OPTUM™

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	

OPTUM™

Contents

1. Overview

1.1. 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.

1.2. 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.

1.3. Assumptions

I. The configuration file "commonsystemprop.xml" is available in the application path.

II. The VB DLL JD_NTBATCH_CMNFUNCS is registered in the system registry.

III. The folders configured in the configuration file "commonsystemprop.xml" are available.

IV. The executable Blat.exe used to send mail is available in the path mentioned in the commonsystemprop.xml.

V. The input files for the batches are named according to the naming convention mentioned in the configuration file.

1.4. Related Application Requirements

#	Other Programs / Modules / Interfaces that will help to meet this requirement
1	JD_NTBATCH_CMNFUNCS.dll

1.5. Key Considerations

NA

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

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

1.6. 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

1.7. Acronyms and Terms

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

Acronym / Term	Description

2. Technical Description

2.1. 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

commonsystemprop.xml

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

Brcp_PostProc_5th3rd.xml

Brcp_PreProc_5th3rd.xml

2. 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


Brcp_PreProc_Fiserv.xml


Brcp_PostProc_Fiserv.xml

3. 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


Brcp_PreProc_Fis.xml


Brcp_PostProc_Fis.xml

4. 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


Brcp_PreProc_CC.xml


Brcp_PostProc_CC.xml

5. 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


Brcp_PreProc_Ach.xml


Brcp_PostProc_Ach.xml

6. 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


PreProc_MCT0.xml


PostProc_MCT0.xml

7. 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


PostProc_CmcBil0.xml


PreProc_CmcBil0.xml

8. 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


PreProc_CdsBil0.xml


PostProc_CdsBil0.xml

9. 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


PreProc_DLQ0.xml


PostProc_Dlq0.xml

10. 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


PreProc_FBA0.xml


PostProc_FBA0.xml

11. 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


PreProc_XSDU_Test.xml


PostProc_XSDU_Test.xml

12. 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


PreProc_Elig.xml


PostProc_Elig.xml

13. 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


PreProc_Clmu.xml


PostProc_Clmu.xml

14. 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


PreProc837i.xml


PostProc837i.xml

15. 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


PostProc_837O.xml

16. 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


PostProc_834O.xml

2.2. Detail Program Specifications

Refer "Batch Process Flow" attached in the APPENDIX 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

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 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 and exit the batch with a return value.

5. Call the function f_ReadGenericInfo to store the generic Information in the Config.xml into the dictionary object g_dicBatchInfo.

6. If the f_ReadGenericInfo function returns false, call the 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 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 and exit with a return code.

12. Call the 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 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 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 and exit with a return code.

17. call the 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 to run the batch with the modified runbook.

19. If the execute batch function returns false, call the 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 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 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.
    a. If the attribute name is not equal to "OverrideParameters", then call the 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.
    b. If the attribute name is "Pre-process" or "Post-process", then call the f_BuildBypassArray function passing the node and process type.
    c. 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.
2. Set the f_ReadSystemFile function to 'True' by default.
3. Load the ErSystCfgSystemDefault.xml to the DOM object g_loadXml.
4. Set the node list to "Category/Item" in the variable l_nodeSysConfig.
5. 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.
6. 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.
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.
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 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.
2. Set the function f_BuildExecuteAction to 'True' by default.
3. Create a shell object l_wshShell to perform shell functions.
4. 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.
5. If there is atleast one parameter, then continue with steps 6 else go to step 7.
6. Add the parameters to an array l_arrParamList and invoke the 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'.

   a. If p_strType is "FunctionInfo" then add the parameter name and values into the dictionary object l_dicFuncParam.
   b. If p_strType is "Script Info" then store the parameter values into the local variable l_strArrayParam with spaces in between.
   c. If p_strType is "SqlInfo" then add the parameter name and value into the parameter array l_arrParamList.
7. Redimension the array l_arrParamList with dimension (0,2).
8. If p_strType is "FunctionInfo", then execute the DLL function CallDLLFunction 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.
   a. If function name is 'CheckInputFile' and Return Code is 3, then assign f_BuildExecuteAction to 'True' and perform step b, else perform step c.
9. 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
   a. Initialized the method in the beginning with True.
   b. Remove the finally block to avoid extraneous execution of code.
10. Destroy the shell object l_wshShell.
11. Write the end of the Build Execute Action function in database by calling the WriteTrace function from the DLL object g_dllCmnFunctions.
12. 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:

   a. Declare the local variables.
   b. Set the function f_CheckBypass function to 'False' by default.

c. Check if the step number/action p_bypassStep is present in the array p_bypassArray.

d. 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.
7. Destroy the l_wshShell object to nothing.
8. 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.
   1) Store the parameter in a local variable l_strFuncParam.
   2) Check if the name of the parameter is either "BatchInfo" or "OverrideInfo".

      (i) If the name is "BatchInfo" then set the g_dicBatchInfo object as the value.

      (ii) If the name is "OverrideInfo" then set the g_dicOverrideInfo object as the value.

   3) If the name is not either of the above then perform the following steps.
      1) Check whether the string "+" is present in the value of the parameter. If no then move to the next element in the array.
      2) If yes then split the string into multiple values and then store them in an array l_arrParameter.
      3) Loop through each item in the array l_strParamValue
         1. Check if it contains the string "Config".
            a. Extract the value present within the Config() string into a local variable l_Item.
            b. Check if the l_Item is present in the dictionary objects g_dicBatchInfo or g_dicOverrideInfo.
            c. If it is not present then exit the function and return a value of FALSE.
            d. 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.
         2. Check if it contains the string "mmddyy.hhmmss"
            a. Then update the item with current date and time in 'mmddyy.hhmmss' format.
         3. Check if it contains the string "mm.dd.yy.Thh"
            a. 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.
         4. If it does not have either of the Strings then move to the next element.
      4) Loop through the array and build a string l_strModParamValue with all the elements in the array.
      5) 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
   a. g_connDatabase
   b. g_dllCmnFunctions
   c. g_loadXml
   d. g_arrPostBypassInfo
   e. g_arrPreBypassInfo
   f. g_dicOverrideInfo
   g. 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
    a. 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
    a. Assign l_strProcessName for @pAPP_MODULE_ID in the parameter array.
    b. Create a connection object with Facets using the dictionary object p_dicBatchInfo.
    c. 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.
    d. 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
6. Job file name will be manipulated for creating a single JOB file name.
   Dim strJobFileName As String = l_strBatchName & "_" & l_strSyinId & "_JOB" & l_strLogFileExt

7. 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
```

8. 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
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
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

- I_strSrcFileDir
- I_strDestFileDir
- I_strFileName
- I_function

1. Set I_ strSrcFileDir as the source file directory.
2. Set I_strDestFileDir as the destination file directory
3. Set I_strFileName as a pattern of the file to be read.
4. Set I_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.

2.3. 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.

2.4. Batch Files

2.5. Restart Processing

2.5.1. 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 >

2.5.2. 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.

2.6. 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.>

2.7. Architectural Mechanism

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

2.8. 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.>

2.9. Security

<NA>

3. Appendix

1. Batch Process Flow Diagram

Batch Process Flow

2. 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

3. 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

Template Express (Version 1.1)

PROJECT BACKGROUND

Idea: Automation of the process to create rollback , insert and control scripts for UHOne 2.0 deployments.

Problem Statement :

This process is related to creating rollback , insert and control scripts for UHOne 2.0 deployments , creating these templates manually was a time taking process and lead to risk of human error. It is impacting all the Jobs as rollback and control scripts are key components for any deployments.

Manual Process:

- Manual templates Creation Process – Susceptible to errors and time consuming.
- Requires lots of manual effort to maintain the exact Format/Standard.
- Awareness of templates standards to all developers
- Need to review the whole script again if any Syntax Error/Format mismatch left during the process.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava
Project/Team Name: UHOne Transformation 2.0
Project Nature: Development
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: Prashant Srivastava
Onsite Approver: Chitra Gupta

SOLUTION TO IMPLEMENT

- Template Express (Version 1.1) is being used to automate the process to create rollback , insert and control scripts for UHOne 2.0 deployments , creating these templates manually was a time taking process and lead to risk of human error.
- Only package list/comp names requires to perform the activities. User can create, verify and share the results in a single click.
- Very minimal manual interference/review required to create the scripts.
- This tool not requires any database credentials/access , So can also be used while user is offline.

BENEFITS

- User-friendly GUI
- User can easily Generate, Edit and Save the components.
- Option to E-Mail the Insert Script and Report.
- User can work even in Offline Mode.
- Can be used for SQL server, Oracle or Sybase as well.
- This tool provides good amount of time savings in terms of massive decrement in manual creation of schema and their validations.

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Components Per Year	200	200		Approx. DB components Per Year
Total Components(Rollback + Control Scripts)	400	400		
Effort Per Component/Year	1400	160 Hrs	Hrs.	3.5Hrs(Rollback + Control)
Billing Rate: $77	$107800	$12320	Dollars	$77 per Hrs.
Total Cost Savings: 1240 Hrs			Dollars	**$95,480.00**

Other Details

UTILIZATION DETAILS

The efforts saved from this tool can be easily utilized to work on other optimizations or in the project work

LEVERAGING OPPORTUNITIES

- ✓ This tool can also be used for other clients, with requirement of template generation in different database environments (Sybase , Oracle or SQL Server). This tool will surely reduce the manual efforts and will save time.

SIGN-OFF MAIL

RE Innovation - Utilities for UHOne .msg

Outlook Item

Common Framework Scripts

PROJECT BACKGROUND

Idea: Common Framework Scripts to automate the deployment process for Talend Jobs common configuration in Database.

Problem Statement :

Currently we need to create different framework scripts for different environments which have details like nas path, job name and parameters, and then need to raise the deployment tickets for each individual environments to insert the details in DB.

The SLA for each tickets is around 1-3 business Days. and it requires manual check-in and validation of scripts as we move to higher environments.

It is impacting all the Jobs as framework scripts are key components for any deployments.

Manual Process:

- Need to create environment specific scripts.
- Manual Tickets for deployment– SLA 1-3 Days.
- Need to track the status of each ticket.
- Automated deployment can not be achieved for these scripts.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava
Project/Team Name: UHOne Transformation 2.0
Project Nature: Development
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: Prashant Srivastava
Onsite Approver: Chitra Gupta

SOLUTION TO IMPLEMENT

- Common framework scripts are dynamically written scripts that's have job specific details for all environments and would be executed based on Database name automatically.
- Once we move to higher environments the specific details only be inserted based on DB check inside the scripts using automated deployments.
- Reduced the time effort and Incidents tickets as deployment was automated using Jenkins.

BENEFITS

- No need to raise new INC tickets for each environment separately.
- Automated Deployment Via Jenkins
- Common Template which can be utilized for other Talend Jobs as well.

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Average Ticket / Month	25	0	INC Tickets	After improvement No tickets will be needed as deployment would be done through Jenkins.
Average Ticket (Yearly)	300	0		
Average Effort / Annum	1950	0	Hrs.	**Before implementation** Script Creation and Validation for Dev : 3hr/ticket App Ops : 3.5hr/ticket **After implementation** Achieved through automated deployments via Jenkins.
Billing Rate: $77	$150150		Dollars	$77 per Hrs.
Total Cost Savings: 1950 Hrs.			Dollars	**$1,50,150.00**

Other Details

UTILIZATION DETAILS

The efforts saved from this tool can be easily utilized to work on other optimizations or in the project work

LEVERAGING OPPORTUNITIES

- ✓ This process can also be used for other Teams with requirement of Framework configuration in different database environments. This tool will surely reduce the manual efforts and will save time.

SIGN-OFF MAIL

RE Innovation - Utilities for UHOne .msg

Outlook Item

Automated MMC Cleanup Script for Old Services

PROJECT BACKGROUND

Idea: Automated MMC Cleanup Script for Old Services
Business Case: Automated MMC Cleanup Script for Old Services

Problem Statement:

Currently after deployments in mule mmc older version of that service remain in the repository, That as a result created a lot of older service in dev and test in past 1 year. These older version will consuming a lot of storage on MMC and also cause slowness during debugging and accessing the server.

Goal Statement:

Built an automated groovy script that will delete the older services that have not been deployed.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava, Afreen Astara
Project/Team Name: UHOne Transformation 2.0
Project Nature: Mule Domain Project
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: K T, Srikanth
Onsite Approver: D'Souza, Vijay

BENEFITS

- Stabilize non-prod environments.
- Complete automated process runs daily in non-prod environments.
- Notification email with list of deleted services.
- Reduced number of INC tickets..

SOLUTION IMPLEMENTED

In addition, we will run this script nightly in DV02 and TS02 so that this situation does not happen in the future and also stabilize the DV02 and TS02. We also added notification in the groovy script which will send mail with an attached list of old Mule services which have been deleted.

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Old Services Per Year (DEV+TEST+STAGE)	1800+900+600 = 3300	500	APIs	Duplicates versions created after deployments
Effort To Remove Services manually	15 mins.	N/A	Mins.	Removing each services was practically impossible from front end , we need to do the manual cleanup from backend only
App Ops Support/Notifications	30 mins.	N/A	Mins.	App Ops needs to work on several INCs and needs to send notification after deleting services.
Total Effort (Yearly)	2475	N/A	Hrs.	(Efforts Per Service*Old Services Per Year) + (App Ops Support Efforts *Old Services per Year)
Savings	2475 Hrs		Dollars	$39 per Hrs.
Total Cost Savings: 2475 Hrs			Dollars	**$96525**

Other Details

UTILIZATION DETAILS

Automated groovy script that will delete the older services that have not been deployed. In addition, we will run this script nightly in non-prod enviroment so that this situation does not happen in the future and also stabilize the DV02 and TS02. We also added notification in the groovy script which will send mail with an attached list of old Mule services which have been deleted.

Leveraging Opportunity

This Concept can be implemented in any project where Mule-soft is being used.

SIGN-OFF MAIL

FW BIs for approval.msg

Mule Domain Project – UhOne-2.0

PROJECT BACKGROUND

Idea: Mule Domain Project .

Business Case: Generic Process to share resources using Mule Domain Project

Problem Statement:

Currently we are sharing different properties like Database connections , Server/Client certificates and other generic resources across multiple mule services. All the services are storing these information in there respective properties file. As part of our yearly changes in certificates and credential this process is not sustainable and needs a lot of rework for maintenance and thus results in high cost.

Goal Statement:

Mule supports the ability to define selected connectors as common resources and expose them to all applications deployed under a same domain.

SOLUTION IMPLEMENTED

These resources are known as shared resources, to host these we create a Mule Domain Project and then reference it on each of the projects that are meant to use the elements in it. Once defined, any Mule application associated with a particular domain can access resources in this file.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava
Project/Team Name: UHOne Transformation 2.0
Project Nature: Mule Domain Project
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: K T, Srikanth
Onsite Approver: D'Souza, Vijay

BENEFITS

Shared resources allow multiple development teams to worl in parallel using the same set of reusable connectors. Defining these connectors as shared resources at the domain level allows the team to:

- Expose multiple services within the domain through the same port
- Share the connection to persistent storage
- Share services between applications through a well-defined interface
- Ensure consistency between applications upon any changes, as the configuration is only set in one place.
- This will also help us to reduce the infrastructure and maintenance cost per year due to the mandatory changes in server certificates and login credentials.

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Services Per Year	150	150	APIs	For each API properties file needs to be added and maintained every year
Yearly Certificates, database credentials and service accounts management effort	15 hrs.	N/A		After development of generic logging API ,no testing needs to be done for event logging API separately, Just needs to verify the event logging data in tables
App Ops Support	3	15 mins.	Hrs.	
Total Effort (Yearly)	2250+450=2700	37.5	Hrs.	(Efforts Per Service*Service Per Year)+ (App Ops Support Efforts *Service per Year)
Savings		2662.5 Hrs	Dollars	$39 per Hrs.
Total Cost Savings: 2662.5 Hrs			Dollars	$103837.50

Other Details

UTILIZATION DETAILS

Generic Process to share resources using Mule Domain Project decreased the manual effort where resource has to spend time during yearly certificates, password and service account maintenance.

Leveraging Opportunity

This Concept can be implemented in any project where Mulesoft is being used.

SIGN-OFF MAIL

FW BIs for approval.msg

Event Logging Framework

PROJECT BACKGROUND

Idea: Event Logging Framework for Mule Services.

Business Case: Generic Event logging framework fro mule services

Problem Statement:

Currently All Mule applications uses the Logger (default) , Mule server will uses this and write the files. Which as results consumes the Heap Memory.

Goal Statement:

Creating a Event Logging Framework for Mule Services to perform Event Logging in DB reduced that Heap Dump issue and helps to track down issues easily using event logs tables and also provides code reusability.

SOLUTION IMPLEMENTED

Creating a reusable Event Logging Framework for Mule Services to perform Event Logging in DB , and reduced the Heap Dump issue and helps to track down issues easily using event logs tables and also provides code reusability.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava
Project/Team Name: UHOne Transformation 2.0
Project Nature: Event Logging Framework
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: K T, Srikanth
Onsite Approver: N/A

BENEFITS

- No need to create the event logging separately for each service.
- Developer can easily use the API using jar reference from maven repository.
- API is open source , easy to maintain and can be enhanced in future as well.
- Easy to access logs from event tables.
- **Implemented in IVR, Stepwise and Facets Attachments.**
- **Currently being utilized in Email Notification API as well.**

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Service Per Quarter	6	6	APIs	For each API custom event logging needs to be added.
Custom Event Logging Efforts Per Service	95	12	Hrs.	After development of generic logging API ,no testing needs to be done for event logging API separately, Just needs to verify the event logging data in tables
AppOps Debugging and Analysis Efforts Per Quarter	32	16	Hrs.	Per Week = 2hrs. Per Quarter = 32 hrs. After Improvement(Per Quarter) : 16hrs
Total Effort (Yearly)	(2220+768) = 3048	(288+256) =544	Hrs.	(Efforts Per Service*Service Per Quarter)*4 + (Debug Efforts *Service per Quarter)*4
Savings	2504 Hrs		Dollars	$39 per Hrs.
Total Cost Savings: 2504Hrs			Dollars	**$97656.00**

Other Details

UTILIZATION DETAILS

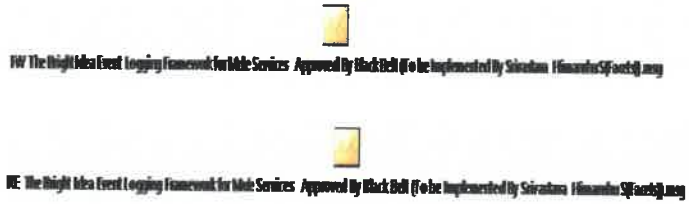
Decreased manual effort where resource has to spend time on writing Event Logging for Mule Services to perform Event Logging in DB. Also reduced that Heap Dump issue and helps to track down issues easily using event logs tables and also provides code reusability.

.

Leveraging Opportunity

This Concept can be implemented in any project where Mulesoft is being used.

SIGN-OFF MAIL



Attachment API

PROJECT BACKGROUND

Idea: Facets Attachment API
Business Case: Generic Attachment API

Problem Statement:
To insert/update attachment fields into facets at Subscriber/Member/Billing level for any given Style ID current process is not reusable and needs to be modified for any new style id or changes in existing style id, Which increases the redundancy in testing and development effort

Goal Statement:
Attachment Microservice to insert/update attachment fields into facets based on Subscriber ID at Subscriber/Member/Billing level for any given Style ID. To reduce the testing and development effort for all upcoming attachment changes, only BRC needs to be created for any new attachments

SOLUTION IMPLEMENTED

Creating a reusable attachment microservice to insert/update attachment fields into facets based on Subscriber ID at Subscriber/Member/Billing level for any given Style ID and also provides code reusability.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava
Project/Team Name: UHOne Transformation 2.0
Project Nature: Event Logging Framework
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: K T, Srikanth
Onsite Approver: N/A

BENEFITS

- No need to create the attachments flows separately for each service.
- Developer can easily use the API using POST/PUT method.
- API is open source , easy to maintain and can be enhanced in future as well.
- Easy to access logs from event tables.
- BRC driven STYLE IDs , Easy to configure for any future Attachments needs.

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Attachment Request Per Quarter	2	2	APIs	For each API Attachment requirement per quarter
Custom Event Logging Efforts Per Service	160	8	Hrs.	After development of generic Attachment API ,no testing needs to be done for attachments process separately, Just needs to verify the attachment tables.
Total Effort (Yearly)	(320*4) = 1280	(16*4) =64	Hrs.	(Efforts Per Service*Service Per Quarter)*4
Savings		1216	Hrs.	$39 per Hrs.
Total Cost Savings: 1216 Hrs			Dollars	**$47,424.00**

Other Details

UTILIZATION DETAILS

Attachment Microservice to insert/update attachment fields into facets based on Subscriber ID at Subscriber/Member/Billing level for any given Style ID. To reduce the testing and development effort for all upcoming attachment changes, only BRC needs to be created for any new attachments

.

Leveraging Opportunity

This Concept can be implemented in any project where Attachment needs to be done.

SIGN-OFF MAIL

Automated Member Creation

PROJECT BACKGROUND

Idea: Automated Member Creation
Business Case: Automated Member Creation
Problem Statement:
Currently for testing or training purpose subscriber needs to be created manually which is a time taking process, User need to pass whole demographics as well as billing and rate level details. This process is very frequent in our test environment and data needs to be created on a daily basis
Goal Statement:
Built an automated process using FXI which creates subscribers in a click.

OTHER DETAILS

Idea Logged By: Himanshu S Srivastava, Afreen Astara
Project/Team Name: UHOne Transformation 2.0
Project Nature: Mule Domain Project
IT Leader: Srinivas Ramsagar
Business Segment: OGS
ADG:
ADG Leader:
Offshore approver: K T, Srikanth
Onsite Approver: D'Souza, Vijay

BENEFITS

- Remove manual effort needed for testing claims and Product go live.
- Complete automated process runs daily in non-prod environments.
- Notification email with list of subscriber ID.
- Data load specific to user defined Test case.
- For random data load , script used fake data generator .

SOLUTION IMPLEMENTED

In addition, we can use this process to create subscriber for or claims load process as well with specific name and address information.

Financial Impact

Description	Before Improvement	After Improvement	Units	Comments
Subscriber created Per Month (DEV+TEST+STAGE)	800+2400+800 = 4000	4000	SBSB	
Effort To Create Data manually	15 mins.	N/A	Mins.	Creating bulk data is practically impossible from front end and needs a lot of manual effort and validation
Total Effort (Yearly)	(4000*12*15)/60 = 12000	N/A	Hrs.	(Efforts Per subscriber Data*Test Data request Per Year)
Savings	12000 Hrs		Dollars	$39 per Hrs.
Total Cost Savings: 468000 Hrs			Dollars	**$468000**

Other Details

UTILIZATION DETAILS

Automated bulk load will create any number of test cases without manual intervention . In addition, we can use this process to create subscriber for claims load process as well with specific name and address information. It also send email with an attached list of subscriber that has been created.

Leveraging Opportunity

This Concept can be implemented in any project where Facets is being used.

SIGN-OFF MAIL