VERSION1.0



February 4, 2019

Prepared by: cognizant offshore

Health Care Service Corporation

Pharmacy prime CET CCF & addendum - Enhancements

Contents

[1. Objective 2](#_Toc199470)

[2. Project Summary 3](#_Toc199471)

[3. Code Changes 3](#_Toc199472)

* Configuration changes………………………………………………………………………………………………………………4
* Scala code Changes………………………………………………………………………………….……………………………….5

## Objective

The objective of this document is to provide our high level understanding, of enhancements what was done while implementing cet conversion process to CCF.

## Project SummarY

Gold claims repository is the centralized location in the data lake to have all the claim categories stored in consistent common claims format.

In current System Data lake receives a Pharmacy CET Claim file daily in raw format. As part of this initiative Pharmacy claims in prime CET file are to be parsed and converted to common claim format and converted claims are to be added to Kafka topic for Claim microservices API to consume and stored in gold claims repository.

Did some enhancements for the existing Claims API code. Those enhancements were list below.

## CODE CHANGES

1. Configuration Changes

**Configuration file based architecture for transformation rules**

***Existing implementation***:

In the existing implementation the transformations were written in code with repeated code with hardcoded transformation values, this is not flexible.

***Enhancement Description***: We have opted for configuration based approach for transformation rules because it’s loosely coupled with code, **trans\_coln** configuration key will be used to configure transformation rules. All the column wise transformations will combined and formed as a single value as shown below, this configuration then be decoded into column wise transformation map data structure with source to target mapping.

trans\_coln = JSONColumnName.{Source value : Target value};….

***Case 1: Source to Target direct mapping***

In below code we are preparing a 2 level map structure like below. This map will be used to match and pick the transformation map for any JSON column , then populate the target value based on source value.

Map ( JSONColumnName -> Map ( Souce value -> Target value ) )

//Parse and create transformation map from configuration file

**def** setTransMap(trans\_config:String) = {trans\_map = trans\_config.split(",").map{ key => Map(key.substring(0,key.lastIndexOf(".")) -> key.substring(key.indexOf("{")+1,key.indexOf("}")).split(";")

.map{col\_val => Map(col\_val.split(":")(0) -> col\_val.split(":")(1))}.reduce(\_++\_))

}.reduce(\_++\_)}

Transformation value fetching code snippet

/\*Check any transformation needs to be applied and populate the same .Incase default output mapping defined , then assign the same , otherwise pass the source value to target column. BLANK\_VAL - Handle transformation for blank input value(""). If the value updated as 'EXCEPTION', then exception will be thrown and particular JSON key value will be dropped DEFAULT\_VAL - Handles transformation of columns that needs to be defaulted in case of any value other than defined in transformation rules . Also handles exception if the value is updated as 'EXCEPTION'\* \*/

transform\_data = val\_trans\_map.getOrElse(

**if** (col\_data equals "") { throwExceptionForBadData(col\_data, "BLANK\_VAL", val\_trans\_map) } **else** { col\_data },

throwExceptionForBadData(col\_data, "DEFAULT\_VAL", val\_trans\_map))

**if** ( transform\_data == "NULL" )

{

transform\_data = **null**

}

***Case 2: Blank value transformation***

This architecture supports blank value at source can be transformed to any default value mentioned based on transformation rule like below.

trans\_coln = JSONColumnName.{BLANK\_VAL : Target value};….

***Case 3: Non specified domain value transformation with default value***

This architecture supports non specified domain value at source can be transformed to any default value mentioned based on transformation rule like below.

trans\_coln = JSONColumnName.{DEFAULT\_VAL : Target value};….

***Case 4: Exception in case of Blank/Non specified domain value at source***

This architecture supports to throw an exception and record the same in logs for non-specified domain value or for blank value at source as shown below.

trans\_coln = JSONColumnName.{DEFAULT\_VAL : EXCEPTION};….

trans\_coln = JSONColumnName.{BLANK\_VAL : EXCEPTION};….

Code snippet for case 2, 3 and 4 shown as below

@throws (classOf[Exception])

**def** throwExceptionForBadData(fieldData: String,check\_type :String,val\_trans\_map:Map[String,String]) : String ={

// Fetch any mapping defined for blank or default value , in case of any exception , throw the same.

**val** trans\_val = val\_trans\_map.getOrElse(check\_type,fieldData)

**if** (trans\_val.equalsIgnoreCase("EXCEPTION")) {

**throw** **new** Exception("No Match Found :" + fieldData)

} **else** **if**(check\_type equalsIgnoreCase("BLANK\_VAL")) {

check\_type

} **else** {

trans\_val

}

1. Scala Code Changes
2. **Many to one mapping resolution based on precedence order**

***Existing implementation***:

In the existing implementation many to one mapping was not handled, so the last transformation in the order will be applied.

***Enhancement Description***: We have handled many to one scenario based on the below mentioned rules.

1) Transformation with lower precedence order will be given priority.

2) In case of null value at source for the lower precedence transformation, next higher precedence transformation will be used.

The below code snippet is about a method , that removes many to one mappings and gives one to one transformation map based on precedence order.

/\*\*

\*

\* Sort out Many to one mappings in config-Map based on precedence order

\* Lower precedence map value will be taken if the record value is not null at that position,if Lower precedence position record value is null ,take higher precedence map value

\*/

**def** removeManyToOneMappings(configMap :HashMap[String, MCEFSchemaRecordTemplate], record:String) : HashMap[String, MCEFSchemaRecordTemplate] ={

**var** finalMap = HashMap[String, MCEFSchemaRecordTemplate]()

configMap.map{**case**(key,value) => **val** finalKey = key.toString().substring(0,key.toString().lastIndexOf("-"));

println("finalkey : "+finalKey)

**val** currentConfig = value

**val** currentVal = record.substring(currentConfig.MCEFStartPosition-1, currentConfig.MCEFEndPosition).trim()

//If the key is repeated .. check precedence order with real record data

**if**(finalMap.contains(finalKey)){

**val** finalConfig = finalMap.get(finalKey).get

**val** finalVal = record.substring(finalConfig.MCEFStartPosition-1, finalConfig.MCEFEndPosition).trim()

**var** lowerPrecedenceRecordData:String = **null**

**var** higherPrecedenceRecordData:String = **null**

**var** lowerPrecedenceMapVal : MCEFSchemaRecordTemplate = **null**

**var** higherPrecedenceMapVal : MCEFSchemaRecordTemplate = **null**

println ("existing val :"+finalVal+" current val :"+currentVal)

//If existing map value has lower precedence

**if**(finalConfig.precedenceOrder < currentConfig.precedenceOrder){

lowerPrecedenceRecordData = finalVal

lowerPrecedenceMapVal = finalConfig

higherPrecedenceRecordData = currentVal

higherPrecedenceMapVal = currentConfig

}**else**{

lowerPrecedenceRecordData = currentVal

lowerPrecedenceMapVal = currentConfig

higherPrecedenceRecordData = finalVal

higherPrecedenceMapVal = finalConfig

}

// Map to higher precedence incase of date column value of 0

**if**(!lowerPrecedenceRecordData.equals("") || (finalConfig.targetDatatype == "string date" && !lowerPrecedenceRecordData.equals("0"))){

finalMap.+=((finalKey,lowerPrecedenceMapVal))

}**else**{

finalMap.+=((finalKey,higherPrecedenceMapVal))

}

}**else**{ // For first occurence of key , place the key value pair in the final map

finalMap.+=((finalKey,currentConfig))

}

}

finalMap.foreach(println)

**return** finalMap;

}

1. **Multiple decimal format handling**

***Existing implementation***:

In the existing implementation all the decimal number will be reformatted to target decimal scale.

***Enhancement Description***: We have handled the scenario where for few claims sources ( CET , Healthmine etc..) , decimal values need not be reformatted to target scale but are needed to be pass-through as it is based on the configuration in config file.

Configuration in config file is as below

demical\_format = "passthrough"

Code snippet below.

**else** **if** (keyVal.\_2.targetDatatype == "number") {

**val** Recordvalue = dataRecord.substring(keyVal.\_2.MCEFStartPosition - 1, endPosition).trim()

// println(Recordvalue)

**if** (decimal\_format.trim().equalsIgnoreCase("passthrough")) {

prevMap = translateCCFAndBuildMapping(keyVal.\_2.CCFFieldName, Recordvalue, prevMap)

**val** JSONstring = (kafkaKey, Json(DefaultFormats).write(prevMap)).toString()

record = Record(Json(DefaultFormats).write(prevMap), **null**, kafkaKey)

}

**else**

{

1. **Handling signed numbers**

***Existing implementation***:

In the existing implementation sign information will be provided at the last position of the number as an encoded symbol, which will be decoded based on sign map to provide sign.

For few sources (CET and Healthmine ) , number fields on the source data will have explicit '+' and '-' signs at start position , these were not handled .

**Enhancement Description:** As per requirement, We have handled signed numbers based on below rules

For sources (CET and Healthmine) where number fields on the source data will have explicit '+' and '-' signs at start position :  
If a particular field has a '+' on the source data, we drop the sign and didn't include it in target.  
If a particular field as a '-' on the source data, include the sign also in target value.  
  
Code Snippet:

**val** Recordvalue = dataRecord.substring(keyVal.\_2.MCEFStartPosition - 1, endPosition).trim().replace("+","")

1. **Removal of special symbols from default value in mapping sheet**

***Existing implementation***:

In the existing implementation the special symbols from default value removal is not handled***.***

***Enhancement Description***:

In case if no source column mapping with target in mapping sheet, it should take TARGET\_DEFAULT\_VALUE from mapping sheet to get populate in JSON output. We have handled scenario where the target value contains some special characters, these special characters should be removed.

Code Snippet:

defaultValue.trim.replace("\"", ""),

1. **Date conversion handling**

***Existing implementation***:

In the exiting implementation the source date conversion is not been handled to cover 7 digit date format (cyyMMdd). Only the following type of source format (“yyyyMMdd","yyyyMMddhhmmss") has been handled .

***Enhancement Description***:

As per our latest code whatever the source date format (“cyyMMdd" ,“yyyyMMdd","yyyyMMddhhmmss"),based on length of the datait will convert into desired target date/timestamp format.

Code snippet:

**def** stringToDateforSourceData(recVal:String):Date={

recVal.length() **match** {

**case** 7 => sourceDt.parse("C".concat(recVal).replace("C0","19").replace("C1","20"))//Added C to match and replace first char

**case** 8 => sourceDt.parse(recVal)

**case** \_ => sourceTmstmp.parse(recVal)

}

}

***Case 1: 7 byte Date***

First we need to take the length of the source date in case if it’s of 7 byte date (“cyyMMdd") we will replace the 0 with 19, 1 with 20 , then we parse it with (“yyyyMMdd") format to get proper date.

***Case 2: 8 byte Date***

After getting the length of the date in case if it’s of 8 byte, then we parse it with (“yyyyMMdd") format to get proper date.

***Case 3: Any other 8 byte Date***

In case of date byte other than 7 or 8 we will consider that source format as (“yyyyMMddhhmmss”) and will convert that into ("yyyy-MM-dd hh:mm:ss") as a target date.

1. **Missing Configuration Handling to Support Running of Older Config Versions**

***Existing implementation***:

In the existing implementation, if any configuration in config file is missing, the code will not execute.

***Enhancement Description***:

We have handled the code to support older configuration files , in case of any config key value pair is not available , we will place a dummy value in place of the missing configuration , to make the code run without any issue.

The old config file did have below configuration.

“trans\_coln “,”decimal\_format“

This is handled in code as below

**val** Transform\_Rules = Util.getConfigValueOrDefault(config,"claim-api.trans\_coln","NA.{NA:NA}")

**val** Decimal\_Format = Util.getConfigValueOrDefault(config,"claim-api.demical\_format","NA")

**def** getConfigValueOrDefault(config:Config,path:String,default:String):String={

**if** (config.hasPath(path)) {

config.getString(path)

}**else** {

default

}

}

The above code will give the transformation output if the source system has any transformation rule, if the source system doesn’t have any transformation rule it give default value means that the code will run successfully irrespective of using old and new configuration file.