin DateTime • Registry_Code = input registry • Recon_Log_Comment = null • Recon_Action EndDateTime = input reconciliation end DateTime • Recon_Snapshot_DateTime = input reconciliation snapshot DateTime • Recon_Phase_Code = input reconciliation phase
Outputs
ResponseObject
Call(s)

464

Figure E81: Write_To_Reconciliation_Status (Function)

Purpose
This function inserts a new record into the Reconciliation_Status_History table.
Inputs
ReconciliationObject
Table(s)
Reconciliation_Status_History
Process
Append to Reconciliation_Status_History table as follows: • Recon_ID = input reconciliation ID • Recon_Status_Code = input reconciliation status code • Recon_Log_DateTime = system time • Recon_Comment = null
Outputs
ResponseObject ReconciliationObject
Call(s)

468 **Figure E82: Write_To_Routing_Log (Function)**
469

Purpose
This function will add a record to the Routing Log in order to track interactions with an STL.
Inputs
Route_Status_Code, Registry_Code, Transaction_ID
Table(s)
Routing_Log
Process
Add record to Routing Log indicating the TransactionObject (or any other object) has been sent to an STL or has been received from an STL.
Outputs
Call(s)

470

Figure E83: Write_Transaction (Function)

Purpose
This function will insert a record into the Transaction_Log table.
Inputs
TransactionObject
Table(s)
Transaction_Log
Process
Append to Transaction_Log the elements in the TransactionObject.
Outputs
Result_Identifier
Call(s)
Write_Transaction_Block

474 **Figure E84: Write_Transaction_Block (Function)**
475

Purpose
This function will insert a record into the Transaction_Block table.
Inputs
TransactionObject, RegistryUnitBlockObject
Table(s)
Transaction_Block
Process
For each instance of the RegistryUnitBlock associated with the TransactionObject, insert a record into the Transaction_Block table.
Outputs
Result_Identifier
Call(s)
Write_Transaction_Status

476

Figure E85: Write_Transaction_Status (Function)

Purpose
This function will insert a record into the Transaction_Log_History.
Inputs
TransactionObject
Table(s)
Transaction_Log_History
Process
Append a record to the Transaction_Log_History table with Transaction_ID, system time stamp and current transaction status code from TransactionObject.
Outputs
Transaction_Log_History_ID, unique identifier for new Transaction_Log_History record, Result_Identifier
Call(s)