Technical Specifications

UHOne Facets Batch Framework - Technical Specifications

|  |  |  |  |
| --- | --- | --- | --- |
|  | Prepared By | Reviewed By | Approved By |
| Name | Himanshu S Srivastava | Himanshu S Srivastava |  |
| Signature | Himanshu | Himanshu |  |
| Date | 03/01/2017 |  |  |

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Description | Author | Reviewer |
| 03/01/2017 | 1.0 | Initial Version | Himanshu S Srivastava |  |
| 03/12/2018 | 1.1 | Upated for CLMU configuration | Himanshu S Srivastava |  |
| 03/18/2018 | 1.2 | Upated for 837 Batch configuration | Himanshu S Srivastava |  |

Contents

[1. Overview 3](#_Toc509405343)

[1.1. Purpose 3](#_Toc509405344)

[1.2. Functional Description 3](#_Toc509405345)

[1.3. Assumptions 3](#_Toc509405346)

[1.4. Related Application Requirements 3](#_Toc509405347)

[1.5. Key Considerations 3](#_Toc509405348)

[1.6. Related Documents References 4](#_Toc509405349)

[1.7. Acronyms and Terms 4](#_Toc509405350)

[2. Technical Description 5](#_Toc509405351)

[2.1. Program Specifications 5](#_Toc509405352)

[2.2. Detail Program Specifications 14](#_Toc509405353)

[2.3. Processing Logic 41](#_Toc509405354)

[2.4. Batch Files 41](#_Toc509405355)

[2.5. Restart Processing 41](#_Toc509405356)

[2.6. Error Processing 41](#_Toc509405357)

[2.7. Architectural Mechanism 41](#_Toc509405358)

[2.8. Application Interfaces 41](#_Toc509405359)

[2.9. Security 41](#_Toc509405360)

[3. Appendix 42](#_Toc509405361)

# Overview

## Purpose

This document addresses the overview of Facets Batch Framework and the steps required to use the common bacth framework to execute any Facets batch.

## Functional Description

This main function of this framework is the processing of Facets core batches using common batch wrapper NT\_COMMON\_BATCHWRPR . The common wrapper executes a custom batch wrapper To read the configuration xml,perform any pre/post processes, modify the run book and execute the batch.

This script receives batch name and database name as parameters. Based on the batch name, the corresponding section in the configuration file is read and all the batch specific information are fetched and stored in a global dictionary object. Also, the generic information are fetched and stored in the dictionary object.

The various steps in the pre process xml are read and the required steps are executed. The runbook for the batch is modified with the override parameters and the modified runbook is run to execute the batch. The various steps in the post process xml are read and the required steps are executed.

## Assumptions

1. The configuration file "commonsystemprop.xml" is available in the application path.
2. The VB DLL JD\_NTBATCH\_CMNFUNCS is registered in the system registry.
3. The folders configured in the configuration file "commonsystemprop.xml" are available.
4. The executable Blat.exe used to send mail is available in the path mentioned in the commonsystemprop.xml.
5. The input files for the batches are named according to the naming convention mentioned in the configuration file.

## Related Application Requirements

|  |  |
| --- | --- |
| # | Other Programs / Modules / Interfaces that will help to meet this requirement |
| 1 | JD\_NTBATCH\_CMNFUNCS.dll |
|  |  |

## Key Considerations

*NA*

|  |  |  |
| --- | --- | --- |
| Impact (New, Enhancement) | Object/Module/Program/File Type | Object/Module/Program/File Name |
|  |  |  |
|  |  |  |
|  |  |  |

## Related Documents References

This document use references from the following artifacts. <Any business document that can be pulled out from ADR\ SharePoint\Rally>

| Lockbox Functional document |
| --- |
|  |

## Acronyms and Terms

The following list of acronyms and terms are used within this document:

| Acronym / Term | Description |
| --- | --- |
|  |  |

# Technical Description

## Program Specifications

### Program Hierarchy

The following steps are involved in the execution of NT batches:

* The TWS invokes the BAT file with the parameters – batch name and database name.
* The BAT file invokes the NT batch wrapper passing the batch name and database name to it.
* The NT batch wrapper executes the batch with the customized runbook.
* Email notification, log files generated by executing the batch and success/failure return code is passed from the NT batch wrapper.

The following logical steps have to be performed in the NT batch wrapper process:

* Read the configuration file, system default xml and fetch the generic and specific information related to the job.
* If there is any pre-process for the batch, then read the pre-process xml and execute the steps mentioned in the same.
* Modify the runbook xml related to the batch with the required override parameters mentioned in the configuration file.
* Execute the batch with the modified run book.
* If there is any post-process for the batch, then read the post-process xml and execute the steps mentioned in the same.
* Send email notification and log files for the successful completion of the batch.

### Configuration XML File

The Configuration file is an xml which stores all the generic and specific information required for the execution of batches. It has two sections:

**Generic Section**

The generic section contains the following information:

* Facets Database Connection details
* HIPAA Database Connection details
* Location of Custom Scripts, Runtime Files and Runbook xmls.
* Mail Server Configuration details
* FTP Server details

**Specific Section**

The specific section contains the following information:

* Batch Name
* Mailing List
* Input File Name and Path
* Output File Name and Path
* Log File Name and Path
* Pre/Post processes
* Runbook Override Parameters



### Pre-process and Post-process XML files

The pre and post process involves a series of steps to be carried out before and after the execution of a batch. All the steps present in the pre and post process xmls would not be carried out for all the batches. The steps that need to be bypassed for a batch are setup in the Config.xml file.

1. **5th3rd Receipt Batch(CMCBRCP\_BILL\_RECEIPTS\_5TH3RD)**

**Pre-process**

The pre-process for the CMCBRCP\_BILL\_RECEIPTS\_5TH3RD batch involves the following steps:

* Check if the Request file is present
* Check if the Input file is present and not empty
* Update the RunDate in the dictionary object
* Copy the keyword file from Nas location to application server location

**Post-process**

The post-process for the CMCBRCP\_BILL\_RECEIPTS\_5TH3RD batch involves the following steps:

* Remove the Input file from NAS location
* dentify the log file name for the batch
* Mail the log file



1. **Fiserv Receipt Batch(CMCBRCP\_BILL\_RECEIPTS\_FISERV)**

**Pre-process**

The pre-process for the CMCBRCP\_BILL\_RECEIPTS\_FISERV batch involves the following steps:

* Check if the Request file is present
* Check if the Input file is present and not empty
* Update the RunDate in the dictionary object
* Copy the keyword file from Nas location to application server location

**Post-process**

The post-process for the CMCBRCP\_BILL\_RECEIPTS\_FISERV batch involves the following steps:

* Remove the Input file from NAS location
* Identify the log file name for the batch
* Mail the log file

 

1. **Fis Receipt Batch(CMCBRCP\_BILL\_RECEIPTS\_FIS)**

**Pre-process**

The pre-process for the CMCBRCP\_BILL\_RECEIPTS\_FIS batch involves the following steps:

* Check if the Request file is present
* Check if the Input file is present and not empty
* Update the RunDate in the dictionary object
* Copy the keyword file from Nas location to application server location

**Post-process**

The post-process for the CMCBRCP\_BILL\_RECEIPTS\_FIS batch involves the following steps:

* Remove the Input file from NAS location
* Identify the log file name for the batch
* Mail the log file

 

1. **Credit Card Receipt Batch(CMCBRCP\_BILL\_RECEIPTS\_CREDITCARD)**

**Pre-process**

The pre-process for the CMCBRCP\_BILL\_RECEIPTS\_CREDITCARD batch involves the following steps:

* Check if the Request file is present
* Check if the Input file is present and not empty
* Update the RunDate in the dictionary object
* Copy the keyword file from Nas location to application server location

**Post-process**

The post-process for the CMCBRCP\_BILL\_RECEIPTS\_CREDITCARD batch involves the following steps:

* Remove the Input file from NAS location
* Identify the log file name for the batch
* Mail the log file

 

1. **ACH Receipt Batch(CMCBRCP\_BILL\_RECEIPTS\_ACH)**

**Pre-process**

The pre-process for the CMCBRCP\_BILL\_RECEIPTS\_ACH batch involves the following steps:

* Check if the Request file is present
* Check if the Input file is present and not empty
* Update the RunDate in the dictionary object
* Copy the keyword file from Nas location to application server location

**Post-process**

The post-process for the CMCBRCP\_BILL\_RECEIPTS\_ACH batch involves the following steps:

* Remove the Input file from NAS location
* Identify the log file name for the batch
* Mail the log file

 

1. **Eligible Member Count Batch(CDSMCT0)**

**Pre-process**

The pre-process for the CDSMCT0 Batch involves the following steps:

* Update the StartDate in the dictionary object

**Post-process**

The post-process for the CDSMCT0 Batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

 

1. **Billing Processing Batch(CMCBIL0)**

**Pre-process**

The pre-process for CMCBIL0 Batch involves the following steps:

* Update the BillDateDate in the dictionary object

**Post-process**

The post-process for the CMCBIL0 Batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

****

1. **Bill Decision Support Batch(CDSBIL0)**

**Pre-process**

The pre-process for CDSBIL0 Batch involves the following steps:

* Update the BillDateDate in the dictionary object

**Post-process**

The post-process for the CDSBIL0 Batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

`  

1. **Delinquency Letter Creation Batch(CMCRUN\_DLQ0 )**

**Pre-process**

The pre-process for the CMCRUN\_DLQ0 batch involves the following steps:

* Update the RunThruDate in the dictionary object

**Post-process**

The post-process for the CMCRUN\_DLQ0 batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

 

1. **Financial Billing Analysis Batch(CMCFBA0)**

**Pre-process**

The pre-process CMCFBA0 for the involves the following steps:

* Transfer File from NAS Drive to Server FTP Location

**Post-process**

The post-process for the CMCFBA0 Batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

 

1. **ZipCode Load Batch (CMCXSDU\_ZIPC\_TEST)**

**Pre-process**

The pre-process CMCXSDU\_ZIPC\_TEST for the involves the following steps:\

* Transfer File from NAS Drive to Server FTP Location

**Post-process**

The post-process for the CMCXSDU\_ZIPC\_TEST Batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

 

1. **Member Eligibility Run Batch(CMCRUN\_ELIG)**

**Pre-process**

The pre-process for the CMCRUN\_ELIG Batch involves the following steps:

* Update the StartDate in the dictionary object

**Post-process**

The post-process for the CMCRUN\_ELIG Batch involves the following steps:

* Identify the log file name for the batch
* Mail the log file

 

1. **Claim Adjudication Process(CMCRUN\_CLMU)**

**Pre-process**

The pre-process for the CMCRUN\_CLMU Process involves the following steps:

* Update the StartDate in the dictionary object

**Post-process**

The post-process for the CMCRUN\_CLMU Analysis Process involves the following steps:

* Identify the log file name for the batch
* Mail the log file

 

1. 837I Claims Load Batch (CCS837I)

Pre-process

The pre-process for the CCS837I involves the following steps:

* Copy the keyword file from Nas location to application server location and archive the file.

Post-process

The post-process for the Batch Billing Receipt Processing involves the following steps:

* Delete thet file from Nas location

 

1. **837Out (Create 837 outbound EDI transaction) Batch(CCS837OUT)**

Pre-process

The pre-process for the CCS837OUT Process involves the following steps:

* Update the StartDate in the dictionary object

Post Process

The post-process for the CCS837OUT Process involves the following steps:

* Identify the log file name for the batch
* Mail the log file



1. **834Out (Create 834 EDI Output EDI File) Batch(FIRSTSOURCE834OBatch)**

This process generates the 834 EDI file and stored in below directory

**\\apsed4491.ms.ds.uhc.com\e$\** **TIBCO\Instream\PF\_PostOffice\** **ICICIONESOURCE\OUT**

Note: This is the directory referenced to the Dev server and similar directory structure shall be used for the other environments as well

RunBook Name: ErCcsRun834o\_GWY2.xml

**Post Process**

The post-process for the CCS837OUT Process involves the following steps:

* Identify the log file name for the batch
* Mail the log file



## Detail Program Specifications

Refer “Batch Process Flow” attached in the [APPENDIX](#_JP_JCBC_JPIUDIR_837_Stored_Procedure) for the detailed program flow.

#### NT Batch Wrapper NT\_BATCHWRPR

The VB script NT\_BATCHWRPR.vbs should be developed to read the generic and batch related information from the Config.xml and store them in dictionary objects, read and execute the pre/post processes, generate runbooks and execute them.

This script takes in two input parameters through command line arguments.

##### Input Parameters:

* Batch name
* Database name

The following are the steps involved in this script.

1. Declare the dictionary object g\_dicBatchInfo to store generic, batch specific information present in the Config.xml file and the information available in the System Configuration file.
2. Declare the dictionary object g\_dicOverrideInfo to store the override parameters information present in the Config.xml file.
3. Declare arrays g\_arrPreBypassInfo and g\_arrPostBypassInfo to store the bypass steps/actions for the pre/post processes related with the batches.
4. Declare the global XML DOM object g\_loadXml to load the configuration xml, system default xml and pre/post process xml.
5. Call the [main](#_Function_main) function

##### Function main

1. Create the object g\_dllCmnFunctions to access the JD\_NTBATCH\_CMNFUNCS DLL functions.
2. Create the XML DOM object g\_loadXml.
3. Call the function [f\_ValidateInputParameter](#_Function_f_ValidateInputParameter()) to validate the input parameters namely batch name and database name passed to the batch and store the database name, batch name, password, generic configuration details and batch specific details present under the ‘Category’ node in the Config.xml in the dictionary object g\_dicBatchInfo.
4. If the validate input parameter function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit the batch with a return value.
5. Call the function [f\_ReadGenericInfo](#_Function_f_ReadGenericInfo()) to store the generic Information in the Config.xml into the dictionary object g\_dicBatchInfo.
6. If the [f\_ReadGenericInfo](#_Function_f_ReadGenericInfo()) function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit the batch with a return value.
7. Create the database connection object g\_connDatabase used for database operations.
8. Establish the connection to the database by invoking the EstablishConnection function from the DLL object g\_dllCmnFunctions by passing the Database Server, Port, Provider, Username, Password and Database as input parameters.
9. Make an entry into the Application Log table stating the ‘Execution of Batch started’ using the WriteTrace function in the dll.
10. Call the function [f\_ReadConfigFile](#_Function_f_ReadConfigFile()) to read the configuration file and store the override parameters information in a dictionary object g\_dicOverrideInfo and the bypass steps/actions in the pre/post processes, if any, in two arrays g\_arrPreBypassInfo and g\_arrPostBypassInfo.
11. If the read configuration file function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit with a return code.
12. Call the function [f\_ReadSystemFile()](#_Function_f_ReadSystemFile()) to read the ErSystCfgSystemDefault.xml, fetch all the information and store them in a dictionary object g\_dicBatchInfo.
13. If the read system file function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit with a return code.
14. Check if pre-process exists. If it exists, go to step 12 else go to step 14.
15. Call the function [f\_ExecuteProcessFile](#_Function_f_ExecuteProcessFile) by passing the pre-process bypass array and the pre-process xml file as parameters to read the pre-process information and execute the steps in them.
16. If the execute pre-process function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit with a return code.
17. call the function [f\_BuildRunBook](#_Function_f_BuildRunBook) to modify the batch runbook with the override parameters present in the g\_dicOverrideInfo object.
18. If the build runbook function returns false, call the function f\_UnloadGlobalObjects and exit with a return code. Else call the [f\_ExecuteBatch](#_Function_f_ExecuteBatch()) to run the batch with the modified runbook.
19. If the execute batch function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit with a return code.
20. Check if post-process exists. If it exists, go to step 19 else exit with a return code.
21. Call the function [f\_ExecuteProcessFile](#_Function_f_ExecuteProcessFile) by passing the post-process bypass array and post-process xml file as parameters to read the post-process information and execute the steps in them.
22. If the execute post-process function returns false, call the function [f\_UnloadGlobalObjects](#_Function_f_UnloadGlobalObjects) and exit with a return code.
23. Exit the process with a return code.

##### Function f\_ValidateInputParameter()

This function is used to validate the input parameters passed to the batch. The following are the steps involved in this function:

1. Declare the local variables to store the shell object and argument object.
2. Set the f\_ValidateInputParameter function to ‘True’ by default.
3. Create the shell script object l\_wshShell to perform shell functions.
4. Create the dictionary object g\_dicBatchInfo to store the database name, batch name, password, generic configuration details and batch specific details like mailing list, subject, body, pre and post process, log file name and directory present under the ‘Category’ node in the Config.xml.
5. Add the items “RETURN CODE” with the value of zero and “MESSAGE” with the value of empty space to the dictionary object g\_dicBatchInfo.
6. Create the argument object l\_argValues to access the command line arguments.
7. Check the count of the arguments (input parameters)
8. If the argument count is equal to two, fetch the batch name and database name from the l\_argValues object and store them in the dictionary object g\_dicBatchInfo else log the message “Invalid No. of Parameters” in the system event log and exit with a negative return code.
9. Load the Config.xml to the DOM object g\_loadXml.
10. Set the node list to “BatchConfig/Category" in the variable l\_nodeBatchConfig.
11. Iterate through each category name in the Config.xml to search for the batch name passed as parameter.
12. If the batch name is not found in the Config.xml set the f\_ValidateInputParameter function to ‘False’.
13. If the batch name is not found in the Config.xml, log the message “Invalid Batch Name” in the system event log and exit with a negative return code.
14. Destroy Command line argument object and shell object.
15. Set the function f\_ValidateInputParameter to False, if any run time are encountered.

##### Function f\_ReadGenericInfo ()

This function is used to read the configuration xml, fetch the Generic Information and store them in the dictionary object.

The following steps are involved in this function

1. Set the function f\_ReadGenericInfo to True.
2. Set the node list to "GenericConfig/Category/Item" in the variable l\_nodeGenConfig.
3. For each node present in the node list l\_nodeGenConfig, set the attribute to “name” and add the attribute value and the node text to the batch dictionary object if the item does not exists else update the value of the item. Thus, all the generic information is added to the batch dictionary object.
4. Set the function f\_ReadGenericInfo to False, if any runtime error is encountered

##### Function f\_ReadConfigFile()

This function is used to read the configuration xml, fetch all the batch specific information in them and store them in dictionary objects and arrays.

The following are the steps involved in this function:

1. Declare the local variables to access the various nodes/items/attributes in the xml.
2. Set the f\_ReadConfigFile function to ‘True’ by default.
3. Create the dictionary object g\_dicOverrideInfo to store the override parameters for a batch.
4. Set the node list to "BatchConfig/Category" in the variable l\_nodeBatchConfig.
5. For each node present in the node list l\_nodeBatchConfig, check the value of the attribute “name” to be equal to the batch name.
6. If the category name is equal to the batch name, then get all the child nodes underneath into the g\_dicBatchInfo object.
   1. If the attribute name is not equal to “OverrideParameters”, then call the function [f\_ModifyParamValue](#_Function_f_ModifyParamValue) function to format the node text, then store the attribute name and formatted node text to the g\_dicBatchInfo object if the item does not exists else update the value of the item.
   2. If the attribute name is “Pre-process” or “Post-process”, then call the [f\_BuildBypassArray](#_Function_f_BuildBypassArray()) function passing the node and process type.
   3. If the attribute name is “OverrideParameters”, then fetch all the override parameters under it and their values and add them to the g\_dicOverrideInfo dictionary object.
7. Set the function f\_ReadConfigFile to ‘False’ when any run time errors are encountered.

##### Function f\_BuildBypassArray()

This function is used to build the pre/post process bypass array, which contains all the bypass steps or actions for the pre/post processes.

**Input Parameters:**

* (ByVal) p\_strNode
* (ByVal) p\_strProcessType

The following are the steps involved in this function:

* 1. Declare all local variables.
  2. Set the f\_BuildBypassArray function to ‘True’ by default.
  3. Get the elements by the tag name “Element” from the p\_strNode and set it to the local variable l\_nodeBypassItems.
  4. Fetch the number of items under l\_nodeBypassItems and store it in the local variable l\_intNoOfItems.
  5. If there are bypass items, then redimension the local array l\_arrBypassInfo with the array length to be equal to the number of bypass items minus two and store the bypass steps/actions in the array. Else, redimension the array to zero.
  6. If p\_strProcessType is equal to "Pre-Process", then set g\_arrPreBypassInfo equal to l\_arrBypassInfo else set g\_arrPostBypassInfo equal to l\_arrBypassInfo.
  7. Write the end of the Build Bypass Array function in database by calling the WriteTrace function from the DLL object g\_dllCmnFunctions.
  8. Set the function f\_BuildBypassArray to ‘False’ when any run time errors are encountered.

##### Function f\_ReadSystemFile()

This function is used to read the system xml, fetch all the information in them and store them in a dictionary object g\_dicBatchInfo.

The following are the steps involved in this function:

1. Declare the local variables to access the nodes/attributes in the System default xml.

1. Set the f\_ReadSystemFile function to ‘True’ by default.
2. Load the ErSystCfgSystemDefault.xml to the DOM object g\_loadXml.
3. Set the node list to "Category/Item” in the variable l\_nodeSysConfig.
4. For each node present in the node list l\_nodeSysConfig, set the attribute to “name” and add the attribute value and the node text to the g\_dicBatchInfo object if the item does not exists else update the value of the item. Thus, all the information is added to the dictionary object.
5. Set the function f\_ReadSystemFile to ‘False’ when the function fails with any run time errors.

##### Function f\_ExecuteProcessFile

This function reads the various steps in the pre/post process xml and executes the steps in them one by one if the step is not present in the g\_arrPreBypassInfo/g\_arrPostBypassInfo array.

**Input Parameters:**

* (ByVal) p\_bypassArray
* (ByVal) p\_processFileName

The following are the steps involved in this function:

1. Declare all the local variables to store the step list, step number and flags for checking steps and actions.
2. Declare the array l\_arrParamList to store the parameters of a function.
3. Set the function f\_ExecuteProcessFile to ‘True’ by default.
4. Load the pre/post process xml p\_processFileName to the DOM object g\_loadXml.
5. Set the node l\_nodeProc to root node.
6. Fetch all the steps in the pre/post process xml into a local variable l\_strStepList.
7. For each step present in the l\_nodeStepList, get the step number into a local variable l\_nodeStepNumber.
8. The flag l\_blnChkStep will be set to true if the bypass step is present in the p\_bypassArray (passed as parameter).
9. Traverse through the p\_bypassArray object to check if the step number l\_nodeStepNumber is present in the array by calling the [f\_CheckBypass](#_Function_f_CheckBypass) function.
10. If the step number is present in the array, set the l\_blnChkStep flag to true and fetch the next step in the pre/post process xml.
11. If the step is not present in the array, that is, if the flag l\_blnChkStep is false, fetch all the actions under the step into a local variable l\_nodeAction.
12. For each action present in the l\_nodeAction, get the action number into a local variable l\_nodeActNumber.
13. Set a flag l\_blnChkAction to False. This flag will be set to true if the bypass action is present in the p\_bypassArray.
14. Traverse through the p\_bypassArray object to check if the action number is l\_nodeActNumber is present in the array by calling the [f\_CheckBypass](#_Function_f_CheckBypass) function.
15. If the action number is present in the array, set the l\_blnChkAction flag to true and go to the next action present under the step.
16. If the action is not present in the array, that is, if the flag l\_blnChkAction is false, then check the child nodes under the action.
17. Call the function [f\_BuildExecuteAction](#_Function_f_BuildExecuteAction) by passing the child nodes and the node name. If it returns false, then set the function f\_ExecuteProcessFile to false.
18. Set the flag f\_ExecuteProcessFile to ‘False’ when any run time errors are encountered.

##### Function f\_BuildExecuteAction

This function is used to execute the VB DLL function or VB script or stored procedure present as step/action in the pre/post process xmls.

**Input Parameters:**

* (ByVal) p\_strActInfo
* (ByVal) p\_strType

The following are the steps involved in this function:

1. Declare the local variables.

1. Set the function f\_BuildExecuteAction to ‘True’ by default.
2. Create a shell object l\_wshShell to perform shell functions.
3. Check if p\_strActInfo has any child nodes. If it has child nodes, then get the function/vb script/procedure name to l\_strName and the number of parameters to l\_intNoOfParam else set the number of parameters l\_intNoOfParam to zero.
4. If there is atleast one parameter, then continue with steps 6 else go to step 7.
5. Add the parameters to an array l\_arrParamList and invoke the function [f\_ModifyParamValue](#_Function_f_ModifyParamValue) passing this array to it to update the elements in the array and perform the following when the function returns ‘True’, else if the function returns ‘False’ then assign f\_BuildExecuteAction to ‘False’.
   1. If p\_strType is “FunctionInfo” then add the parameter name and values into the dictionary object l\_dicFuncParam.
   2. If p\_strType is "Script Info" then store the parameter values into the local variable l\_strArrayParam with spaces in between.
   3. If p\_strType is “SqlInfo” then add the parameter name and value into the parameter array l\_arrParamList.
6. Redimension the array l\_arrParamList with dimension (0,2).
7. If p\_strType is “FunctionInfo”, then execute the DLL function [CallDLLFunction](#_CallDLLFunction_Function) by passing the function name l\_strName and parameter array l\_arrParamList as parameters to the function. If the function returns a Boolean value ‘False’, then perform the following.
   1. If function name is ‘CheckInputFile’ and Return Code is 3, then assign f\_BuildExecuteAction to ‘True’ and perform step b, else perform step c.
8. If p\_strType is “ScriptInfo”, then execute the VB script l\_strName in command prompt using shell function passing the all the required parameters for the script contained in the string l\_strArrayParam. If the Return Code for the function is not equal to zero, then aggign f\_BuildExecuteAction as ‘False’ else perform the following
9. Initialized the method in the beginning with True.
10. Remove the finally block to avoid extraneous execution of code.
11. Destroy the shell object l\_wshShell.
12. Write the end of the Build Execute Action function in database by calling the WriteTrace function from the DLL object g\_dllCmnFunctions.
13. Set the flag f\_BuildExecuteAction to ‘False’ when any run time errors are encountered.

##### Function f\_CheckBypass

This function is used to check if the bypass step or action is present in the pre/post bypass array.

**Input Parameters:**

* (ByVal) p\_bypassStep
* (ByVal) p\_bypassArray

The following are the steps involved in this function:

* 1. Declare the local variables.
  2. Set the function f\_CheckBypass function to ‘False’ by default.
  3. Check if the step number/action p\_bypassStep is present in the array p\_bypassArray.
  4. If it is present, set the f\_CheckBypass function to ‘True’ and exit the function.

##### Function f\_BuildRunBook

This function is used to modify the runbook with the override parameters present in the g\_dicOverrideInfo dictionary object, and generic parameters present in the g\_dicBatchInfo dictionary object and save the modified runbook with the name <database>.runbook name in a location configured in the configuration file.

The following are the steps involved in this function:

1. Declare the local variables to store the run book name and run book location.
2. Set the function f\_BuildRunBook function to ‘True’ by default.
3. Fetch the run book name from the g\_dicOverrideInfo object and store it in a local variable l\_strRunBook.
4. Fetch the location of the uncommented run book l\_strRunBook from the g\_dicBatchInfo object and store them in the local variable l\_strRunBookLoc.
5. Load the runbook xml from the l\_strRunBookLoc location to the DOM object g\_loadXml.
6. Set the node l\_nodeRoot to root node in the runbook.
7. Fetch all the items under the root and load them in the variable l\_nodeItems
8. For each item in the l\_strItemNode, fetch the text present in the runbook xml into a local variable l\_nodeOldText.
9. Fetch the attribute “name” of the item into a local variable l\_nodeItemAttrib.
10. Check if the attribute value is present in the dictionary object g\_dicOverrideInfo.
11. If the attribute is present, then create a new text node l\_nodeNewText with the value present in the g\_dicOverrideInfo object and replace the old text l\_nodeOldText with the l\_nodeNewText for the item.
12. Check if the attribute value is present in the dictionary object g\_dicBatchInfo.
13. If the attribute is present, then create a new text node l\_nodeNewText with the value present in the g\_dicBatchInfo object and replace the old text l\_nodeOldText with the l\_strNewText for the item
14. If the attrib value is not present, then go to the next item in l\_nodeItems.
15. Fetch l\_strRunbook as by concatenating the trailing characters after ‘facets’ in the database name (p1 if database name is facetsp1), ”.” and the Runbook name
16. Save the runbook xml with the name l\_strRunBook in the Workfile path.
17. Set the f\_BuildRunBook function to ‘False’ when any run time errors are encountered.

##### Function f\_ExecuteBatch()

This function f\_ExecuteBatch() is used to execute the batch with the modified run book xml.

The following are the steps involved in this function:

1. Declare the local variables to store the run book name and run book location.
2. Set the function f\_ExecuteBatch to ‘True’ by default.
3. Create the shell script object l\_wshShell to execute the batch.
4. Fetch the customized run book name from the g\_dicOverrideInfo object and store it in a local variable l\_strRunBook.
5. Fetch the location of the uncommented run book l\_strRunBook from the g\_dicBatchInfo object and store them in the local variable l\_strRunBookLoc.
6. Run the batch from the shell object l\_wshShell using the following command.

CScript ErSys0FrmExecuteJob.wsf l\_strRunBook,1,True.

1. Destroy the l\_wshShell object to nothing.
2. Set the f\_ExecuteBatch function to ‘False’ when any run time errors are encountered.

##### Function f\_ModifyParamValue

**Input Parameters:**

* (ByRef) p\_arrParamList

1. Declare the local variables to store the parameter value of a function, position of first occurrence of "+" in the parameter, and array to store different strings separated by "+" present in a parameter.
2. Write the start of the Modify Parameter Value function in database by calling the WriteTrace function from the DLL object g\_dllCmnFunctions.
3. Set the function f\_ModifyParamValue to ‘True’ by default.
4. Get the number of parameters in the array into the local variable l\_intNoOfParam.
5. Loop through each parameter in the array.
6. Store the parameter in a local variable l\_strFuncParam.
7. Check if the name of the parameter is either “BatchInfo” or “OverrideInfo”.

If the name is “BatchInfo” then set the g\_dicBatchInfo object as the value.

If the name is “OverrideInfo” then set the g\_dicOverrideInfo object as the value.

1. If the name is not either of the above then perform the following steps.
2. Check whether the string “+” is present in the value of the parameter. If no then move to the next element in the array.
3. If yes then split the string into multiple values and then store them in an array l\_arrParameter.
4. Loop through each item in the array l\_strParamValue

Check if it contains the string “Config”.

Extract the value present within the Config() string into a local variable l\_Item.

Check if the l\_Item is present in the dictionary objects g\_dicBatchInfo or g\_dicOverrideInfo.

If it is not present then exit the function and return a value of FALSE.

If it is present in either of the two then fetch the value of the item from the dictionary object and reassign the value of the item of the array.

Check if it contains the string "mmddyy.hhmmss"

Then update the item with current date and time in ‘mmddyy.hhmmss’ format.

Check if it contains the string “mm.dd.yy.Thh”

Then replace ‘mm’ with month, ‘dd’ with date, ‘yy’ with year as of current date and ‘hh’ with hour as of current time in the item.

If it does not have either of the Strings then move to the next element.

1. Loop through the array and build a string l\_strModParamValue with all the elements in the array.
2. Update the value of the parameter with the string l\_strModParamValue.

##### Function f\_UnloadGlobalObjects

**Input Parameters:**

* + - None

1. Unload the following global objects if exists
   * + 1. g\_connDatabase
       2. g\_dllCmnFunctions
       3. g\_loadXml
       4. g\_arrPostBypassInfo
       5. g\_arrPreBypassInfo
       6. g\_dicOverrideInfo
       7. g\_dicBatchInfo

##### VB DLL JD\_NTBATCH\_CMNFUNCS

A VB DLL JD\_NTBATCH\_CMNFUNCS should be developed which contains the common functions, which will be used by the wrapper process. These functions will be either used directly in the wrapper script or be configured in the pre/post-process xml files.

##### EstablishConnection Function

**Input Parameters:**

* p\_strProvider
* p\_strUserName
* p\_strDBName
* p\_strServerName
* p\_strServerPort

1. Declare a connection variable.
2. Set the function EstablishConnection as ‘True’
3. Extract the decrypted password by calling function - CerGetPassword with required parameters.
4. Set the password in class session.
5. Build a oracle connection string with the password extracted.
6. Create a oracle connection object with the string built.
7. Open connection and close the same at finally block.
8. Write exception and return false when error raised.

##### FileDelete Function

**Input Parameters:**

* p\_strSourceFile
* p\_blnMultipleFiles
* p\_strSourceDir
* p\_intDaysOld

1. Set the return value for the function, FileDelete to ‘True’
2. Declare a IO.File objects to perform File operation.
3. Fetch the name of the source file, which is to be deleted and store the file name along with the path in local variable.
4. If p\_blnMultipleFile is ‘N’ then continue with step 4 else with step 7
5. If the l\_strSource file exists, then check for the date difference between the current date and last modified date.
6. If the date difference is greater than p\_intDaysold then delete the file using the DeleteFile function of File objects.
7. Using the file object fetch each file in the p\_strSourceDir having name like ‘p\_strSouvceFile’ and perform the following
   1. If the date difference between the current date and the Last modified date is greater than p\_intDaysold then delete the file.
8. Destroy the FileStream object and exit the function.

##### CheckInputFile Function

**Input Parameters:**

* p\_strSourceFile
* p\_strSourceDir
* p\_dicBatchInfo (By Ref)

1. Set ‘CheckInputFile’ to ‘True’ by default.
2. Declare a IO.File object to perform file operation.
3. Check if the source file exists using ‘Fileexists’ function.
4. If the file is present check the size of the file.
5. If the file is not present, assign the function with a return value of ‘False’.
6. If the file size is greater than zero then exit the function with a return value of ‘True’. If not assign a return value ‘False’ to the function.
7. If the CheckInputFile is ‘False’ and if the batch name is either ‘JDHC\_CCSMMS0\_ECONN\_EMPLOYER’ or ‘JDHC\_CCSMMS0\_ECONN\_MEMBER’ then assign a Return Code as 0 in the BatchInfo dictionary object, else assign a Return Code as ‘3’.
8. Destroy the FileStream object and exit the function.

##### Function SendMail

**Input Parameters:**

* p\_dicBatchInfo (By Ref)
* p\_strFile
* p\_strReceiver
* p\_strSubject

1. Declare the local variables l\_strMailServerName, l\_strMailPortNumber, l\_strMailSender, l\_strBlat, l\_strBlatPath and l\_strLogFile to hold the name of the mail server, port number of the server, variable to call blat.exe along with parameters, path of the Blat executable and log file name from the p\_dicBatchInfo.
2. If multiple log file delimited by ‘,’ exists in the l\_strLogFile then split the files into the array.
3. Create the parameters for Blat.exe and store it a local variable l\_strBlat with the mail server name, sender, receiver, Subject, Body and attachments if any.
4. Create a shell object to run the Blat command.
5. Run the Blat command from the shell object and check the return code.
6. If the return code is not equal to zero then set the f\_SendMail function to ‘False’ and make an entry in the Application Log by calling the WriteTrace function, else set it to ‘True’.
7. Destroy the shell object.

##### Function UpdateDicOverride

**Input Parameters:**

* p\_strItem
* p\_dicOverrideInfo (ByRef)
* p\_dicBatchInfo (ByRef)

This function is used to update the Override dictionary object.

The following are the steps involved in this function:

1. If the value passed for the parameter p\_strItem is “RunDate”/”BillDate”/”PayDate” then
   * 1. Assign l\_strProcessName for @pAPP\_MODULE\_ID in the parameter array.
     2. Create a connection object with Facets using the dictionary object p\_dicBatchinfo.
     3. Using the function ExecuteSP, execute the procedure FSG\_BTCH\_FETCH\_RUNDATE to fetch the run date from the application control table into the local variable l\_strRunDate.
     4. Update ‘mm/dd/yyyy’ in the l\_strItem with the Rundate fetched in the l\_strRundate

##### GetLogFileName Function

**Input Parameters:**

* p\_strBatchName
* p\_dicBatchInfo (By Ref)

1. Declare the local variables to store the Database Connection object, batch name, system instance id and the log file name.
2. Use the function ExecuteSP to execute the procedure FSG\_BTCH\_FETCH\_INST passing the p\_strBatchName as parameter to fetch the latest system instance id of the batch and store it in a local variable l\_strSyinId.
3. If the Records returned is not zero or returned result is Null then Log a message in the Application control table stating ‘Invalid Record set returned from FSG\_BTCH\_SELECT\_INST’ with Return Code as ‘1’.
4. Fetch l\_strlogFileExt as “.xml” or ‘.txt’ based on the value of the item OutputType in the BatchInfo dictionary object.
5. Trizetto core batch output file names will be manipulated with the name of the batch and Batch id as shown below.

Dim strLogFileName As String = l\_strBatchName & "\_" & l\_strSyinId & "\_LOG" & l\_strLogFileExt

Dim strErrFileName As String = l\_strBatchName & "\_" & l\_strSyinId & "\_ERR" & l\_strLogFileExt

Dim strOutFileName As String = l\_strBatchName & "\_" & l\_strSyinId & "\_OUT" & l\_strLogFileExt

1. Job file name will be manipulated for creating a single JOB file name.

Dim strJobFileName As String = l\_strBatchName & "\_" & l\_strSyinId & "\_JOB" & l\_strLogFileExt

1. Merge the LOG, ERR & OUT file in sequence with the LOG file, with code provided below using MergeFiles functions available in the common dll.

If IO.File.Exists(strCustOutDir & "\" & strJobFileName) Then

IO.File.Delete(strCustOutDir & "\" & strJobFileName)

End If

Dim strFile As String

For Each strFile In (strLogFileName & "|" & strErrFileName & "|" & strOutFileName).Split("|")

MergeFiles(strFile, strCustOutDir, strJobFileName, strCustOutDir, False)

Next

1. End function.

##### FileRename Function

**Input Parameters:**

* p\_strSourceFile
* p\_strSourceDir
* p\_strDestinationFile
* p\_dicBatchInfo (By Ref)

1. Declare the local variables l\_strSourceFile, l\_strSourceDir and l\_strDestFilename to store the source file, source directory and destination file name.
2. If l\_strSourceDir is not empty, form the destination file with path l\_strDestFilePath by combining l\_strSourceDir + l\_strDestFileName.
3. If l\_strSourceDir is empty, then fetch the source directory from l\_strSourceFile and use it to form the destination file with path l\_strDestFilePath.
4. Create IO.File object l\_fsoRenameFile.
5. If the l\_strSourceFile does not exists then assign FileRename to ‘False’ and log a error message into the Application Log table with Return Code ‘10’ by calling the ‘WriteTrace’ function, else perform Step 6.
6. Set the Name of the l\_strSourceFile to l\_strDestFile using the File object.
7. Set the l\_fsoRename object to ‘Nothing’ and exit the function.

##### FileCopy Function

**Input Parameters:**

* p\_strSourceFile
* p\_strSourceDir
* p\_strDestDir
* p\_dicBatchInfo (By Ref)

1. Declare the local variables l\_strSourceFile, lstrSourceDir and l\_strDestDir to store the source file, source directory and destination directory name.
2. Combine the l\_strSourceFile and l\_strSourceDir to form the source file with complete path l\_strSourceFilePath.
3. Create a IO.File Object l\_fsoFileCopy.
4. If the Source file and the Destination Dir exists, Call the CopyFile method of the object l\_fsoFileCopy passing the l\_strSourceFilePath and l\_strDestdir. The file will be copied to the destination folder, else assign FileCopy as ‘False’ and log an error message in the Log table with Return code ‘10’.
5. Assign l\_fsoFileCopy to ‘Nothing’ and exit the function.

##### FTPFile Function

**Input Parameters:**

* p\_dicBatchInfo (ByRef)
* p\_strOperation
* p\_strFTPServer
* p\_strSourceFile
* p\_strDestFile (Optional)

1. Declare the local variables to store the source file and destination.
2. If p\_strDestination = “NASDRIVE”, then fetch the NAS Details (NAS\_Path) from the p\_dicBatchInfo object and store them in the local variables l\_strFTPServerName.
3. If p\_strDestination = “WDRIVE”, then fetch the server name (WD\_ServerName), user name (WD\_UserId) and (WD\_Password) password from the p\_dicBatchInfo object and store them in the local variables l\_strFTPServerName, l\_strFTPUName and l\_strFTPPwd.
4. Check whether the FTP Connection can be established.
5. Establish the FTP Connection.
6. If FTP operation is ‘GET’ then, Call the FtpPutFile function passing the FTP connection, source and destination file names. The file will be FTPed to the required location.
7. Else check whether the Source file exists in the FTP server, if exists continue with Step 8, else assign FTPFile as ‘False’ and Log an error message in the Application Log table with the WriteTrace function.
8. If the FTP operation is ‘GET’ then call the function FtpGetFile to fetch the p\_strSourceFile from the FTP server and assign the return code to l\_blnReturnValue.
9. If the FTP operation is ‘DELETE’ then call the function FtpDeleteFile to delete the file from the FTP server and assign the return code to l\_blnReturnValue.
10. If l\_blnReturnValue returns False then log an error message in the Application log table with return code ‘12’ using the function ‘WriteTrace’.

* Initialized the method in the beginning with True.
* Update the CheckforFile loop to include the FTPFile = True.

##### FileMove Function

**Input Parameters:**

* p\_strSourceFile
* p\_strSourceDir
* p\_strDestDir
* p\_dicBatchInfo (By Ref)

1. Declare the local variables l\_strSourceFile, l\_strSourceDir and l\_strDestDir to store the source file, source directory and destination directory.
2. Create a FileSystemObject l\_fsoFileMove.
3. Check whether the SourceFile and the Destination directory exists, if exists continue with Step 4 else with Step 6.
4. Check whether the any file with the SourceFile name exists in the Destination directory, if exists then delete the File.
5. Call the MoveFile method of the object l\_fsoFileMove passing the l\_strSourceFile along with l\_strSourceDir and l\_strDestDir. The file will be moved and placed in the required destination.
6. Assign FileMove as ‘False’ and log an error message in the Log table with Return Code as ’10’ using the WriteTrace’ function.
7. Set l\_fsoFileMove to ‘Nothing’ and exit the function.

##### MergeFiles Function

**Input Parameters:**

* p\_strSourceFile
* p\_strSourceDir
* p\_strDestinationFile
* p\_strDestinationDir
  1. Declare the local variables l\_strSourceFile, l\_strSourceDir, l\_strDestFile and l\_strDestDir to store the source file, source directory, destination file and directory.
  2. Create a FileSystemObject.
  3. Create a file with the name l\_strDestFile in the l\_strDestDir using the FileSystemObject
  4. Fetch the Source file with name like l\_strSourceFile one by one from the Source Directory.
  5. Read the content of each of the source file and append its contents to the l\_strDestFile using the FileSystemObject.
  6. Save the file.

##### BcpInData Function

**Input Parameters:**

* p\_dicBatchInfo (By Ref)
* p\_strInputFile
* p\_strInputTable
* p\_blnCleanupTable
* p\_strInputSystem

The following are the steps involved in this function.

1. If the p\_strInputSystem is “FACETS” then create a connection object for FACETS database by using the information from the p\_dicBatchInfo else if it is “HIPAA” then create a connection object for HIPAA database.
2. If the flag p\_blnCleanupTable is “Y” then truncate the input table using the connection object.
3. Incase of any error exit the function with a non-zero return code.
4. Bcp the input data file into the input table.
5. Incase of any error exit the function with a non-zero return code.

##### BcpOutData Function

**Input Parameters:**

* p\_dicBatchInfo
* p\_strOutputFile
* p\_strOutputTable
* p\_strOutputSystem

The following are the steps involved in this function.

1. If the p\_strOutputSystem is “FACETS” then create a connection object for FACETS database by using the information from the p\_dicBatchInfo else if it is “HIPAA” then create a connection object for HIPAA database.
2. Bcp the data from the output table into the output data file.
3. Incase of any error exit the function with a non-zero return code.

##### ExecuteSP Function

**Input Parameters:**

* p\_strStoredProcName
* p\_arrParameters (Optional)

The following are the steps involved in this function:

* 1. Create a command object l\_cmdExecuteSp to execute the stored procedure.
  2. Assign the connection object p\_connDatabase to the command object.
  3. If there are any parameters for the procedure, create the parameter array.
  4. Execute the stored procedure p\_strStoredProcName with the constructed parameters.

##### CallDLLFunction Function

**Input Parameters:**

* p\_strName
* p\_dicFuncParam (By Ref)
* p\_dicBatchInfo (By Ref)
* p\_dicOverrideInfo (By Ref)

This function acts as the wrapper function to call other DLL functions based on the parameter p\_funcName.

The following are the steps involved in this function:

* 1. Check for the value of the function name parameter p\_strName.
  2. Based on the name of the function, call the appropriate function present in the DLL JD\_NTBATCH\_CMNFUNCS passing the parameter p\_arrParamList.
* Initialized the method in the beginning with True.
* Remove the finally block to avoid extraneous execution of code.

##### CheckSlash Function

**Input Parameters:**

* p\_strFoldName (By Ref)

This function is used in the dll to add a trailing “\” to the folder name

The steps involved in the function are

1. If the last character in string containing the folder path does not contain the character, “\” and length of the folder path is not zero then add a trailing “\” to the folder name.
2. Return the folder name.

##### CreateEmptyFile Function

**Input Parameters:**

* p\_strSourceDir (By Val)
* p\_strFileName (By Val)

This function is used in the dll to create an empty file using Filesystemobject

* 1. Add a trailing “\” to the input directory if not found using the function [CheckSlash](#_CheckSlash_Function_4)
  2. Create an empty file using the Filesystemobject with Input file name (p\_strFileName) in the Source Dir (p\_strFileName)

##### RemoveTrailingSpaces Function

**Input Parameters:**

* p\_strSourceDir (By Val)
* p\_strFilename (By Val)

This function is used in the dll to remove the trailing spaces in end of the input file.

1. Add a trailing “\” to the input directory if not found using the function [CheckSlash](#_CheckSlash_Function_4)
2. Using a text stream object read content of the file (p\_strFilename) in the input directory (p\_strSourceDir) into a string l\_strReadStr.
3. Using a text steam object create a file (p\_strFilename) in the input directory (p\_strSourceDir).
4. Write the content of the string into the file (p\_strFilename)
5. Incase of any error exit the function with a non-zero return code.

##### CheckSlash Function

**Input Parameters:**

* p\_strFoldName (By Val)

This function is used in the dll to add a trailing “\” to the folder name.

The steps involved in the function are:

1. If the last character in string containing the folder path does not contain the character, “\” and length of the folder path is not zero then add a trailing “\” to the folder name.
2. Return the folder name.

* **VB Scripts**

##### FA\_CopyDelArch\_SourceToDest.vbs

This VB script is used to copy, archive and delete files from source to destination location and vice versa.

The following are the steps involved in this script:

**Input Parameters**

* l\_strSrcFileDir
* l\_strDestFileDir
* l\_strFileName
* l\_function
  1. Set l\_ strSrcFileDir as the source file directory.
  2. Set l\_strDestFileDir as the destination file directory
  3. Set l\_strFileName as a pattern of the file to be read.
  4. Set l\_function as COPY, DELETE or ARCHIVE.
* COPY: Copies all the file from source to destination location based on the file pattern
* DELETE: Deletes all the files from the location based on the file pattern
* ARCHIVE: Copies all the file from source to destination location based on the file pattern and add timestamp to it.

## Processing Logic

Add configuration in commonsystemprop.xml and provide batch name under tag <Category name>, add all generic configurations such as Input/Output dir.In OverrideParameters add all batch specific configurations :

Add the input files as per the configured InputDir and make sure that the input files are named according to the naming convention mentioned in the configuration file.

Call NT\_COMMON\_BATCHWRPR script using batch name and database name as parameters. Based on the batch name, the corresponding section in the configuration file is read and all the batch specific information are fetched and stored in a global dictionary object. Also, the generic information are fetched and stored in the dictionary object.

Once batch completed go into C:\Batch\_Framework\Log to check logs.

## Batch Files

## Restart Processing

* + 1. Program Restart

*<****TBD****: Describe commit/restart-processing logic. Include commit checkpoints and any synchronization that is required to restart the program from the last commit >*

* + 1. Batch Restart

There is no specific processing or logic to be applied to restart the job. The job can be triggered again from start if it fails at any of the steps. The job will start processing the remaining data.

## Error Processing

*<Describe error processing logic here including pseudo-code, input/output table details (Table Name, Field, Field Size, and Field Format), new screens, reports, error codes and error descriptions, etc.>*

## Architectural Mechanism

*<Describe interdependencies and interactions with other programs, modules, events, and interfaces.>*

## Application Interfaces

*<If this Technical Specification is for a new application interface or changes to an existing application interface, complete the Application Interface Specification Template defined in UDP.>*

## Security

<NA>

# Appendix

1. Batch Process Flow Diagram



1. Error Return Codes

|  |  |
| --- | --- |
| VBScript Return code | |
| Return code | **Description** |
| 0 | Successful execution of the script |
| 1 | SQL Execution (Stored Procedure) failed |
| 2 | Empty Record set returned |
| 3 | File not found error |
| 4 | Empty File |
| 5 | File deletion failed in VBScript |
| 6 | File copy failed in VBScript |
| 7 | File move failed in VBScript |
| 8 | File append failed in VBScript |
| 9 | No Input rows to process |
| 10 | No Input File to process |
| 11 | Map execution failed |
| 12 | FTP operation failed |
| 13 | Mail operation failed |
| 14 | Gateway Batch execution failed |
| 99 | Run Time Error |
| -1 | Invalid parameters passed to VBScript |
| -2 | Error while Establishing Database connection |
| -3 | Error while closing the Database connection |

1. Installation Steps

DLL Registraton : regSvr32 E:\Batch\_Framework\VBDLL\JD\_NTBATCH\_CMNFUNCS.dll

Blat Setup : E:\Batch\_Framework\Mail\blat.exe -install mailo2.uhc.com uho\_fac\_bat\_tst