Oracle Banking Digital Experience

File Upload Flow and Template Definition Guide
Release 18.2.0.0.0

Part No. E97823-01

June 2018

ORACLE®

File Upload Flow and Template Definition Guide

June 2018

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001

www.oracle.com/financialservices/

Table of Contents

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

Preface gives information on the intended audience. It also describes the overall structure of the User Manual.

Introduction provides brief information on the overall functionality covered in the User Manual.

The subsequent chapters provide information on transactions covered in the User Manual.

Each transaction is explained in the following manner:

- Introduction to the transaction
- Screenshots of the transaction
- The images of screens used in this user manual are for illustrative purpose only, to provide improved understanding of the functionality; actual screens that appear in the application may vary based on selected browser, theme, and mobile devices.
- Procedure containing steps to complete the transaction- The mandatory and conditional fields of the transaction are explained in the procedure.

If a transaction contains multiple procedures, each procedure is explained. If some functionality is present in many transactions, this functionality is explained separately.

1.5 Related Information Sources

For more information on Oracle Banking Digital Experience Release 18.2.0.0.0, refer to the following documents:

- Oracle Banking Digital Experience Licensing Guide
- Oracle Banking Digital Experience Installation Manuals

2. File Upload Template Definition

A file upload template is an XML file, which defines the schema of the file, which one can upload in OBDX File Uploads.

Accounting Types:

- **Single Debit Multiple Credits (SDMC)**: An actual SDMC File with data contains a Header component and a Body component. The SDMC template file contains only the definitions of the Header and the Body components.
  The Header definition specifies the fields required to define the debit account. The Body definition specifies the fields required to define each credit account.
- **Single Debit Single Credit (SDSC)**: In the SDSC template, we define fields required to represent N distinct transactions. There is no header and therefore each record must be of the type body (B).
- **Multiple Debit Multiple Credit (MDMC)**: In the MDMC template, we define fields required to represent N distinct transactions. There is no header and therefore each record must be of the type body (B).

The structure of each template consists of the following XML nodes:

- FileDefinition – Parent Node
- RecordDefinition – Child Node of FileDefinition
- Field – Child Node of RecordDefinition

FileDefinition:

A FileDefinition node can be explained with the help of the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FileDefinition fileName="DomesticFTSDMC"
    fileHandlerClassName="com.ofss.digx.app.fileupload.handlers.DomesticFTDetailsFileHandler"
    decryptionClass="" charSet="UTF-8" delimiter="," comments=""
    isFirstRecHeader="true" simpleOrMixed="M" fillchar="" partialProcessing="100" transactionType="DTF" skipRecsCnt="0">
```

- **fileHandlerClassName**: The value of this variable represents the fully qualified class name of the file handler. This class provides the methods for file processing. The specified class includes:

A Method to perform file level validations for a given File ID, a method to process the file for a given File ID etc.

The following image shows a sample custom File Handler code. The File Handler needs to mandatorily extend

com.ofss.digx.app.fileupload.handlers.AbstractFileHandler

```java
package com.ofss.digx.app.fileupload.handlers;

import java.util.List;

import com.ofss.digx.app.dto.fileuploads.AbstractRecordDTO;
import com.ofss.digx.enumeration.fileupload.ApprovalType;
import com.ofss.fc.app.context.SessionContext;
import com.ofss.fc.infra.validation.error.ValidationError;
import com.ofss.fc.service.response.TransactionStatus;
import com.ofss.digx.app.fileupload.handlers.AbstractFileHandler;

public class MyCustomFileHandler extends AbstractFileHandler{

    @Override
    public List<ValidationError> isValid(String fileRefId) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public TransactionStatus processApprovals(SessionContext sessionContext, AbstractRecordDTO abstractRecordDTO)
            throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public ValidationError[] process(String fileRefId, String recRefId, String templateId, ApprovalType approvalType) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void checkAccess(SessionContext sessionContext) throws com.ofss.digx.infra.exceptions.Exception {
        // TODO Auto-generated method stub

    }

    @Override
    public void checkAccountAccess(AbstractRecordDTO abstractRecordDTO)
            throws com.ofss.digx.infra.exceptions.Exception {
        // TODO Auto-generated method stub

    }

    @Override
    public void checkInitiationLimitForFile(AbstractRecordDTO abstractRecordDTO)
            throws com.ofss.digx.infra.exceptions.Exception {
        // TODO Auto-generated method stub

    }


}
```

The File Handler is the heart of the File Upload Engine and at this point of time, we do not expect anyone to completely customize the File Upload Engine itself. Therefore, we are not going to dive into the details of each method.

- **decryptionClass**: Specifies a fully qualified class name which performs encryption and decryption of the file to be uploaded. One can write their own class for encryption and decryption and this class should implement com.ofss.digx.framework.fileupload.encryption.IEncryptorDecryptor interface. The secret key can be stored in database or in Weblogic's key store. The location of secret key is specified in the table DIGX_FW_CONFIG_ALL_B with prop_id “ENCRYPTION_KEY_LOCATION”

- **Delimiter**: Specifies the character that should be used to separate the fields in a record. The value of this field is empty in the case of a fixed length file.

  Example of a record in a variable length file:

  A,001199,AT30011990016,10.49,GBP,02-01-2014,Donald,68088DRYJU,SWI,BARCGB22XXX,PAY1,PAY2,PAY3,PAY4,PAYER,DrNarrative1,CrNarrative1

- **isFirstRecHeader**: This field specifies whether the first record is of type Header or not. This would be true only in the case of SDMC, since the first record in SDMC is debit account record and its type is Header.
- **simpleOrMixed**: The two possible values of this attribute are "M" and "S". "M" indicates that the file contains records of different transaction types (adhoc and beneficiary). If the file has records of only 1 type of transaction (adhoc/beneficiary), then the value of this attribute will be "S".
- **fillChar and fillcharPos**: fillChar field specifies the character that is used for padding when the length of a field value is less than the length specified in the length attribute of a field. Fillchar is empty in the case of variable length template. FillcharPos specifies whether the padding should be done from left or right.
- **partialProcessing**: Represents the percentage of records that should pass the pre-processing, so that the status of file uploading will be "verified".

RecordDefinition:

All the below attributes represents the definition of a record.

A RecordDefinition node can be explained with the help of the following example :

```
<RecordDefinition
  recordHandlerClassName="com.ofss.digx.app.fileupload.handlers.PaymentHeaderHandler"
  recordType="H"
  dtoClassName="com.ofss.digx.domain.fileupload.entity.PaymentsHeaderDTO"
  multiplicity="1" maxFields="" comments=""
  parent="" length="59" transaction=""
  mixedIdentifier="">
```

- **recordHandlerClassName**: The value of this variable represents the fully qualified class name of the record handler and this class provides the methods for record processing. The specified class includes:
  - Method to perform record level validations for a given record
  - Method to execute business policies
  - Method to process the record etc.,

The Record Handler needs to mandatorily extend com.ofss.digx.app.fileupload.handlers.AbstractRecordHandler.

- **recordType**: Specifies type of the Record. i.e., "B" represents Body and "H" represents Header. In the case of SDMC (Single Debit Multiple Credit), the record specified in the header represents the debit account details while the records specified in the body represents the details of multiple credit accounts. In the case of MDMC and SDSC recordType should be "B".

- **dtoClassName**: Each record in a file can be represented in Java as a DTO class. Such a DTO class would typically contain all data fields in the record as its attributes and the corresponding getter and setter methods. The dtoClassName represents the fully qualified class name of such a DTO.
- **Multiplicity**: This field defines the number of times a record can be repeated in a file.

  -1 represents infinite times. In the case of SDMC, since there will be only one debit account, the multiplicity of the header record would be 1.
- **maxFields**: The maximum number of fields allowed for a particular record.
- **Length**: This field represents the total length of the record, in case of Fixed Length Files. This value is empty for variable length file types.

  Ex: If you specify length as 100, then the first 100 characters are considered as one record.
- **Transaction**: This field represents the type of transaction (Internal, international etc.). The value of this field have an entry in the table DIGX_FW_CONFIG_ALL_B with prop_value as fully qualified class name of the class which provides the service for checking the role access, for sending file/record details to approval engine and for checking the initiation limits.
- **mixedIdentifier**: This field identifies the type of transfer for domestic payments. “A” represents Adhoc transfer and “B” represents Beneficiary.

Field:

All the below attributes represent the definition of a field.

A Field represents various attributes that are required to perform a transaction. It contains:

- Name : Name of the field.
- Length : Length of the field.

For Example:

```
<Field name="partyId" length="10"/>
```

So, the first 10 characters are considered as partyId. Length field is required for a fixed length template in order to parse the file and retrieve the values of various attributes. For a variable length record, this field should be empty.

- **Enricher**: Enrichers are used to enrich or fetch a value for a given field. Let us say the field is Debit Account Id and enricher is Account Currency, so it means that the currency for that debit account Id needs to be fetched or enriched. The value specified in this field has corresponding ENRICHMENT_VALUE which is the fully qualified class name of the implemented enricher in the table DIGX_FW_ENRICHMENTS_B. Currently OBDX supports only Java enrichers. Enrichers can be in any package but must implement the ‘IEnrichment’ interface.

Enrichers are of two types:

Static Enrichers:

Example 1:

```
<Field name="valueDate" enricher="DATE" enricherArgs="dd-MM-yyyy" length="15"/>
```

The field name ‘valueDate’ has enricherArgs ‘dd-MM-yyyy’ meaning that the date has to be specifically in ‘dd-MM-yyyy’ format. This value is simply available to the enricher for processing purpose. This enricher does not add any new field but simply modifies the value of the current field.

Example 2:

```
<Field name="debitAccountId" enricher="ACCTCURR" enricherArgs="" length="20"/>
```

The field name ‘debitAccountId’ has an enricher ‘ACCTCURR’ with no enricherArgs. In this case, ‘DIGX_FW_ENRICHMENTS_B’ will be queried and search for ‘ACCTCURR’ and ‘AccountCurrencyEnricher’ class is invoked. This enricher derives the debitAccountCurr. Hence, this attribute must be present in the record DTO with its setters defined.

Dynamic Enricher:

```
<Field name="beneId"/>
<Field name="beneName"/>
<Field name="beneAddr" enricher="ADDRESSENRICHER" enricherDynArgs="beneId~beneName"/>
```

For Example: enricherDynArgs=”beneId~beneName” on beneficiary address field, the parser simply invokes getters on **beneId** and **beneName** fields and passes the values to the enricher in a map. It should be noted that these fields must be defined previously/above the beneficiary address field, so that parser has already completed the setter operation.

- **Fillchar**: This field specifies the character that is used for padding when the length of a field value is less than the length specified in the length attribute of a field. Fillchar is empty in the case of variable length template. FillcharPos specifies whether the padding should be done from left or right. If fillchar or fillcharPos is not defined for a particular field at record level in the case of fixed length template, then the file level values of fillchar and fillcharPos are applied.

```
<Field name="totalAmount" type="D" length="10" fillchar="0" fillcharPos="L"/>
```

3. File Upload Flow

3.1 How does the File Upload work

The first step is the parsing and the validation of all the records in the file that needs to be uploaded.

The validation success depends on the approval level. The approval level is defined by the admin at the time File Identifier Mapping. The two types of Approval Level are:

- **File Level**: Only if all the records of the file are marked as verified, the status of the file will change to VERIFIED.
- **Record Level:** If a certain number of records pass the validation, then the File status will be updated as Verified even though certain records would have failed the validation. That certain number is a value that is defined during the File identifier mapping by the admin user and also in the file template as an attribute named *partialProcessing***.** If both the values differ, then higher priority is given to the value in File Identifier Mapping.

Approval Level support for different accounting types:

- SDMC → File Level (Record Level Approval not supported)
- MDMC → Both File and Record Level
- SDSC → Both File and Record level

Only after all the validations are successful, all the records are stored in the table

DIGX_FU_RECORD_MASTER and the data about the file is stored in the table **DIGX_FU_FILEDETAILS** as a single record**.**
Now, this data needs to be passed to the UBS schema for core banking processing.

However, before that, we populate this data in two tables referred as a mirror tables in the B1A1 schema. These tables are named **DIGX_FU_FT_DETAILS** and **DIGX_FU_FT_MASTER**. They are exact replicas of the two tables **MSTM_COM_PMT_IN** and **MSTM_COM_PMT_INFILE** respectively in the UBS schema where we are going to dump the file data finally.

After the data is populated in the tables DIGX_FU_FT_DETAILS and DIGX_FU_FT_MASTER of B1A1 schema, a Database Job performs a sanity check before putting the data into the UBS schema.

The name of the DB job is *DIGX_CPG_FILEUPLOAD*

The data is dumped from DIGX to UBS schema in the following sequence.

DIGX_FU_FILEDETAILS (**DIGX**) → DIGX_FU_FT_MASTER (**B1A1**) → MSTM_COM_PMT_INFILE (**UBS**).

DIGX_FU_RECORD_MASTER (**DIGX**) → DIGX_FU_FT_DETAILS (**B1A1**) → MSTM_COM_PMT_IN (**UBS**).

There is a column in the table **DIGX_FU_RECORD_MASTER** named **REC_STATUS**, which indicates the status of the records of the File.

Different Types of Record Status for the table DIGX_FU_RECORD_MASTER (The various procedures/stages of File Upload)

- **UPLOADED**: After the user uploads the file the status of the records are in the uploaded state.

- **VERIFIED**: After all the validations performed by OBDX are successful the status of the records changes to Verified.
- **APPROVED/REJECTED**: Consider two scenarios, the first one is where a maker does the file upload and needs approval from the checker for further process. Therefore, until the checker approves the transaction the status remains in verified state. Only after the checker approves, the status will change from Verified to Approved. However if the file upload is done by an AutoAuth user , the status will immediately change from verified state to approved state as no approval is required in this case. The status will be modified to Rejected if the checker rejects the transaction.
- **PROCESSING IN PROGRESS**: Once all the contracts are successfully booked from the UBS, a corresponding CONTRACT_REF_NO is generated which is updated in the table **MSTM_COM_PMT_IN** in the UBS schema and the status changes to 'P' (Processed) **MSTM_COM_PMT_INFILE** and PROCESSING_IN_PROGRESS in DIGX schema.
- **COMPLETED**: Once the contracts are liquidated by the UBS (Funds transfer are completed by the UBS).
- **ERROR**: If the processing of a record fails due to some reason, then the UBS updates the status of the record as error. The status will change to Error if there is problem in any step/procedure. For example : If there is some error while contracts are booked by the UBS , the status changes to Error, or the records fail the validation it again enters the ERROR status.

Different Types of File Status for the table DIGX_FU_FILEDETAILS.

- **UPLOADED**: After the user uploads the file, the status of the records are in the uploaded state.
- **VERIFIED**: After all the validations performed by OBDX are successful the status of the file changes to Verified.
- **APPROVED/REJECTED**: Consider two scenarios, the first one is where a maker does the file upload and needs approval from the checker for further process. Therefore, until the checker approves the transaction the status remains in verified state. Only after the checker approves, the status will change from Verified to Approved. However if the file upload is done by an AutoAuth user , the status will immediately change from verified state to approved state as no approval is required in this case. The status will be modified to Rejected if the checker rejects the transaction.
- **PROCESSING IN PROGRESS**: Once all the contracts are successfully booked from the UBS, the status changes to 'P' (Processed) in **MSTM_COM_PMT_INFILE** and PROCESSING_IN_PROGRESS in **DIGX_FU_FILEDETAILS**.
- **PROCESSEXCP:** Processed with Exceptions. If even a single record are updated with status ERROR in the **DIGX_FU_RECORD_MASTER** then the status of the File is updated as PROCESSEXCP in the table **DIGX_FU_FILEDETAILS**.
- **PROCESSED:** Once all the records of a file enters COMPLETED state, the status of the file is modified to PROCESSED.
- **ERROR**: If the processing of all the record fails due to some reason, then the UBS updates the status of the File as error.

Different types of file status in the table MSTM_COM_PMT_INFILE.

- **W (Work_In_Progress):** After the data is populated in the UBS table, the status is marked as 'W'.
- **U (Uploaded):** UBS performs certain jobs and updates the status as 'U' after the completion of the job.
- **P (Processed):** After the contracts are booked by the UBS successfully, the status is updated as 'P'.
- **R (Rejected):** If there is some error while the contracts are booked by the UBS, the status is updated as 'R'.

Few Examples/Scenarios and expected Behavior.

Now what if there are multiple records of a file and half the records have one status and other half of records have other status, what will be the status of the File in **DIGX_FU_FILEDETAILS**?

The sequence of the statuses is

- UPLOADED
- VERIFIED
- APPROVED/REJECTED
- PROCESSING_IN_PROGRESS
- COMPLETED

So Consider that half the records of a file are in the Verified status and other half are Approved. In such scenarios the status of the File will always be of the lower scale of the two statuses, i.e. the status of the file will be verified. Only after all the records attain the Approved status, the File status will change to Approved.

UBS updates the status of the File Upload after every stage of host processing, in the tables MSTM_COM_PMT_INFILE and MSTM_COM_PMT_IN.

There is an EJB timer, which synchronizes the status of the table DIGX_FU_RECORD_MASTER with MSTM_COM_PMT_IN and DIGX_FU_FILEDETAILS with MSTM_COM_PMT_INFILE.

Now consider a future dated transaction. Here the status in the table MSTM_COM_PMT_INFILE is 'U' (Uploaded) after the UBS jobs are completed successfully.

At this stage, the status of the file in the DIGX table **DIGX_FU_FILEDETAILS** will be seen in the **Approved** state. The status will change to **Processing in Progress** in DIGX_FU_FILEDETAILS only after all the contracts are booked by the UBS and the status is updated to **P (Processed)** in the table **MSTM_COM_PMT_INFILE** in the UBS schema. This is because the cancellation functionality is only available when the status is processing in progress. That implies that if the status changes to Processing in Progress before the contracts are booked by the UBS, the cancellation functionality will be possible which is not logically correct. Hence, after the transaction is approved, it waits for the contracts to be booked by the UBS before updating the status as Processing in Progress.