PPC Mnemosyne Web Service for Deltas

Mnemosyne Rest API

One call to start the cleansing of the delta records. The call will start the workflow tasks, i.e. a pipeline of actions. In summary the actions that will be executed are:

1. Call a stored procedure (e.g., **`DeltaPrepare`**) in Synapse DB. This procedure will prepare the input table, i.e., it will copy the modified records to a table with known (agreed) name. The schema of the table will be the same as the schema we work. All records of this table will be processed from Mnemosyne with no filtering. There is no need for Mnemosyne to know a range of dates for records that will be processed. The second task of the **`DeltaPrepare`** stored procedure is to clear (truncate) the four output tables that will be filled with the processed records.
2. The input table will be transferred to SQL Server.
3. The processing of the records will start. The process contains several distinct tasks and creates several CSV files. At the end of processing of every task the web service will call a Synapse Stored procedure (e.g., **`DeltaClassLogging`**) with the progress status, i.e., task executed. In case of an error there will be an indication or error code.
4. The CSV files will be loaded in SQL Server and post processing tasks will be executed to produce the four output tables, one for each source. In a newer version a part of the CSV files will be loaded to Synapse and will be merged with the final table at the last processing step.
5. The produced/output tables will be transferred to Synapse SQL
6. The web service will call a stored procedure in Synapse SQL (e.g., **`DeltaEndOfProcessing`**) that will indicate the end of the processing from Mnemosyne and will prepare the table for IIS. In case of an error the **`DeltaEndOfProcessing`** will not be called.

There are three endpoints of the REST call presented below:

- `http://<mnemosyne_host_ip>:10001/ppc/mnemosyne/v2/NlpCleanse?gr_input_table=<INPUT_TABLE_NAME>`
  This call will start the workflow from the point it stopped. In case that the previous call finished successfully the workflow will start a new process cycle. In case we had a problem and the workflow stopped, the next time it will continue from the failed task. In case that the workflow is already run the call will return information for the task that runs (e.g., task number, elapsed time, ...). The parameter `gr_input_table` holds the name of the produced table with Golden Records and will be passed to the `[predicta].[DeltaSynapsePrepare]` stored procedure. The value of the parameter must not contain the character '_' because it is used as a replacement for space characters in the processing scripts.

- http://<mnemosyne_host_ip>:10001/ppc/mnemosyne/v2/NlpCleanse/Reset
  This call will stop the workflow and reset the counter (task number) in order the next call will start the process from the beginning..
- http://<mnemosyne_host_ip>:10001/ppc/mnemosyne/v2/NlpCleanse/Output
  With the call of the web service we will bring the log file from the server. This is a more detailed log file than the table in synapse.

The first two calls will return immediately with a status string that will indicate if workflow started successfully (“OK”) or an info JSON message in case the workflow runs (1st URL above) or an error message in case there is a problem in the Mnemosyne environment and workflow could not start. As described above the result of the processing will be notified through the stored procedures **`DeltaProcessing`** and **`DeltaEndOfProcessing`**.

Delta Workflow Actions

1. SynapsePreProcessing
2. Synapse2Sql
3. SqlPreProcessing
4. ProcessAddresses
5. ProcessCustomerNames
6. TranslateFields
7. ClearVatFields
8. ClearTelFields
9. ClearIdFields
10. ClearEmailFields
11. LoadResults
12. SqlPostProcessing
13. Sql2Synapse
14. SynapsePostProcessing

Synapse Stored Procedures

Store procedure for logging in Synapse DB Table
`[predicta].[DeltaClassLogging]`

SQL Stored Procedures

-

Synapse Tables

The table with trace log messages
`[predicta].[mnemosyne_DeltaProcessingLog]`

The table with running configuration, status and history
[predicta].[mnemosyne_DeltaProcessingRunnings]

The tables of the four (4) sources with the analyzed results for all the customers
[predicta].[Source_SAP]
[predicta].[Source_EBILL]
[predicta].[Source_EVALUE]
[predicta].[Source_FAIDRA]

SQL Server Tables

The table with cached results of matched addresses in Terra address database
[dbo].[cache_Address_Matched_Terra]

The table with cached results of customers' name analysis
[dbo].[cache_Customer_Name_Analysis]

The table with cached results of clearing customers' name
[dbo].[cache_Cleared_Customer_Name]

Files

The following four files keep the state of the workflow process. The FLOW_DELTA_POS file keeps the last executed task. It is used in case the process stops and we want to continue it. The FLOW_DELTA_MESSAGE file keeps the last message from the corresponding running task in the FLOW_DELTA_POS file. The bin/pipeline_delta.log contains the output of the shell script process. Finally the output/pipeline_delta.log file has the sequence of messages that are presented in the FLOW_DELTA_MESSAGE file.
${MHOME}/bin/FLOW_DELTA_POS
${MHOME}/bin/FLOW_DELTA_MESSAGE
${MHOME}/bin/pipeline_delta.log
${MHOME}/output/pipeline_delta.log

The Mnemosyne Java Command file (jcm) that executes the Synapse Stored Procedure
${MHOME}/bin/Commands/ppc/exec_delta_processing.jcm

Shell Script

${MHOME}/bin/run_ppc_delta_pipeline.sh

SynapsePreProcessing

It executes the preprocessing action in Synapse SQL. The action is implemented in a stored procedure. It fills the input delta table and truncates the output tables that will receive the

results. For filling the input delta table the date of the previous run must be checked and the condition is:
[curated_sap_isu].[General_Customer_Master_Table].[Last_Update_Date] >= [predicta].[mnemosyne_DeltaProcessingRunnings].[Last_Running_Date]

Shell Scripts

```
TASK=1
TASKNAME="SynapsePreProcessing"
MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Pre-processing
    tasks. Create and fill the input delta files."

${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_preprocess.sh "synapse"
```

Synapse Stored Procedures

The procedure clears the delta tables that will be the output of the whole procedure and It prepares (fill) the input delta table.
[predicta].[DeltaSynapsePrepare]

SQL Stored Procedures

-

Synapse Tables

The input table that was prepared (filled) with delta records.
[predicta].[General_Customer_Master_Table_Delta_For_Cleansing]

The four (4) delta source tables that will be the output of the workflow process.
[predicta].[mnemosyne_Source_SAP_Delta]
[predicta].[mnemosyne_Source_EBILL_Delta]
[predicta].[mnemosyne_Source_EVALUE_Delta]
[predicta].[mnemosyne_Source_FAIDRA_Delta]

The tables with the delta cleared “identifiers” fields that will be the output of the workflow process.
[predicta].[DELTA_CLVAT_VAT_ID];
[predicta].[DELTA_CLID_AT];
[predicta].[DELTA_CLID_Passport];
[predicta].[DELTA_CLTel_Cellphone_Number];
[predicta].[DELTA_CLTel_Fixed_Phone_Number];
[predicta].[DELTA_CLEMAIL_Email];
[predicta].[DELTA_CLTel_B2B_Fixed_Phone_Number];
[predicta].[DELTA_CLEMAIL_B2B_Email];
[predicta].[DELTA_CLEMAIL_Faidra_Email];

```
[predicta].[DELTA_CLVAT_Faidra_VAT_ID];
[predicta].[DELTA_CLTEL_Faidra_Phone_Number];
[predicta].[DELTA_CLTEL_Faidra_Phone_Number_2];
[predicta].[DELTA_CLVAT_Evalue_VAT_ID];
[predicta].[DELTA_CLID_Evalue_ID_Number];
[predicta].[DELTA_CLEMAIL_Evalue_Email];
[predicta].[DELTA_CLTEL_Evalue_Call_Back_Phone];
[predicta].[DELTA_CLTEL_Evalue_Phone_1];
[predicta].[DELTA_CLTEL_Evalue_Phone_2];
[predicta].[DELTA_CLTEL_Evalue_Phone_3];
[predicta].[DELTA_CLTEL_Evalue_Phone_4];
[predicta].[DELTA_CLTEL_Evalue_Phone_5];
[predicta].[DELTA_CLTel_Ebill_Telephone];
[predicta].[DELTA_CLTel_Ebill_Mobile];
[predicta].[DELTA_CLVAT_Ebill_AFM];
[predicta].[DELTA_CLID_Ebill_AT];
[predicta].[DELTA_CLEMAIL_Ebill_Email];
```

The table with the delta phonetic transformed fields in English that will be the output of the workflow process.

```
[predicta].[delta_Greeklish_results];
```

SQL Server Tables

-

Files

The jcm that is used

```
${MHOME}/bin/Commands/ppc/exec_delta_synapseprepare.jcm
```

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to call the Preprocessing stored procedure in Synapse

Synapse2Sql

It transfers the input table from Synapse to SQL DB

Shell Scripts

```
TASK=2
TASKNAME="Synapse2Sql"
```

```
    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Copy input
    table from Synapse to SQL Server"

${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_transfer.sh "synapse2sql"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

The input table that will be copied to SQL Server.
`[predicta].[General_Customer_Master_Table_Delta_For_Cleansing]`

SQL Server Tables

`[dbo].[delta_General_Customer_Master_Table_For_Cleansing]`

Files

The jcm that is used
`${MHOME}/bin/Commands/ppc/exec_delta_synapse2sql.jcm`

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to copy the input table from synapse to the output table in SQL

SqlPreProcessing

It executes the preparation actions inside the SQL Server. These actions prepare the tables that will be used in the following cleansing actions from Mnemosyne.The first action is the truncation of the tables and then the output tables are filled with the matched values from the cache tables. The unmatched values are inserted into the pending tables and will be analyzed in next steps.

For filling with the cached matched values of cleared customers names the trimmed input value of a name is checked to be equal with a value in the field ‘Name’ of cache table ‘`cache_Cleared_Customer_Name`’. For the cached matched values of customers analysis names the value `trim(CONCAT_WS('  ', trim(Last_Name), trim(First_Name),`

trim(Father_Name))) is checked to be equal with a value in the field 'Name' of cache table 'cache_Customer_Name_Analysis'. Respectively, for addresses the value trim(concat_ws(' ', trim(Street), trim(House_Number), trim(Postal_Code), trim(Municipality_Name), trim(Region))) is checked to be equal with a value in the field 'Address' of cache table 'cache_Address_Matched_Terra'.

Shell Scripts

```
TASK=3
    TASKNAME="SqlPreProcessing"
    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Pre-processing
    tasks in SQL Server. Prepare the cleansing data."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_preprocess.sh "sql"
```

Synapse Stored Procedures

-

SQL Stored Procedures

[dbo].[DeltaSqlPrepare]

Synapse Tables

-

SQL Server Tables

Temporary delta table for keeping the cleared customer names
[dbo].[delta_Cleared_Customer_Name]

Temporary delta table for keeping the results of the customer names analysis
[dbo].[delta_Cleared_Customer_Name]

Temporary delta table for keeping the results of the addresses matching in Terra's database
[dbo].[delta_Address_Matched_Terra]

Temporary delta tables which keep the pending customer names for clearing
[dbo].[delta_Cleared_Customer_Name_pending]
[dbo].[delta_Cleared_Customer_Name_pending_dupl]

Temporary delta tables which keep the pending customer names for analyzing
[dbo].[delta_Customer_Name_Analysis_pending]
[dbo].[delta_Customer_Name_Analysis_pending_dupl]

Temporary delta tables which keep the pending addresses for matching in Terra's database

[dbo].[delta_Address_pending]
[dbo].[delta_Address_pending_dupl]

Files

The jcm that is used
${MHOME}/bin/Commands/ppc/exec_delta_sqlprepare.jcm

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to call the [dbo].[DeltaSqlPrepare] stored procedure in SQL Server

ProcessAddresses

Process the pending addresses of the delta dataset that have not matched with the cached ones. Firstly, the pending addresses are exported to a CSV file. Then Mnemosyne's mechanism resolves these addresses using TERRA DB and writes the results in an output CSV File. Finally, the results are imported into SQL Server DB.

Shell Scripts

```
TASK=4
    TASKNAME="ProcessAddresses"

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Address
    Processing: Export addresses to CSV file."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_dbexport.sh "address"

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Address
    Processing: Resolve addresses using TERRA DB."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_synthesis.sh "address"

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Address
    Processing: Import the results to SQL DB."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_dbimport.sh "address"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

-

SQL Server Tables

[dbo].[delta_Address_pending]
[dbo].[delta_Address_Matched_Terra]

Files

The jcm that used for resolving pending addresses
`${MHOME}/bin/Commands/ppc/delta_analysis_addresses_disk.jcm`

The input file with the pending addresses:
`data/ppc/delta/addresses/ppc_delta_addresses_pending.txt`

The file with the results for the resolved pending addresses:
`data/ppc/delta/addresses/ppc_delta_addresses_pending.txt.output`

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to call the script that `exports addresses to CSV file`
3. Exec the shell script to write to the Synapse Log
4. Exec shell script to call the script that `resolves addresses using TERRA DB`
5. Exec the shell script to write to the Synapse Log
6. Exec shell script to call the script that `imports the results into SQL Server DB`

ProcessCustomerNames

Process the pending customers' names of the delta dataset that have not matched with the cached ones. Firstly, the pending customers' names are exported into two CSV files. One file for the names that must be cleared and the other for those that must be analyzed. Then Mnemosyne’s mechanism computes these files and writes the results in two output CSV files. Finally, the results are imported into SQL Server DB.

Shell Scripts

```
TASK=5
    TASKNAME="ProcessCustomerNames"
```

```
    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Customer Names
    Processing: Export customer names to CSV file"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_dbexport.sh "customer"

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Customer Names
    Processing: Analysis of customer names"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_synthesis.sh "customer"

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.  Customer Names
    Processing: Import the results to SQL DB"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_dbimport.sh "customer"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

-

SQL Server Tables

[dbo].[delta_Customer_Name_Analysis_pending]
[dbo].[delta_Customer_Name_Analysis_results]
[dbo].[delta_Customer_Name_Analysis]

[dbo].[delta_Cleared_Customer_Name_pending]
[dbo].[delta_Cleared_Customer_Name_results]
[dbo].[delta_Cleared_Customer_Name]

Files

The jcm that used for analyzing pending customer names
`${MHOME}/bin/Commands/ppc/delta_customer_name_analysis.jcm`

The input file with the pending customer names for analyzing:
`data/ppc/delta/customers/ppc_delta_customer_name_analysis_pending.txt`

The file with the results for the analyzed pending customer names:

data/ppc/delta/customers/ppc_delta_customer_name_analysis_pending.txt.output

The jcm that used for clearing pending customer names
${MHOME}/bin/Commands/ppc/delta_cleared_customer_name.jcm

The input file with the pending customer names for clearing:
data/ppc/delta/customers/ppc_delta_cleared_customer_name_pending.txt

The file with the results for the cleared pending customer names:
data/ppc/delta/customers/ppc_delta_cleared_customer_name_pending.txt.output

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to call the script that exports the pending customer names for clearing & analyzing to CSV file
3. Exec the shell script to write to the Synapse Log
4. Exec shell script to call the script that clears & analyzes the pending customer names
5. Exec the shell script to write to the Synapse Log
6. Exec shell script to call the script that imports the results into SQL Server DB

TranslateFields

It executes the translation of text fields from Greek alphabet to English phonetic alphabet. Firstly, it calls a stored procedure in SQL Server to prepare all the text fields for translation and also updates the cache tables with the new values. Next, the Mnemosyne mechanism reads the values of the text fields, does the translation and writes the results in a JSON file. Finally, the results are imported into SQL Server DB.

Shell Scripts

```
TASK=6
    TASKNAME="TranslateFields"

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Translation:
    Prepare text fields for translation to english phonetic alphabet."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_translate_prepare.sh

    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Translation:
    Translate fields"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_translate_process.sh
```

```
    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Translation:
    Load translated fields"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_translate_load.sh
```

Synapse Stored Procedures

-

SQL Stored Procedures

The procedure prepares the text fields for translation
[dbo].[DeltaSqlPrepareTranslation]

The procedure updates delta & cache tables with the resulted values from the delta pending tables
[dbo].[DeltaSqlPrepareCache]

The procedure prepares the text fields in SAP delta table for translation
[dbo].[DeltaSqlPrepareSourceSap]

The procedure prepares the text fields in Ebill delta table for translation
[dbo].[DeltaSqlPrepareSourceEbill]

The procedure prepares the text fields in Evalue delta table for translation
[dbo].[DeltaSqlPrepareSourceEvalue]

The procedure prepares the text fields in Faidra delta table for translation
[dbo].[DeltaSqlPrepareSourceFaidra]

Synapse Tables

-

SQL Server Tables

[dbo].[cache_Customer_Name_Analysis]
[dbo].[cache_Cleared_Customer_Name]
[dbo].[cache_Address_Matched_Terra]

[dbo].[delta_Source_SAP]
[dbo].[delta_Source_EBILL]
[dbo].[delta_Source_EVALUE]
[dbo].[delta_Source_FAIDRA]

[dbo].[delta_Greeklish_results]

Files

The jcm that used for `text fields translation to english phonetic alphabet`
`${MHOME}/bin/Commands/ppc/delta_greeklish.jcm`

The file with the results `text fields translation to english phonetic alphabet`:
`data/ppc/delta/greeklish/PPC_CUSTOMER_NAMES_LATIN.json`

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to call the script that `prepares text fields for translation to english phonetic alphabet`
3. Exec the shell script to write to the Synapse Log
4. Exec shell script to call the script that `translates fields`
5. Exec the shell script to write to the Synapse Log
6. Exec shell script to call the script that `loads translated fields into SQL Server DB`

ClearTelFields

This task clears and extracts the correct values from fields that contain telephone numbers, fixed or mobile.

Shell Scripts

```
TASK=7
    TASKNAME="CleanTelFields"
    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Clean Telephone
    fields."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_fields_clean.sh "tel"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

-

SQL Server Tables

-

Files

The jcm that used for the cleansing of telephone fields is:
`bin/Commands/ppc/delta_clean_tel.jcm`

The task will produce the following files:

```
data/ppc/results/DELTA_CLTEL_Cellphone_Number.csv
data/ppc/results/DELTA_CLTEL_Fixed_Phone_Number.csv
data/ppc/results/DELTA_CLTEL_B2B_Fixed_Phone_Number.csv
data/ppc/results/DELTA_CLTEL_Ebill_Telephone Ebill_Mobile.csv
data/ppc/results/DELTA_CLTEL_Faidra_Phone_Number.csv
data/ppc/results/DELTA_CLTEL_Faidra_Phone_Number_2.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_1.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_2.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_3.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_4.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_5.csv
data/ppc/results/DELTA_CLTEL_Evalue_Call_Back_Phone.csv
```

Actions

```
TEL_FIELDS=( Cellphone_Number Fixed_Phone_Number
                          B2B_Fixed_Phone_Number Ebill_Telephone
Ebill_Mobile
                          Faidra_Phone_Number Faidra_Phone_Number_2
                          Evalue_Phone_1 Evalue_Phone_2 Evalue_Phone_3
                          Evalue_Phone_4  Evalue_Phone_5
Evalue_Call_Back_Phone    )
```

The cleansing configuration of Mnemosyne analyzer presented below:

```
<arg class="Integer"
name="analyzer.cleansing.normalization.method">2</arg>
<arg class="Integer" name="analyzer.cleansing.tokenization.method">0</arg>
<arg class="WLocale"
name="analyzer.cleansing.locale">dicts/ppc/en_number.loc</arg>

<arg class="String"  name="analyzer.cleansing.valid.chars">+ /*-
01234567890</arg> <!-- alphabet                        -->
<arg class="Boolean"
name="analyzer.cleansing.remove.space.chars">false</arg>    <!--  3rd step
clear spaces        -->
```

```
<arg class="Boolean"
name="analyzer.cleansing.remove.invalid.chars">true</arg>   <!--  3rd step
clear invalid chars -->
<arg class="String"  name="analyzer.cleansing.extract.patterns">
<!--  4th step extract patterns    -->
      <![CDATA[
            "(^[0-9]{10})(?=[ ]?[ -*/][ ]?[0-9]{4,10})":="1"
            "(^[0-9]{5,9})([ ]?[ -*/][ ]?)([0-9]{10})":="3"
            "(^[0-9]{10})(?=[0-9]{10})":="1"
            "(^[0-9]{10})(?=[-][0-9]{1,2})":="1"
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.replace.patterns">  <!--
5th step replace patterns -->
      <![CDATA[
            "^[+]30":=""
            "^0030":=""
            "O":="0"
            "^30(?=[0-9]{10})":=""
            "^0(?=[0-9]{9})":="2"
            " ":=""
            "^[01](?=2[0-9]{9})":=""
            "/":=""
            "-":=""
            "\*":=""
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.valid.patterns">    <!--
6th step validation & tag patterns -->
      <![CDATA[
            "2[0-9]{9}":=Fixed_Format
            "6[0-9]{9}":=Cell_Format
      ]]>
</arg>
<arg class="String"
name="analyzer.cleansing.unknown.pattern.info">Unknown_Format</arg>  <!--
6th step value for unknown values         -->
<arg class="String"
name="analyzer.cleansing.invalid.value.info">Invalid_value</arg>       <!-
-  6th step value for invalide/deummy values -->
<arg class="String"
name="analyzer.cleansing.dummy.value.info">Invalid_value</arg>       <!--
6th step value for invalide/dummy values  -->
<arg class="String"  name="analyzer.cleansing.dummy.values">
<!--   1st Check dummy value                    -->
      <![CDATA[
    "6900000000"
```

```
            "0000"
            "0000000000"
            "6933333333"
            "6940000000"
            "00"
            "3"
            "**********"
            "-"
            "1"
            "000"
            "69"
            "7"
            "C"
            "24"
            "001"
            "6"
            "21"
            "2"
            "4"
            "000000"
            "00000"
            "0000000"
            "1111111111"
            "000000000"
            "0000"
            "+"
            "210"
            "---"
            "5"
            "00000000"
            "9999999999"
            "2541099999"
            "2610311111"
            "2381023333"
            "2108080011"
            "6936779559"
            "2810314700"
            "2103287100"
            "2721022222"
            "6907648456"
            "2105700000"
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.dummy.regexps"> <!--   2nd
Check dummy regexp -->
      <![CDATA[
            ".{1,4}"
```

```
            "[^0-9]{1,}"
            "2[0-9]{3}000000"
            "2[0-9]{3}011111"
            "2[0-9]{3}022222"
            "2[0-9]{3}033333"
            "2[0-9]{3}044444"
            "2[0-9]{3}055555"
            "2[0-9]{3}066666"
            "2[0-9]{3}077777"
            "2[0-9]{3}088888"
            "2[0-9]{3}099999"
            ".{1,2}0000"
            ".{1,2}1111"
            ".{1,2}2222"
            ".{1,2}3333"
            ".{1,2}4444"
            ".{1,2}5555"
            ".{1,2}6666"
            ".{1,2}7777"
            ".{1,2}8888"
            ".{1,2}9999"
            ".*000000.*"
            ".*111111.*"
            ".*222222.*"
            ".*333333.*"
            ".*444444.*"
            ".*555555.*"
            ".*666666.*"
            ".*777777.*"
            ".*888888.*"
            ".*999999.*"
      ]]>
</arg>
```

ClearIdFields

This task clears the field values that represent a type of ID like Person ID, Passport, Army ID, etc.

Shell Scripts

```
TASK=8
    TASKNAME="CleanIdFields"
MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Clean Ids (Police
Id, Passport) fields."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
```

```
${MHOME}/bin/run_ppc_delta_fields_clean.sh "id"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

-

SQL Server Tables

-

Files

The jcm that is used for the cleansing of ID fields is:
`bin/Commands/ppc/delta_clean_id.jcm`

The task will produce the following files:

```
data/ppc/results/DELTA_CLID_AT.csv
data/ppc/results/DELTA_CLID_Ebill_AT.csv
data/ppc/results/DELTA_CLID_Evalue_ID_Number.csv
data/ppc/results/DELTA_CLID_Passport.csv
```

Actions

For each of the following fields that contain values of person IDENTIFIER we apply the rules of the xml configuration script that follows

```
ID_FIELDS=(AT Ebill_AT Evalue_ID_Number Passport)
<arg class="Integer"
name="analyzer.cleansing.normalization.method">2</arg>
<arg class="Integer" name="analyzer.cleansing.tokenization.method">0</arg>
<arg class="WLocale"
name="analyzer.cleansing.locale">dicts/ppc/en_number.loc</arg>

<arg class="String"
name="analyzer.cleansing.valid.chars">0123456789ABΓΔEZHΘIKΛMNΞOΠPΣTYΦXΨΩAB
CDEFGHIJKLMNOPQRSTVUYXWZabcdefghijklmnopqrstvuyxwz</arg>
<arg class="Boolean"
name="analyzer.cleansing.remove.space.chars">true</arg>   <!--  3rd step
clear spaces        -->
```

```
<arg class="Boolean"
name="analyzer.cleansing.remove.invalid.chars">true</arg>  <!--  3rd step
clear invalid chars -->
<arg class="String"  name="analyzer.cleansing.extract.patterns.1">  <!--
4th step extract patterns    -->
      <![CDATA[
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.replace.patterns">     <!--
5th step replace patterns -->
      <![CDATA[
            "\.":=""
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.valid.patterns">  <!--  6th
step validation & tag patterns -->
      <![CDATA[
            "(^[ABΓΔEZHΘIKΛMNΞOΠPΣTYΦXΨΩ]{1,2}[0-9]{6}$)":=AT_Format

      "(^[ABCDEFGHIJKLMNOPQRSTVUYXWZabcdefghijklmnopqrstvuyxwz]{1,2}[0-
9]{7}$)":=Passport_Format
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.valid.group.patterns">   <!-
-  7th step validation grous & tag patterns -->
      <![CDATA[
            "(^(ΔIAB|APΔ|APΔI|ΔIABAT|AΔ|DIAB)[-. /]?)([A-
ZABEZHIKMNOPTYX]{0,2}[0-9]{6,12})$":=Passport_Format,3
      ]]>
</arg>
<arg class="String"
name="analyzer.cleansing.unknown.pattern.info">Unknown_Format</arg>
<arg class="String"
name="analyzer.cleansing.invalid.value.info">Invalid_value</arg>
<arg class="String"
name="analyzer.cleansing.dummy.value.info">Invalid_value</arg>       <!--
6th step value for invalide/dummy values  -->
<arg class="String"  name="analyzer.cleansing.dummy.values">    <!--   1st
Check dummy value -->
      <![CDATA[
    "01111"
            "0000000"
            "0"
            "1"
            "11"
            "111"
            "1111"
```

```
            "11111"
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.dummy.regexps">   <!--   2nd
Check dummy regexp -->
      <![CDATA[
            ".*[0-9]{2}00000.*"
            ".*[0-9]{2}11111.*"
            ".*[0-9]{2}22222.*"
            ".*[0-9]{2}33333.*"
            ".*[0-9]{2}44444.*"
            ".*[0-9]{2}55555.*"
            ".*[0-9]{2}66666.*"
            ".*[0-9]{2}77777.*"
            ".*[0-9]{2}88888.*"
            ".*[0-9]{2}99999.*"
            "^[0-9]{3}$"
            ".*000000.*"
            ".*111111.*"
            ".*222222.*"
            ".*333333.*"
            ".*444444.*"
            ".*555555.*"
            ".*666666.*"
            ".*777777.*"
            ".*888888.*"
            ".*999999.*"
            "^[.]+$"
            "^.{1,2}$"
      ]]>
</arg>
```

ClearEmailFields

This task clears and extracts the correct values from fields that contain emails

Shell Scripts

```
TASK=9
    TASKNAME="CleanEmailFields"
MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Clean Email
fields."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_fields_clean.sh "email"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

-

SQL Server Tables

-

Files

The jcm that used for the cleansing of email fields is:
`bin/Commands/ppc/delta_clean_email.jcm`

The task will produce the following files:
`data/ppc/results/DELTA_CLEMAIL_B2B_Email.csv`
`data/ppc/results/DELTA_CLEMAIL_Ebill_Email.csv`
`data/ppc/results/DELTA_CLEMAIL_Email.csv`
`data/ppc/results/DELTA_CLEMAIL_Evalue_Email.csv`
`data/ppc/results/DELTA_CLEMAIL_Faidra_Email.csv`

Actions

The fields that holds an email value and the configuration script that recognizes and corrects the erroneous values follows

```
EMAIL_FIELDS=( Email B2B_Email Ebill_Email Faidra_Email Evalue_Email )

<arg class="Integer"
name="analyzer.cleansing.normalization.method">2</arg>
<arg class="Integer" name="analyzer.cleansing.tokenization.method">0</arg>
<arg class="WLocale"
name="analyzer.cleansing.locale">dicts/ppc/en_number.loc</arg>

<arg class="String"  name="analyzer.cleansing.valid.chars.1"></arg>
<!-- NO special alphabet, all UNICODE chars -->
<arg class="Boolean"
name="analyzer.cleansing.remove.space.chars">false</arg>    <!--  3rd step
clear spaces        -->
```

```
<arg class="Boolean"
name="analyzer.cleansing.remove.invalid.chars">false</arg>  <!--  3rd step
clear invalid chars -->
<arg class="String"  name="analyzer.cleansing.extract.patterns">
<!--  4th step extract patterns    -->
      <![CDATA[
            "(^.*[<])([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-
.]+)([>].*$)":="2"
            "(^['])([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-
.]+)([']$)":="2"
            "([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)([
(].*$)":="1"
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.replace.patterns">      <!--
5th step replace patterns -->
      <![CDATA[
            "'":=""
    "'":=""
    "gmailcom$":="gmail.com"
    "gmail$":="gmail.com"
            "gmail.con$":="gmail.com"
            "@gmail.com.*":="@gmail.com"
            "gmail.c[ipm][mo]$":="gmail.com"
            "gmai.com[a-z]$":="gmail.com"
            "gmail.cim$":="gmail.com"
            "gmail.c$":="gmail.com"
            "gmail.ciom":="gmail.com"
            "hotmailgr$":="hotmail.gr"
            "hotmailcom$":="hotmail.com"
            "hotmail.con$":="hotmail.com"
            "hotmail.c$":="hotmail.com"
            "yahoo.con$":="yahoo.com"
            "outlook.con$":="outlook.com"
            "\.couk$":="\.co.uk"
            "\.cpm$":="\.com"
            "\.CO.KK$":="\.CO.UK"
            "\.g$":="\.gr"
            ",gr":="\.gr"
            ",com":="\.com"
            "\.r$":="\.gr"
            "\.gr[a-z]$":="\.gr"
            "\.gr[0-9]+$":="\.gr"
            "\.comm$":="\.com"
            "\.col$":="\.com"
            "\.som$":="\.com"
            "\.xom$":="\.com"
```

```
            "\.ocm$":="\.com"
            "\.kom$":="\.com"
            "\.ccom$":="\.com"
            "\.gom$":="\.com"
            "\.vom$":="\.com"
            "\.com[A-Z]{1,3}$":=".com"
            "\.kgr$":="\.gr"
            "\.G$":="\.GR"
            "\.F\.R$":="\.GR"
            "g.mail$":="gmail.com"
            "\.coml$":=\.com"
            "otenet.or$":="otenet.gr"
            "otenet.[0-9]{2,9}$":="otenet.gr"
            "\.com[0-9]+$":="\.com"
            "\.co\.uc$":="\.co\.uk"
            "\.DOM$":=".COM"
            "yhoo.rg$":="yahoo.gr"
            "\.CPM$":="\.COM"
            "@in.dr$":="@in.gr"
            "\.commi$":="\.com"
            "\.lcom$":="\.com"
      ]]>
</arg>

<arg class="String"  name="analyzer.cleansing.valid.patterns">  <!--  6th
step validation & tag patterns -->
      <![CDATA[
            "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-
.]+$":=Email_Format
      ]]>
</arg>
<arg class="String"
name="analyzer.cleansing.unknown.pattern.info">Unknown_Format</arg>  <!--
6th step value for unknown values         -->
<arg class="String"
name="analyzer.cleansing.invalid.value.info">Invalid_value</arg>     <!--
6th step value for invalide/dummy values -->
<arg class="String"
name="analyzer.cleansing.dummy.value.info">Invalid_value</arg>       <!--
6th step value for invalide/dummy values  -->
```

ClearVatFields

This task clears and extracts the correct values from fields that contain VAT.

Shell Scripts

```
TASK=10
    TASKNAME="CleanVatFields"
    MESSAGE="TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}. Clean VAT
    fields."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_fields_clean.sh "vat"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

-

SQL Server Tables

-

Files

The jcm that used for the cleansing of VAT fields is:
`bin/Commands/ppc/delta_clean_vat.jcm`

The task will produce the following files:

```
data/ppc/results/DELTA_CLVAT_VAT_ID
data/ppc/results/DELTA_CLVAT_Faidra_VAT_ID
data/ppc/results/DELTA_CLVAT_Evalue_VAT_ID
data/ppc/results/DELTA_CLVAT_Ebill_AFM
```

Actions

The field values and the configuration script for Mnemosyne analyzer follows.

```
VAT_FIELDS=(VAT_ID Faidra_VAT_ID Evalue_VAT_ID Ebill_AFM)

<arg class="Integer"
name="analyzer.cleansing.normalization.method">2</arg>
<arg class="Integer" name="analyzer.cleansing.tokenization.method">0</arg>
<arg class="WLocale"
name="analyzer.cleansing.locale">dicts/ppc/en_number.loc</arg>
```

```
<arg class="String"
name="analyzer.cleansing.valid.chars">01234567890</arg> <!-- alphabet
-->
<arg class="Boolean"
name="analyzer.cleansing.remove.space.chars">true</arg>     <!--  3rd step
clear spaces        -->
<arg class="Boolean"
name="analyzer.cleansing.remove.invalid.chars">false</arg>   <!--  3rd
step clear invalid chars -->
<arg class="String"  name="analyzer.cleansing.replace.patterns">  <!--
5th step replace patterns -->
      <![CDATA[
            "O":="0"
            "^(?=[0-9]{8}$)":="0"
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.valid.patterns">    <!--
6th step validation & tag patterns -->
      <![CDATA[
            "[0-9]{9}":=VAT_Format
      ]]>
</arg>

      <!--  6th step validation & tag patterns -->
<arg class="Boolean"
name="analyzer.cleansing.validation.function">true</arg>
<arg class="String"
name="analyzer.cleansing.unknown.pattern.info">Unknown_Format</arg>  <!--
6th step value for unknown values         -->
<arg class="String"
name="analyzer.cleansing.dummy.value.info">Invalid_value</arg>       <!--
6th step value for invalide/dummy values  -->
<arg class="String"
name="analyzer.cleansing.invalid.value.info">Invalid_value</arg>     <!--
6th step value for invalide/dummy values  -->
<arg class="String"  name="analyzer.cleansing.dummy.values">
<!--   1st Check dummy value                    -->
      <![CDATA[
    "000000000"
            "011111111"
            "099999999"
            "000000001"
            "022222222"
            "0"
            "0000000000"
            "00000000"
```

```
            "044444444"
            "000000009"
      ]]>
</arg>
<arg class="String"  name="analyzer.cleansing.dummy.regexps"> <!--   2nd
Check dummy regexp -->
      <![CDATA[
            "[0]{7,8}\d"
            "[0]?[1]{7,9}"
            "[0]?[2]{7,9}"
            "[0]?[3]{7,9}"
            "[0]?[4]{7,9}"
            "[0]?[5]{7,9}"
            "[0]?[6]{7,9}"
            "[0]?[7]{7,9}"
            "[0]?[8]{7,9}"
            "[0]?[9]{7,9}"
      ]]>
</arg>
```

LoadResults

The task loads the csv files produced from the previous cleansing tasks to SQL Managed Server or Synapse

Shell Scripts

```
TASK=11
    TASKNAME="CleanVatFields"
    MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK-
    >${TASK}. Load results to Synapse or SQL Managed Server."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_fields_load_results.sh "synapse" # "sql"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

```
[predicta].[DELTA_CLEMAIL_B2B_Email]
[predicta].[DELTA_CLEMAIL_Ebill_Email]
```

[predicta].[DELTA_CLEMAIL_Email]
[predicta].[DELTA_CLEMAIL_Evalue_Email]
[predicta].[DELTA_CLEMAIL_Faidra_Email]
[predicta].[DELTA_CLID_AT]
[predicta].[DELTA_CLID_Ebill_AT]
[predicta].[DELTA_CLID_Evalue_ID_Number]
[predicta].[DELTA_CLID_Passport]
[predicta].[DELTA_CLTEL_B2B_Fixed_Phone_Number]
[predicta].[DELTA_CLTEL_Cellphone_Number]
[predicta].[DELTA_CLTEL_Ebill_Mobile]
[predicta].[DELTA_CLTEL_Ebill_Telephone]
[predicta].[DELTA_CLTEL_Evalue_Call_Back_Phone]
[predicta].[DELTA_CLTEL_Evalue_Phone_1]
[predicta].[DELTA_CLTEL_Evalue_Phone_2]
[predicta].[DELTA_CLTEL_Evalue_Phone_3]
[predicta].[DELTA_CLTEL_Evalue_Phone_4]
[predicta].[DELTA_CLTEL_Evalue_Phone_5]
[predicta].[DELTA_CLTEL_Faidra_Phone_Number]
[predicta].[DELTA_CLTEL_Faidra_Phone_Number_2]
[predicta].[DELTA_CLTEL_Fixed_Phone_Number]
[predicta].[DELTA_CLVAT_Evalue_VAT_ID]
[predicta].[DELTA_CLVAT_Faidra_VAT_ID]
[predicta].[DELTA_CLVAT_VAT_ID]
[predicta].[DELTA_CLVAT_Ebill_AFM]

SQL Server Tables

DELTA_CLEMAIL_B2B_Email
DELTA CL EMAIL Ebill Email
DELTA_CLEMAIL_Email
DELTA_CLEMAIL_Evalue_Email
DELTA_CLEMAIL_Faidra_Email
DELTA_CLID_AT
DELTA_CLID_Ebill_AT
DELTA_CLID_Evalue_ID_Number
DELTA_CLID_Passport
DELTA_CLTEL_B2B_Fixed_Phone_Number
DELTA_CLTEL_Cellphone_Number
DELTA_CLTEL_Ebill_Mobile
DELTA_CLTEL_Ebill_Telephone
DELTA_CLTEL_Evalue_Call_Back_Phone
DELTA_CLTEL_Evalue_Phone_1
DELTA_CLTEL_Evalue_Phone_2
DELTA_CLTEL_Evalue_Phone_3
DELTA_CLTEL_Evalue_Phone_4
DELTA_CLTEL_Evalue_Phone_5
DELTA_CLTEL_Faidra_Phone_Number

DELTA_CLTEL_Faidra_Phone_Number_2
DELTA_CLTEL_Fixed_Phone_Number
DELTA_CLVAT_Evalue_VAT_ID
DELTA_CLVAT_Faidra_VAT_ID
DELTA_CLVAT_VAT_ID
DELTA_CLVAT_Ebill_AFM

Files

The jcm files used for the load of cleansed field files are:
bin/Commands/ppc/delta_load_fields2sql.jcm
bin/Commands/ppc/delta_load_fields2synapse.jcm

The files that will be loaded to the SQL DB are:
data/ppc/results/DELTA_CLTEL_Cellphone_Number.csv
data/ppc/results/DELTA_CLTEL_Fixed_Phone_Number.csv
data/ppc/results/DELTA_CLTEL_B2B_Fixed_Phone_Number.csv
data/ppc/results/DELTA_CLTEL_Ebill_Telephone Ebill_Mobile.csv
data/ppc/results/DELTA_CLTEL_Faidra_Phone_Number.csv
data/ppc/results/DELTA_CLTEL_Faidra_Phone_Number_2.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_1.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_2.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_3.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_4.csv
data/ppc/results/DELTA_CLTEL_Evalue_Phone_5.csv
data/ppc/results/DELTA_CLTEL_Evalue_Call_Back_Phone.csv
data/ppc/results/DELTA_CLID_AT.csv
data/ppc/results/DELTA_CLID_Ebill_AT.csv
data/ppc/results/DELTA_CLID_Evalue_ID_Number.csv
data/ppc/results/DELTA_CLID_Passport.csv
data/ppc/results/DELTA_CLEMAIL_B2B_Email.csv
data/ppc/results/DELTA_CLEMAIL_Ebill_Email.csv
data/ppc/results/DELTA_CLEMAIL_Email.csv
data/ppc/results/DELTA_CLEMAIL_Evalue_Email.csv
data/ppc/results/DELTA_CLEMAIL_Faidra_Email.csv
data/ppc/results/DELTA_CLVAT_VAT_ID
data/ppc/results/DELTA_CLVAT_Faidra_VAT_ID
data/ppc/results/DELTA_CLVAT_Evalue_VAT_ID
data/ppc/results/DELTA_CLVAT_Ebill_AFM

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to load the csv files produced from the previous cleansing tasks to SQL Managed Server or Synapse

SqlPostProcessing

It executes the Post-Processing action in SQL Server. The action is implemented in a stored procedure. It calculates the final values of the fields for the 4 source delta tables from the intermediate result tables. This action could take place at Synapse during SynapsePostProcessing action in case all the intermediate results tables have been copied from SQL Server to Synapse.

Shell Scripts

```
TASK=12
    TASKNAME="SqlPostProcessing"
MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Post processing tasks. Create and fill the output files."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_postprocess.sh "sql"
```

Synapse Stored Procedures

-

SQL Stored Procedures

[dbo].[DeltaSqlEndOfProcessing]

The procedure calculates the final values of the fields at SAP delta table [dbo].[DeltaSqlEndOfProcessingSourceSap]

The procedure calculates the final values of the fields at Ebill delta table [dbo].[DeltaSqlEndOfProcessingSourceEbill]

The procedure calculates the final values of the fields at Evalue delta table [dbo].[DeltaSqlEndOfProcessingSourceEvalue]

The procedure calculates the final values of the fields at Faidra delta table [dbo].[DeltaSqlEndOfProcessingSourceFaidra]

Synapse Tables

-

SQL Server Tables

[dbo].[delta_Source_SAP]
[dbo].[delta_Source_EBILL]

[dbo].[delta_Source_EVALUE]
[dbo].[delta_Source_FAIDRA]

Files

The jcm that is used
`${MHOME}/bin/Commands/ppc/delta_sqlendprocessing.jcm`

Actions

3. Exec the shell script to write to the Synapse Log
4. Exec shell script to call the DeltaSqlEndOfProcessing stored procedure in SQL Server

Sql2Synapse

This task copies final output delta tables from SQL to Synapse Server

Shell Scripts

```
TASK=13
MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Copy output tables from SQL to Synapse Server".
TASKNAME="Sql2Synapse"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
${MHOME}/bin/run_ppc_delta_transfer.sh "sql2synapse"
```

Synapse Stored Procedures

-

SQL Stored Procedures

-

Synapse Tables

[predicta].[mnemosyne_Source_SAP_Delta]
[predicta].[mnemosyne_Source_EBILL_Delta]
[predicta].[mnemosyne_Source_EVALUE_Delta]
[predicta].[mnemosyne_Source_FAIDRA_Delta]

[predicta].[delta_Greeklish_results];

[predicta].[DELTA_CLEMAIL_B2B_Email]

[predicta].[DELTA_CLEMAIL_Ebill_Email]
[predicta].[DELTA_CLEMAIL_Email]
[predicta].[DELTA_CLEMAIL_Evalue_Email]
[predicta].[DELTA_CLEMAIL_Faidra_Email]
[predicta].[DELTA_CLID_AT]
[predicta].[DELTA_CLID_Ebill_AT]
[predicta].[DELTA_CLID_Evalue_ID_Number]
[predicta].[DELTA_CLID_Passport]
[predicta].[DELTA_CLTEL_B2B_Fixed_Phone_Number]
[predicta].[DELTA_CLTEL_Cellphone_Number]
[predicta].[DELTA_CLTEL_Ebill_Mobile]
[predicta].[DELTA_CLTEL_Ebill_Telephone]
[predicta].[DELTA_CLTEL_Evalue_Call_Back_Phone]
[predicta].[DELTA_CLTEL_Evalue_Phone_1]
[predicta].[DELTA_CLTEL_Evalue_Phone_2]
[predicta].[DELTA_CLTEL_Evalue_Phone_3]
[predicta].[DELTA_CLTEL_Evalue_Phone_4]
[predicta].[DELTA_CLTEL_Evalue_Phone_5]
[predicta].[DELTA_CLTEL_Faidra_Phone_Number]
[predicta].[DELTA_CLTEL_Faidra_Phone_Number_2]
[predicta].[DELTA_CLTEL_Fixed_Phone_Number]
[predicta].[DELTA_CLVAT_Evalue_VAT_ID]
[predicta].[DELTA_CLVAT_Faidra_VAT_ID]
[predicta].[DELTA_CLVAT_VAT_ID]
[predicta].[DELTA_CLVAT_Ebill_AFM]

SQL Server Tables

[dbo].[delta_Source_SAP]
[dbo].[delta_Source_EBILL]
[dbo].[delta_Source_EVALUE]
[dbo].[delta_Source_FAIDRA]

[dbo].[delta_Greeklish_results]

[dbo].[DELTA_CLEMAIL_B2B_Email]
[dbo].[DELTA_CLEMAIL_Ebill_Email]
[dbo].[DELTA_CLEMAIL_Email]
[dbo].[DELTA_CLEMAIL_Evalue_Email]
[dbo].[DELTA_CLEMAIL_Faidra_Email]
[dbo].[DELTA_CLID_AT]
[dbo].[DELTA_CLID_Ebill_AT]
[dbo].[DELTA_CLID_Evalue_ID_Number]
[dbo].[DELTA_CLID_Passport]
[dbo].[DELTA_CLTEL_B2B_Fixed_Phone_Number]
[dbo].[DELTA_CLTEL_Cellphone_Number]

[dbo].[DELTA_CLTEL_Ebill_Mobile]
[dbo].[DELTA_CLTEL_Ebill_Telephone]
[dbo].[DELTA_CLTEL_Evalue_Call_Back_Phone]
[dbo].[DELTA_CLTEL_Evalue_Phone_1]
[dbo].[DELTA_CLTEL_Evalue_Phone_2]
[dbo].[DELTA_CLTEL_Evalue_Phone_3]
[dbo].[DELTA_CLTEL_Evalue_Phone_4]
[dbo].[DELTA_CLTEL_Evalue_Phone_5]
[dbo].[DELTA_CLTEL_Faidra_Phone_Number]
[dbo].[DELTA_CLTEL_Faidra_Phone_Number_2]
[dbo].[DELTA_CLTEL_Fixed_Phone_Number]
[dbo].[DELTA_CLVAT_Evalue_VAT_ID]
[dbo].[DELTA_CLVAT_Faidra_VAT_ID]
[dbo].[DELTA_CLVAT_VAT_ID]
[dbo].[DELTA_CLVAT_Ebill_AFM]

Files

The jcm that is used
`${MHOME}/bin/Commands/ppc/delta_sql2synapse.jcm`

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to copy final output delta tables from SQL to Synapse Server

SynapsePostProcessing

It executes the Post-Processing action in Synapse. The action is implemented in a stored procedure. Firstly, it calculates the final values of the fields for the 4 source delta tables from the intermediate result tables. This action could take place at SQL Server during SQLPostProcessing action in case the intermediate results tables have not been copied from SQL Server to Synapse. Next, it updates the four (4) sources tables for all the customers with the analyzed results from delta tables. Finally, it renames the 4 source tables adding date (“_DD_MM_YYYY”), calls the stored procedure which creates the final input table for IIS and updates the running status table.

Shell Scripts

```
TASK=14
    TASKNAME="SynapsePostProcessing"
MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Post processing tasks. Create and fill the output files."
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
```

```
${MHOME}/bin/run_ppc_delta_postprocess.sh "synapse"
```

Synapse Stored Procedures

[predicta].[DeltaSynapseEndOfProcessing]

The procedure that updates the four (4) sources tables for all the customers with the analyzed results from delta tables
[predicta].[DeltaSynapseMnemosyneUpdateSourceTables]

The procedure that creates the final input table for IIS
[predicta].[ppc_build_ibm_table]

SQL Stored Procedures

-

Synapse Tables

The tables of the four (4) sources with the analyzed results for delta customers
[predicta].[mnemosyne_Source_SAP_Delta]
[predicta].[mnemosyne_Source_EBILL_Delta]
[predicta].[mnemosyne_Source_EVALUE_Delta]
[predicta].[mnemosyne_Source_FAIDRA_Delta]

The tables of the four (4) sources with the analyzed results for all the customers
[predicta].[Source_SAP]
[predicta].[Source_EBILL]
[predicta].[Source_EVALUE]
[predicta].[Source_FAIDRA]

The snapshot tables of the four (4) sources with the analyzed results for all the customers for the current running
[predicta].[Source_SAP_DD_MM_YYYY]
[predicta].[Source_EBILL_DD_MM_YYYY]
[predicta].[Source_EVALUE_DD_MM_YYYY]
[predicta].[Source_FAIDRA_DD_MM_YYYY]

The table with running configuration, status and history
`[predicta].[mnemosyne_DeltaProcessingRunnings]`

SQL Server Tables

-

Files

The jcm that is used
`${MHOME}/bin/Commands/ppc/exec_delta_synapseprepare.jcm`

Actions

1. Exec the shell script to write to the Synapse Log
2. Exec shell script to call the DeltaSynapseEndOfProcessing stored procedure in Synapse

---

APPENDIX: The delta pipeline script

```
#!/bin/bash

set -x
set -e
PS4="$LINENO:"


if [[ -f "env.sh" ]]; then
  source env.sh
elif [[ -f "bin/env.sh" ]]; then
  source bin/env.sh
else
  echo "No env.sh found."

  exit 1
fi

if hash free 2>/dev/null; then
    MEM=`awk '/^Mem/ {printf("%u", $2/1024 -4);}' <(free -m)`
else
    MEM=7
fi

export MEM
```

```
#cd ${MHOME}/

GR_INPUT_TABLE="GR_INPUT_TABLE"
POSITION_FILE=${MHOME}/bin/FLOW_DELTA_POS
MESSAGE_FILE=${MHOME}/bin/FLOW_DELTA_MESSAGE
PIPELINE_LOG=${MHOME}/output/pipeline_delta.log

FIRST=1
LAST=14
TASK=$FIRST
TASKNAME=""
MESSAGE=""

if test -f "$POSITION_FILE"; then
  TASK=$(< $POSITION_FILE)
fi

if test "$#" -eq 1; then
  GR_INPUT_TABLE=$1
else
  TASKNAME="DeltaPipelineProcess"
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK=0. Wrong
call! Missing the GR_INPUT_TABLE."

  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo '' >> ${PIPELINE_LOG}
  echo ${MESSAGE} >> ${PIPELINE_LOG}

  exit 1
fi

if test -f "$POSITION_FILE"; then
  TASK=$(< $POSITION_FILE)
fi

if [ "$TASK" -ge $FIRST ] && [ "$TASK" -le $LAST ];
then
  echo TASK=${TASK}
else
    echo "**** WRONG TASK POSITION ****"
fi

echo $TASK > $POSITION_FILE

############## Pipeline of tasks for deltas
```

```
ErrorHandler()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Error in task's script!."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE

  exit 1
}

SynapsePreProcessing()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Pre-processing tasks. Create and fill the input delta files."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="SynapsePreProcessing"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_preprocess.sh "synapse" $GR_INPUT_TABLE
}

Synapse2Sql()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Copy input table from Synapse to SQL Server"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="Synapse2Sql"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_transfer.sh "synapse2sql"
}

SqlPreProcessing()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Pre-processing tasks in SQL Server. Prepare the cleansing data."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="SqlPreProcessing"

  MESSAGE="${MESSAGE// /_}"
```

```
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_preprocess.sh "sql"
}

ProcessAddresses()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Address Processing: Export addresses to CSV file"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="ProcessAddresses"

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_dbexport.sh "address"

  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Address Processing: Resolve addresses using TERRA DB"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_synthesis.sh "address"

  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Address Processing: Import the reults to SQL DB"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_dbimport.sh "address"
}

ProcessCustomerNames()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Customer Names Processing: Export customer names to CSV file"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="ProcessCustomerNames"

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_dbexport.sh "customer"

  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Customer Names Processing: Analysis of customer names"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
```

```
  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_synthesis.sh "customer"

  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Customer Names Processing: Import the reults to SQL DB"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_dbimport.sh "customer"
}

TranslateFields()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Translation: Prepare text fields for translation to english phonetic
alphabet."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="TranslateFields"

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_translate_prepare.sh

  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK-
>${TASK}.Translation: Translate fields"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_translate_process.sh

  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Translation: Load translated fields"
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_translate_load.sh
}

ClearVatFields()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Clean VAT fields."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
```

```
  TASKNAME="CleanVatFields"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_fields_clean.sh "vat"
}

ClearTelFields()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Clean Telephone fields."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="CleanTelFields"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_fields_clean.sh "tel"
}

ClearIdFields()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Clean Ids (Police Id, Passport) fields."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="CleanIdFields"

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_fields_clean.sh "id"
}

ClearEmailFields()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Clean Email fields."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="CleanEmailFields"

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_fields_clean.sh "email"
}

LoadResults()  {
```

```
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Load results to SQL Managed Server."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="LoadResults"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_fields_load_results.sh "synapse"     # "sql"
}

SqlPostProcessing()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Post processing tasks. Create and fill the output files."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="SqlPostProcessing"

  MESSAGE="${MESSAGE// /_}"
  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_postprocess.sh "sql"
}

Sql2Synapse()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Copy output tables from SQL to Synapse Server".
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="Sql2Synapse"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
  ${MHOME}/bin/run_ppc_delta_transfer.sh "sql2synapse"
}

SynapsePostProcessing()  {
  MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->${TASK}.
Post processing tasks. Create and fill the output files."
  echo ${MESSAGE} > ${MESSAGE_FILE}
  echo ${MESSAGE} >> ${PIPELINE_LOG}
  TASKNAME="SynapsePostProcessing"

  MESSAGE="${MESSAGE// /_}"

  ${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE
```

```
  ${MHOME}/bin/run_ppc_delta_postprocess.sh "synapse"
}

ExecTask()  {
  case $1 in
    1)
      SynapsePreProcessing
      ;;
    2)
      Synapse2Sql
      ;;
    3)
      SqlPreProcessing
      ;;
    4)
      ProcessAddresses
      ;;
    5)
      ProcessCustomerNames
      ;;
    6)
      TranslateFields
      ;;
    7)
      ClearTelFields
      ;;
    8)
      ClearIdFields
      ;;
    9)
      ClearEmailFields
      ;;
    10)
      ClearVatFields
      ;;
    11)
      LoadResults
      ;;
    12)
      SqlPostProcessing
      ;;
    13)
      Sql2Synapse
      ;;
    14)
      SynapsePostProcessing
      ;;
```

```
    *)
      echo "**** PIPELINE: delta, ERROR TASK NUMBER ****"
      ;;
  esac
}

trap ErrorHandler ERR

TASKNAME="DeltaPipelineProcess"

if [ "$TASK" -eq $FIRST ] ;
then
export MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK=0.
Start a new delta processing cycle."
else
export MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK=0.
Continue previously interrupted delta processing cycle."
fi

echo ${MESSAGE} > ${MESSAGE_FILE}
echo '' >> ${PIPELINE_LOG}
echo ${MESSAGE} >> ${PIPELINE_LOG}

MESSAGE="${MESSAGE// /_}"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE

while [ $TASK -le $LAST ]
do
  echo $TASK > $POSITION_FILE

  ExecTask $TASK
  TASK=$((TASK + 1))
done

TASKNAME="DeltaPipelineProcess"
MESSAGE="PIPELINE: delta, TIME: $( date '+%F_%H:%M:%S' ), TASK->0. End of
processing cycle."

echo ${MESSAGE} > ${MESSAGE_FILE}
echo '' >> ${PIPELINE_LOG}
echo ${MESSAGE} >> ${PIPELINE_LOG}

MESSAGE="${MESSAGE// /_}"
${MHOME}/bin/run_ppc_delta_dblog.sh $TASKNAME $MESSAGE


echo 1 > $POSITION_FILE
```