Generic Payments Design

Oracle FLEXCUBE Direct Banking

Release 12.0.3.0.0

Part No. E52543-01

April 2014

ORACLE®

Generic Payments Design

April 2014

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/

CONTENTS

1.Preface

1.1. Intended Audience

This document is primarily targeted at

- Oracle FLEXCUBE Direct Banking Development Teams
- Oracle FLEXCUBE Direct Banking Implementation Teams
- Oracle FLEXCUBE Direct Banking Implementation 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 OFSS Support

https://support.us.oracle.com

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

Generic Payments Design provides information on Developing a new Payments Transaction , Validations using txn_data_master and txn_data , Configuring Deal for Payments Transaction etc.

Chapter post Generic Payments Design is dedicated to Add/Modify a Beneficiary Type

1.5. Related Information Sources

For more information on Oracle FLEXCUBE Direct Banking Release 12.0.3.0.0, refer to the following documents:

- Oracle FLEXCUBE Direct Banking Licensing Guide

2.Abbreviations

FCDB / FC DB / FC Direct Banking / Direct Banking	Oracle FLEXCUBE Direct Banking
DBA	Database Administrator
URL	Uniform Resource Locator
JNDI	Java Naming and Directory Interface
DB	Database
IP	Internet Protocol
FCAT BASE DIR	FCDB Base Working Directory

3.Generic Payments Design

After designing the screen for payments using Channel Workbench Tool one can define the payment module using the generic payment service using the following steps.

3.1. DEVELOPING A NEW PAYMENTS TRANSACTION

To start a payment Transaction a developer needs to make an entry in following tables

- Msttxn
- MstUserTypeTxn
- Appldata for dataname TXN_DESC for the corresponding idtxn.
- MstChannelATS for the idtxn for each screen number.
- MstServices for the services

Map the Transaction to the Role to allow it to appear in the Menu of the Customer Login.

For each prepare screen a generic service XSL has been used. For e.g. for RRPPT01 service xsl is genericpaymentprepare.xsl.

```
<ServiceRequest>
        <serviceName>GenericPaymentServiceInterface.GenericPaymentService.prepare
                GenericPaymentDetails
        </serviceName>
<request>
        <NullRequestDTO>
        </NullRequestDTO>
</request>
</ServiceRequest>
```

The service name used in this xsl is GenericPaymentService and the method is prepareGenericPaymentDetails. To hook this method of GenericPaymentService to the service of the desired transaction the developer needs to make an entry in following table

- GenericPaymentServiceMapper.

The developer needs to give the full qualified class name of the service. An additional column of requestdtomapper has been provided which can be used to provide a mapper class which can map the GenericPaymentRequestDTO to the desired RequestDTO. The new mapper class must extend GenericPaymentsDataMapper.java. If no mapper class name is given the system will assume that the service is to be invoked using GenericPaymentRequestDTO.

For each initiation screen a generic service xsl has been used. For e.g. for RRPPT02 service xsl is genericpaymentinit.xsl

```
<ServiceRequest>
        <serviceName>
                GenericPaymentServiceInterface.GenericPaymentService.initiate
                GenericPaymentDetails
        </serviceName>
        <serviceAction>
```

```
                <xsl:value-of select="/faml/request/fldnextaction"/>
        </serviceAction>
        <referenceNo>
                <xsl:value-of select="/faml/request/fldreferenceno"/>
        </referenceNo>
        <stateBit>
                <xsl:value-of select="/faml/request/fldstatebit"/>
        </stateBit>
<request>
<GenericPaymentRequestDTO>
        <srcAccount>
                <AccountNoInputDTO>
                </AccountNoInputDTO>
        </srcAccount>
        <destAccount>
                <AccountNoInputDTO>
                </AccountNoInputDTO>
        </destAccount>
        <txnAmount>
                <xsl:value-of select="//faml/request/fldtxnamount"/>
        </txnAmount>
        <txnCurrency>
                <xsl:value-of select="//faml/request/fldtxncurrency"/>
        </txnCurrency>
        <valueDate>
                <xsl:value-of select="/faml/request/fldvaluedate"/>
        </valueDate>

<xsl:if test="/faml/request/fldudf">>
        <udfFields>
                <xsl:for-each select="/faml/request/fldudf">
                        <UDFDTO>
                                <xsl:variable name="udfname">
                                        <xsl:value-of select="concat ('/faml/request/', .)"/>
                                </xsl:variable>
                                <udfName>
                                        <xsl:value-of select="."/>
                                </udfName>
                                <udfValue>
                                        <xsl:value-of select="dyn:evaluate($udfname)"/>
                                </udfValue>
                        </UDFDTO>
                </xsl:for-each>
        </udfFields>
</xsl:if>
</GenericPaymentRequestDTO>
</request>
</ServiceRequest>
```

The xml tags declared in this XSL are the basic parameters that are required for a payment transaction. Apart from these fields, what else new field has to be handled on the screen are handled as udfdata.

Now for each screen the output XSL used is “genericscreentemplate.xsl”. This XSL is the master xsl for building the entire screen using the screen definition developed through Channel Workbench. The developer needs to define these fields in screen design xml so that the respective values are passed to the service. The field for which value of “ISUDF” in screen design xml is ‘Y’ are automatically taken care and passed to the service as udfdata.

3.2.VALIDATIONS USING TXN_DATA_MASTER AND TXN_DATA

All the transactions that are developed through generic payment service should have flag value “Y” for the column “isgenerictemplate” in the table MstChannelATS. Once the value of this column is set as Y, all the entries that are to be done in txn_data_master and txn_data should have the requestId or IdTxn appended before the validation field separated by a “.” as explained below.

RRPPT02.GENERICPAYMENTSERVICEINTERFACE.GENERICPAYMENTSERVICE.INITIATEGENERICPAYMENTDETAILS.SRCACCOUNT

Or

PPT.GENERICPAYMENTSERVICEINTERFACE.GENERICPAYMENTSERVICE.INITIATEGENERICPAYMENTDETAILS.SRCACCOUNT

The concept of using the second format of validation entries is to prevent replication of entry, say the same validation entry is required for screen no 2 and 3, in that case a developer can do the entry by appending the idtxn before the validation field so they need not need to replicate the entry on the basis of idrequest.

3.3 CONFIGURING DEAL FOR PAYMENTS TRANSACTION

If cross currency involve during the payments, then such transaction require deal box or deal functionality.

Availability of deal functionality is depending on customer profile of the primary customer of the current logged in user.

If the value of 'isallowcombinspotfwd' column in customer profile is 'Y' then only the deal box will be appear in transaction else otherwise it will not.

Deal box functionality can be included in generic payments using three different type which are 'DT','DV' and 'DC'.

'DT' type should be included in prepare page. (select * from screentemplate where identity='B001' and idrequest ='RRPPT01' and type = 'DT')

'DV' type sholud be included in verify,confirm and auth-template page(select * from screentemplate where identity='B001' and idrequest in
('RRPPT02','RRPPT03','RRPPT89') and type = 'DV')

'DC' type should be included in confirm and auth-template page. (select * from screentemplate where identity='B001' and idrequest in('RRPPT03','RRPPT89') and
type = 'DC')

There are some functions present in commonftfunctions.xsl. These functions should be called from respective js file
Also respective js file should have some functions like ‘fnGo’,’fnIsDealBoxRequired’ etc
(Please refer RRPPT01.js, RRPPT02.js, RRPPT03.js, RRPPT89.js etc).

Once the types are included and js files are created then the rest of the functionality will be automatically handled by the framework.

3.4 CONFIGURING AUTH FOR PAYMENTS TRANSACTION

Creating 'save as Template' screens (RRXXX97)

It requires screen definition xml file as 'RRXXX97.xml' and js file as 'RRXXX97.js'
Confirm screen ('RRXXX03') and 'save as template confirm' screen (RRXXX97) are identical. So there is no need of creating extra screentemplate for 'RRXXX97'.
Developer can copy RRXXX03.xml file to 'RRXXX97.xml' using copy functionality in screentemplate xml generator tool.

Creating 'save as draft' screens (RRXXX98)

It requires screen definition xml file as 'RRXXX98.xml' and js file as 'RRXXX98.js'
Confirm screen ('RRXXX03') and 'save as draft confirm' screen (RRXXX98) are identical. So there is no need of creating extra screentemplate for 'RRXXX98'.
Developer can copy RRXXX03.xml file to 'RRXXX98.xml' using copy functionality in screentemplate xml generator tool.

Creating 'transactions - template' screens (RRXXX89)
Separate screen definition (RRXXX89) and respective js file (RRXXX89.js) require for 'transactions - template' screens (RRXXX89)

(Please refer 'RRPPT97.js', 'RRPPT98.js', and 'RRPPT89.js')

Creating 'Auth_prepare' screen (RRXXX99)

This screen will be same as Prepare screen (Example RRXXX01).
This screen will be called with id request as RRXXX99. Screen tables entries are not required for this screen. (**No need to generate RRXXX99.xml and no need to create RRXXX99.js**). Framework will use the same screen of prepare transaction (like RRXXX01.xml and RRXXX01.js)
Make sure that the screen sequence number of prepare screen is '01'. If is not, then developer need to specify target screen for 'RRXXX99 request'. This specification can be done inside 'RRXXX89.js' file. (** RRXXX89 is auth view screen in LEAP)
For Example – Consider the following cases.

Case 1 – Prepare screen's sequence number is '01' (RRXXX01)

```
    //no need to specify target explicitly.
```

Case 2 – Prepare screen's sequence number is other than '01' (like RRXXX03)

```
//developer need to call the function 'fnSetNextScreenRequestId ' from //RRXXX89.js as shown below.
    function initialize () {
        fnSetNextScreenRequestId ('RRXXX03');
    //rest of the code
}
```

Validation for 'save as template' (RRXXX97)

It does not require separate entry in txn_data_master. It follows the same validation which has been followed by verify and confirm transaction. In FCDB 6.0.0.0.0.0.0 entries for RRXXX97 has been added in day zero setup. One can configure the field validations as per the functional requirements of saving a transaction as template.

Validation for 'save as draft' (RRXXX98)
It requires separate entry in txn_data_master and txn_data.
(e.g select * from txn_data_master where idrequest like 'RRPPT98%' and select * from txn_data where idrequest like 'RRPPT98%')

Validation is not require for 'transaction - template' screen (RRXXX89)

3.5 CONFIGURING THE BENEFICIARY FOR PAYMENTS TRANSACTION

As per the current architecture a developer needs to follow a fixed set of screen sequence number for creating Beneficiary Transaction of the corresponding Payment module. Developer need to use screen no 26, 27, 28 for Beneficiary creation screen flow and 56, 57, 58 for Beneficiary modification screen flow. So, for configuring Beneficiary for UK Payments (idtxn PPT), entry required in mstchannelats would be RRPPT26, RRPPT27, RRPPT28, RRPPT56, RRPPT57, and RRPPT58. Screen sequence no 26, 27 and 28 are meant for initiating a new beneficiary creation, while 56, 57 and 58 are meant for modifying an existing beneficiary. To support authorization on beneficiary of each transaction one needs to define a new idtxn. For example, idtxn for beneficiary of UK Payments (idtxn PPT) is PPB.

4.Add/Modify a Beneficiary Type

Beneficiary data is stored in MSTTEMPLATEMASTER table in flat columns. The UDF fields used for beneficiary are also stored in flat columns. Following steps are to be followed in case of adding or modifying a beneficiary type.

4.1 ADDING A UDF FIELD

While adding any “UDF” field, following components should be modified-

1. MSTTEMPLATEMASTER:- Add a column name same as udf filed name by adding prefix “UDF_”. For example if newly added udf field is –“fldpetname” then column name must be “**UDF_**fldpetname”.
2. FCAT_VW_MSTBENEFICIARY:- Select newly added column in this view.

4.2 ADDING A NON-UDF FIELD

While adding any “non-UDF” field, following components should get modified-

1. MSTTEMPLATEMASTER:- Add appropriate column in this table
2. FCAT_VW_MSTBENEFICIARY:- Select newly added column in this view.
3. PROC_CREATE_BENE_TEMPLATE:- Modify the procedure appropriately to save/update/delete data.
4. genericbeneficiarytemplate.xsl:- Modify service xsl as per new addition of new field.
5. Extend the endpoint to fetch/insert/update the new beneficiary field.
6. DTO: - Create a new DTO with BeneficiaryDTO and the new field as per the requirement.