

Online Banking Solution

Customization Guide

11.2.10

15 April, 2018

[© 2018 EdgeVerve Systems Limited](#). Strictly private and confidential. No part of this document should be reproduced or distributed without the prior permission of EdgeVerve Systems Limited.

Table of Contents

Customization Guide.....	5
About this Help.....	5
Introduction	6
Presentation Level Customization	7
Journaling and Auditing.....	8
Overall Maintenance.....	9
Menu Maintenance	10
Submit Button Customization	12
Application Flow	13
Parameterization of Files and Tables.....	14
Session Time-out.....	15
BayAudit	17
Generic Email.....	18
Report Generation	19
Business Level Customization.....	20
Business Level Customization	20
Functional Customization	21
Modifying Business Logic.....	24
Error Code Range	25
OpConsole.....	26
Hold or Release Functionality	27
Multiple Funds Transfer Functionality	28
Drill Down Functionality.....	29

Command.properties File	30
Custom Hooks	31
User Menu Linkage	32
Payments	33
Bank Details.....	34
ChargeCalculator	35
InitialNetworkValidations.....	36
PaymentsLien	37
ProcessPaymentsCustom	38
RMApapprovalRequirementStatus	39
TransactionValidation	40
MultiCurrencyHook	41
Interest Computation.....	42
Imaging.....	43
File Upload Custom Hooks	44
Calculate Charge and Commission Amount	50
Mark Lien on an Account	51
Removal of Lien on an Account	52
Offline Processing of a Transaction	53
Role ID for an RM	54
Framework Level Customization	55
Framework Level Customization	55
Customization Infrastructure	56
Achieving Customization in Type System Layer	60
Customization in Formsgroup Layer	67

Customization in Service Formsgroup Layer	75
Customization in Service Layer	79
Customization in Transaction Layer	81
Customization in TAO/TMO Layer	83
Customization in DAL Layer	84
Customization in HIF Layer	89
Context	93
Error Code/Incidence Code Customization	94
Customization in UI Layer	96
Multi Branding.....	100
Customization of CorpAdmin Framework.....	106
Customization of Formmanagement Framework	113
Customization Toolkit	126
Entity Reference	130
Caching Framework	131
Filtration Framework.....	132
Common Event Registry Framework	133
List of Abbreviations	134

Customization Guide

About this Help



Welcome to the Finacle e-Banking Solution Customization Help. This document is meant for the IT team in the bank or Infosys customer support engineers to customize e-Banking to suit the bank's specific needs.

e-Banking is a highly customizable application. Customization is possible at both the presentation and functional levels.

This Help does not cover the customization options available to the user. The same are discussed in detail in the e-Banking Corporate Help and Retail Help.

Notational Conventions

This document uses the following notational conventions:

Notational Convention	Significance
	Signifies that a note follows.
	Signifies a warning message. The text in the warning is red in color.
Bold	Indicates names of fields, directories, files and buttons.
<i>Bold italics</i>	Indicates the file path and screen names.
<i>Italics</i>	Indicates information to be entered by you and messages displayed by the application.
SMALL CAPS	Indicates menu names.
Sample	Indicates sample scripts or files.

Introduction

e-Banking Solution is extremely flexible and can be customized to suit the requirements of the bank. An understanding of application commands is essential to fully exploit the functionality of e-Banking and create an effective user interface.

The bank can customize the application in accordance with its requirements. The systems administrator of the bank need to adhere to only a few specific inputs at the page level design. This enables the provision of a wide range of functions. Once you select a basic layout style at the time of installation, you can replace all images and backgrounds. The layout too is completely customizable. You can replace all images including menu option button images.

For ease of understanding, customization issues are discussed at the Presentation and Functional levels.

Presentation Level Customization

Each bank can customize the look-and-feel of the Finacle e-Banking solutions according to their requirements. At the presentation level, you can customize static parameters, menus, images, buttons, application flows, parameters, business logic and the duration to time-out a session.

The following can be executed at the presentation level:

- [Overall Maintenance](#)
- [Menu Maintenance](#)
- [Submit Button Customization](#)
- [Application Flow](#)
- [Parameterization of Files and Tables](#)
- [Session Time-out](#)
- [BayAudit](#)
- [Generic Email](#)
- [Journaling and Auditing](#)

Note:

For detail of FEBA customization, refer to Customization Framework in Architecture Manual.

Journaling and Auditing

The auditing feature in the eBanking application has been provided as an enhanced security measure to provide non-repudiation of transactions done and to provide flexibilities in terms of using the recorded data for the future.

Features

The auditing process takes place in a two stage process:

1. Journaling
2. Functional Auditing

The Journaling and Functional Auditing processes can be switched on or off. But for Functional Auditing to take place on an activity, Journaling on that event that triggers the activity must be enabled.

Journaling

Journaling is the process of recording the user's values entered on the screen as a dataset or name value pairs. The information to be recorded, if the transaction is a modify kind of transaction, will be the old values and new values which are the existing values and the new values the user has entered respectively, or only new values in case of other transactions like create or query or only old values in case of a delete kind of transaction.

The configuration information of the Journaling process is maintained in the Event.xml.

Functional Auditing

Functional Auditing is the process of recording the activity's progress at a particular checkpoint while processing an event. Each functional auditing process is associated with a primary table. The information audited here are the key of the primary table for the given transaction, additional parameters, status of the transaction and some other details.

A functional auditing process is enabled only if it is associated with a Journaling process and also the Journaling process must be enabled, but the vice versa is not true. The configuration information regarding the functional auditing is maintained in the Activity.xml.

XMLs and their Usage

[Event.xml](#)

[Activity XML](#)

Overall Maintenance

You can add or modify the static parameters, like **align** or **bgcolor**, associated with each image/object, or the property of a table/cell, etc using the recommended procedure.

The following are examples to modify static parameters:

Example:

If you want to specify the border color of a table as **Alice Blue** then specify `bordercolor="#F0F8FF"` in the following table declaration.

```
<table border=0 cellpadding=0 cellspacing=0 bordercolor="#F0F8FF" width=80% align="center">
```

Similarly, the specifications for the following colors are:

Antique White = #FAEBD7

Aquamarine = #7FFFD4

If you want to specify the back ground color, then specify `bgcolor="#FAEBD7"`

- ***The minimum image replacement involves inserting the Bank's logo, byline, corporate identity-based inputs viz., background, etc.***

Menu Maintenance

You can customize images for menu buttons according to the style used.

It is important to maintain a consistent naming convention for all images, across all styles. For example, if the **Account Statement** menu option image is **mnu_as.gif** then, the same convention should be followed for all styles. Typically, every button has two states - Up and Down, which are represented by two independent images. Follow the same convention for both states of menu button images.

You can modify the names of these image pairs using the ADMINISTRATION menu option in the main menu screen of e-Banking.

- **Ensure that the modified/new images are saved into the respective style directories under the images folder.**

Sub Menu Option Flexibility

The sub menu option feature offers flexibility to the administrator while providing access to the user. The administrator can give access to a sub menu option for a user, as required. The administrator has the permissions to add or delete any granular functionality from the menu profile of the user. For example, earlier if a user had access to the FUNDS TRANSFER module, the application would display the **Transfer** link in the menu for the user. By default, the user had access to all the sub menu options of the module. Now, the application provides the administrator greater flexibility in giving access to the user. The administrator can create a menu ID for each sub menu option. For example, a menu ID can be created for each sub option of the FUNDS TRANSFER module. If the user's menu profile is attached to any of these menu IDs, the application will display a link to the FUNDS TRANSFER module. If the user does not have access to any of the newly created menu IDs, the corresponding link will not be displayed.

To enable this functionality, modifications should be made to the following:

- **menu.jsp**

Modify the **menu.jsp** to check the access for each newly created menu ID, instead of the previously existing menu IDs. Each menu ID is appended to form the menu ID for the corresponding module. For example, if an user has access to self transfer, third party transfer and pending transfer, the menu ID for Funds Transfer will be FTSLF-FTTPT-FTPEN.

- **SideMenuBar.jsp**

Modify the **SideMenuBar.jsp**, which is different for every module, in the following manner:

Check that the Menu ID created in **menu.jsp** displays only the links selected for the user. For example, in case of funds transfer, if the menu ID created is FTSLF-FTTPT-FTPEN then only the following links will be displayed to the user:

- Self Transfer
- Third Party Transfer
- Pending Transfer

- **Navigation properties:**

Modify the **navigation.properties** file to display the change made to the Menu ID. The command, **com.infosys.bankaway.common.command.CheckMenuAccessCmd** is the first command called on clicking any link from the side menu bar. If you do not attach the menu ID **NEGRAT** used for the **Negotiated Rate** to the user

profile, the corresponding side menu bar JSP will read the menu ID received from **menu.jsp** and call different JSPs without the negotiated rate fields.

Submit Button Customization

An HTML submit button or an input image can be used to execute the submit action.

The image that replaces the HTML button will have to be saved under the respective style/images directory.

The following syntax is used to customize the submit button:

The typical syntax is as follows:

```
<input type="submit" name="Action.CorpUser.UpdateCorpUserProfile"
value="UpdateProfile">
```

The above displays a button with the label on the button as : **UpdateProfile**

If you want to display an image instead of the button and perform the same task, the syntax will be as follows:

```
<INPUT name= "Action.CorpUser.UpdateCorpUserProfile" type="image"
src="/web/L001/images/Gif.gif">
```

Application Flow

Application Flow for Admin

The bank can customize the application flow by executing other commands and displaying JSPs according to the flow required. For example, when the login page is submitted, the application executes the Java class `com.infosys.bankaway.user.loginCmd`. If login is successful, the `onSuccess` JSP is displayed and if it is a failure, the `onFailure` JSP is displayed. The bank can modify the default behavior, by adding another JSP before the `onSuccess` JSP is displayed.

In order to modify the application flow, it is necessary to understand the conventions used for commands and tags in the JSPs.

All JSPs in the application follow a convention for the Submit button names and images embedded in the HTML `<FORM>` tag. The same convention is followed in the `<A HREF>` tags in all pages. The names of the buttons and images have the prefix `Action`. For example,

```
<INPUT TYPE=SUBMIT NAME=Action.User.Login VALUE="Login">
```

```
<A HREF="com.infosys.bankaway.servlet.TransactionServlet?Action.User.Login=Y"> <img src=login.gif> </A>
```

When you submit a form, or click the hyperlink, the application submits the button name or the action parameter. The application detects this parameter and decides the action required.

It uses file `navigationproduct.xml`, in which action parameters are associated with commands. It also contains the JSPs to be displayed on success or failure of the command. Commands are Java classes which perform a logical business function. To modify the behavior of the application, modify the parameters in the `navigationproduct.xml` file.

The following is a sample from the `navigationproduct.xml` file:

```
User.Login.Commands= com.infosys.bankaway.user.loginCmd
```

```
User.Login.onSuccess=web/menu.jsp
```

```
User.Login.onFailure=web/signon.jsp
```

- **For details on the usage of various commands, refer to Finacle e-Banking solutions Technical Documentation.**

Note:

For auditing for FEBA, Check Audit Framework of FEBA in Architecture Manual.

Parameterization of Files and Tables

e-Banking behavior can be altered by changing the values of the parameters in the following files/tables:

- BankAway.properties File
- Property Manager Table
- Table Property Manager Table
- Batch.properties File

BankAway.properties File

The **BankAway.properties** file define the install time parameters required to start the application. These parameters are set during installation and the administrator does not require to change them subsequently. If necessary, they can be modified.

- ***A detailed documentation of the parameters in the file is available in Finacle e-Banking solutions Technical Documentation.***

Property Manager Table

The Property Manager table is maintained in the database. It defines parameters like user expiry period and logon password attempts.

- ***A detailed documentation of the parameters in the file is available in Finacle e-Banking solutions Technical Documentation. You can make changes to the parameters in this table using the PROPERTY MANAGER option in the e-Banking Administration menu.***

Table Property Manager Table

The Table Property Manager table stores information about audit and verification requirements for administrator activities. The administrator of the bank can modify these parameters as per the audit and workflow requirements of the bank.

- ***You can make changes to the parameters in this table using the TABLE PROPERTY MANAGER TABLE option in the Finacle e-Banking Administration menu.***

Batch.properties File

The **Batch.properties** file is one of the requirement for bringing up the batch program user interface for e-Banking. The parameters like Database URL, Database Name, User Name and Password as required in the application is to be set in the batch.properties file. You can also enable the multi-lingual interface for the application by setting the appropriate values in the LANGUAGE and COUNTRY fields.

- ***See Batch.properties topic in the Operations Guide for more details on batch.properties file parameters.***

Session Time-out

A user will be timed out of the e-Banking if the application interface is left idle for a fixed period. This session time-out period can be configured by the bank. This is an added security measure.

Once logged into e-Banking, the user can browse through the web site. If the session remains idle for the specified time, the user's session is logged out and the user is required to log in again.

The session time-out value is set in the **Bankaway.properties** file.

property name : SESSION_TIMEOUT

This property sets the session time. The value is in milliseconds. For example, 70000000. The session will expire automatically if the session is inactive for a duration exceeding this value.

Example:

SESSION_TIMEOUT = 90000000

Extended Timeout Facility

Some end user actions like compose mail, may require a duration more than the session time-out period set in the **BankAway.Properties** file. The extended time-out facility provides the end user an option to continue with the same session, when the session is about to time-out.

The application allows you to alert the user n seconds before the session expires. Specify this parameter in the JSP. The application displays an alert message n seconds before the session expires, prompting the user to click **Continue**, in the application if he/she desires to keep the session live. The session will not remain live if the user clicks **OK** on the pop-up message.

The following is a sample of the code used to extend time-out:

<!-- For making a jsp to throw popup message before session times out:-

- 1) copy and paste code from <script language="javascript"> to </script> tags.
- 2) put (onload = "jspTimeout();") in body tag as used below.(remember that this should come before any other parameter in body tag)
- 3) the value "30" should be replaced by the number of seconds by which you want to throw message

before time out.

----->

<BODY onload = "jspTimeout();" bgcolor= "#FFFFFF">

```
<script language="javascript">

var secondsbefore;

function jspTimeout(){

<% int strTemp= session.getMaxInactiveInterval();%>

var valueTimeoutSec = <%= strTemp%> ;

secondsbefore = 30;

var valueAlert = valueTimeoutSec - secondsbefore;

var valueAlertMilli = 1000 * valueAlert;

var timeOut = setTimeout("alertTimeout();",valueAlertMilli);

}

function alertTimeout(){

alert("your session is going to be timed out in " +secondsbefore + " seconds". Click 'Continue' to
continue);

}

</script>
```


BayAudit

BayAudit for Admin

The BayAudit feature is provided to capture the session values for a user and keep track of user activity. This is an optional feature. The bank can choose to view the data entered by the user while submitting the form using the BayAudit option.

The bank can log audit information either in the Audit Table (ADTT) or in data files with the extension .dat. If you specify the parameter, LOG_INT0_DATABASE, in the BankAway.properties file to Yes, the log is maintained in the ADTT table, else the logs are maintained in a data file. View the data files from server in the folder ..\setup\application\data.

- Modifying the audit.properties file

The following is a sample entry in the `audit.properties` file:

```
CommonAdt.OnSuccess=bankAwayUserInfo:session,RemoteIP:session,RemoteHost:session,Mode,BrowserType:session,DisplayMessage,DisplayMessageCode
```

CommonAdt.OnFailure=bankAwayUserInfo:session,RemoteIP:session,RemoteHost:session,Mode,BrowserType:session,DisplayMessage,DisplayMessageCode

In the above example, `CommonAdt` is the name for an audit type. The audit type is based on the module name. Each audit type can have different names like `MailsAdt`, `RequestAdt`, etc. However, the name of the audit type must end with `Adt` for the application to identify the entry as an audit entry.

Specify two entries - one for success and another for failure so that the application maintains a log of the success and failure of a request. The bank may want to maintain an audit on different parameters in case of success and failure.

The name of the parameters mentioned in the audit.properties file should be exactly the same as in the application.

Specify audit requirements of session level parameters as follows:

bankAwayUserInfo:session

Only the parameter name must be specified for audit of transaction level parameters.

- Modifying the Navigationproduct.xml file

Make an entry in the `Navigationproduct.xml` file at the point where audit is required, after you make an entry into the `audit.properties` file, for example,

```
CustInfo.UpdatePassword.Commands = CommonAdt.com.infosys.bankaway.user.command.ChangePasswordCmd
```

CustInfo.UpdatePassword.OnSuccess = <CURRENT>

CustInfo.UpdatePassword.OnFailure= <CURRENT>

Only those events in the **navigation.properties** file for which the parameter, **CommonAdt**, is specified will be audited.

Note:

For auditing for FEBA, Check Audit Framework of FEBA in Architecture Manual.

Generic Email

The Generic Email feature can be used to send a mail to other Relationship Managers of a bank. The need arises whenever a user attempts transactions which fall under special conditions decided by bank. The bank can enable this feature while customizing the NAV files provided in the product.

- **NAV files are navigation files written in Java, used wherever there are a series of events to execute and execution of next event depends on the output of the previous event.**

For example, if a user attempts a funds transfer in the local currency equivalent of more than two million, the bank may want to send a mail to the RM of the treasury department specifying details of the transaction.

The RM of the treasury department should be a e-Banking user.

The following are steps to modify the NAV file:

Step	Action
1.	Specify the IF condition for which the generic email has to be sent.
2.	Specify the following fields in the Cache Manager. These fields are required to send a mail: <ul style="list-style-type: none"> ▪ mailId - the ID of the mail recipient ▪ fromId - the ID of the mail sender ▪ subject - subject of the mail ▪ body - body of the message ▪ branchId - the ID of the recipient branch can be blank
3.	Now call the com.infosys.bankaway.mails.command.RetSendCmd command to send the mail. This command will use the information put in Cache Manager as specified in step 2 to send the mail.
4.	Specify the code to suppress any message returned by the RetSendCmd command.
5.	In case of either success or failure while sending the mail, existing behavior of navigation (without sending a mail) should be retained.

Report Generation

The format of the reports generated by the bank can be customized. The order of the title, header and fields to be displayed can be decided in the custom methods. These formats can be defined at the corporate level also.

The two methods used to generate reports are :

- Reports with the titles based on Report Type and Report Format.
- Reports with data in the required format, based on the parameters mentioned.

Business Level Customization

Business Level Customization

Each bank can customize the business logic of Finacle e-Banking solutions to suit their requirements. e-Banking allows you to customize various business level functions. The bank can specify the methods to convert the branch ID to the SOL ID, dealer reference ID and validate requests. The bank can also modify descriptions of requests and text of Alert messages.

The following can be executed at the business level:

- [Functional Customization](#)
- [Modifying Business Logic](#)
- [Error Code Range](#)
- [OpConsole](#)
- [Hold-Release Functionality](#)
- [Multiple Funds Transfer Functionality](#)
- [Drill Down Functionality](#)
- [Command.properties File](#)
- [Command.properties File](#)
- [Multi byte Support](#)

Functional Customization

The bank can modify the behavior of the application at a functional level by altering the JAVA code in custom classes. The following functionality provided by the application can be customized.

- **Get other Finacle Branch ID or SOL ID for a particular e-Banking Branch ID or SOL ID**

Different versions of Finacle store the branch ID differently. The bank can customize the method to convert the branch ID to the SOL ID. A default custom script is provided with the application.

- **Request Validation**

e-Banking allows the bank to add validations for any request through customization module. These include requests which cannot be accommodated by REQUEST_FLD_TABLE (synonym: RFLD) table.

- **Modify User Alert Messages**

Finacle e-Banking has a facility to send standard alerts to the user. The bank can modify the text of these messages.

- **Validate Dealer Reference ID in case of Negotiated Exchange Rate Transactions**

A valid dealer reference ID must be specified to the e-Banking during negotiated exchange rate transactions. The dealer reference ID is generally obtained from the treasury. The application helps to add logic to validate the dealer reference ID.

- **Validate File Upload**

e-Banking provides standard validations for bulk file uploads. Account rules and file formats are among many other parameters that are validated. The application helps to add validations, if required.

- **Host Access**

e-Banking provides the interface to Finacle core banking solution and Finacle CRM solution hosts. The local host which is e-Banking database, is also the default host for this application. Based on the bank's requirement, any host can be configured during customization. The bank can also customize to interface with other hosts apart from the one mentioned above.

Configuring other non-default hosts

e-Banking provides custom classes for each module to configure hosts that are not set as default hosts.

Step	Action
1.	There is a custom class for each module having two methods, <i>getPrimaryHost</i> and <i>getSecondaryHost</i> .
2.	These methods may or may not have request type parameters passed to them. If the Request Type parameter is present, then different hosts can be configured to meet different requests.
3.	Set the first host to be accessed in <i>getPrimaryHost</i> method.
4.	Set the second host to be accessed in the <i>getSecondaryHost</i> method. Secondary host will be accessed for

	the request only if connection to the primary host fails.
5.	Bank may not want to implement the secondary host. In such a case where the connection to the primary host fails, an error can be thrown to the end user stating that ' <i>Feature is not available. Try after some time</i> '.

Adding a new host

Finacle e-Banking solutions provides the facility to the bank to write the interface classes for any other back-end host, which conform to the architecture followed in e-Banking.

Step	Action
1.	Every module in eChannels has a HostAccess interface class. These can be used to add any new host.
2.	For every new host that is added, a new host-package for each module (interfacing with back-end access) should be created. All host interfaces related classes should get added to the new host-package.
3.	The main host interface class will have methods for preparing the messages to be sent to the host. This class will have separate methods for each feature available for back-end access in the corresponding module.
4.	Main host interface class will have to implement the HostAccess interface class provided for each module.
5.	If a new host does not support any request, an error message is displayed as part of the implementation of this request. It states that the feature is not supported by this host.

- **Transaction Details:** The bank can customize their own request form parameter details like Get Amount, Get Account, Get Broken Amount, Get Deposit Amount, Get Account Summary, etc. for a request ID.
- **For details on customizing transaction details, refer to the Finacle e-Banking solutions Technical Documentation.**
- **Extended Logging:** The bank can enable and customize the extended logging feature provided in e-Banking by modifying the **CustomOpConsole** class present in the **com.Infosys.bankaway.util** package. The extended logging feature enables a user to continue with the same session beyond the default duration specified for the activity.

Modify **CustomOpConsole** class to set the following:

/ Update the following with the path where the config files are present */*

```
errConfigFile = "C:/errfile.cfg";
```

```
warnConfigFile = "C:/warnfile.cfg";
```

```
limoConfigFile = "C:/limoclient.cfg";
```

/ Update the Bank Id */*

```
bankId = "IBKL";
```

```
/* The error code for Fatal Exception. If this is present in errfile.cfg then FatalExceptions are  
sent to OpConsole. This is because like FatalException do not have error codes like  
NonFatalExceptions */
```

```
BafeErrCode = "0";
```

```
BafeErrDesc = "BankAway Fatal Exception";
```

The OpConsole is enabled by default. However, errors and warnings to be sent to the OpConsole have to be specified in the configuration files.

Remove the implementation in **CustomOpConsole.logMessage(String, String)** to disable the OpConsole.

Modifying Business Logic

A custom class is provided with the application, for functionalities such as HTML form validations and online/offline accounts inquiry. The bank can change these functionalities by modifying the Java code in the `com.infosys.bankaway.custom.Custom` class.

- **See *Finacle e-Banking solutions Technical Documentation* for details of the various methods in this class.**

The bank can also customize other functionalities and the business logic.

- **See [Functional Customization](#) topic for a list of the business functionalities that can be customized.**

Error Code Range

e-Banking allows addition of new error codes as part of customization to enable new features. To avoid a conflict with the existing error codes, a separate slot is provided. Error code for FEBA is autogenerated.

OpConsole

e-Banking provides a mechanism to log all errors to the OpConsole server by linking it to the Finacle e-Banking application. Whenever an error or warning is generated by e-Banking, the message can be logged in OpConsole.

This feature can be used by installing the OpConsole server and updating the IP configurations in the **limoclient.cfg** and **limosrvr.cfg** configuration files.

The following files should be copied to the working directory of the application server machine, for example, **c:**

- **limoclient.cfg**
- **limosrvr.cfg**
- **errfile.cfg**
- **warnfile.cfg**

Hold or Release Functionality

Hold or Release functionality affects a change in the transaction flow because of the need to release payment at a later point of time and not immediately at the end of the approval cycle. When a transaction is entered, it goes through the usual approval cycle. At any stage, the transaction can be put on hold using the Hold functionality.

If a bank does not wish to implement this functionality, then the parameter **TXN_HOLD_RELEASE_REQD** has to be set to N in the PRPM table. The bank administrator can also control this functionality at corporate level.

Multiple Funds Transfer Functionality

The bank can decide whether or not, it wants to provide multiple funds transfer facility. The parameter **FTSingleOrMultiple**, is used to decide whether or not a bank is providing this functionality. This parameter can be set to values Single or Multiple, signifying which functionality is supported by the bank.

Drill Down Functionality

To drill down to transaction level details from an online statement, you can either choose to take transaction date or the value date of the transaction as the basis, to fetch further details of the transaction. The bank can customize the date it wants to use to drill down to further details. According to the need of the bank, the custom method can be modified to set the transaction date or value date.

The bank can decide the requests for which it wants to give the drill down to further details facility. The custom method for request can be modified, to add new activity types that require drill down. If the online request is completed successfully, the custom method determines whether this is a candidate for drill down.

The **BackEndRefId** field is passed to the custom method that will take field **125** of the ISO message as an input string and pass it to extract the **BackEndRefId** and **TxnDate** fields in the TXDT table. In case backend does not support ISO messages, this functionality should be customized to get the inputs from the supported message format.

After a transaction is successfully performed, the logic for extracting the **BackEndRefId** and **Txn Date** from the response (from backend) can be customized to suit a bank's requirement.

Command.properties File

Command.properties file contains a list of all the commands used in the application. Corresponding to those commands, it also contains the menu options that give access permissions to the commands. When a command is executed, the application checks if the user has at least one menu option that is linked to this command.

Some of the menu options that are given for the fundamental commands are as follows:

Command Name	Description
Basic	If a command is given ' Basic ' permissions, then all the menu options can access that command. These are typically used for basic functions like signon, etc.
MenuOptionM	A menu option that has only maker permissions.
MenuOptionV	A menu option that has only verifier permissions.
MenuOption	A menu option that has both maker and verifier permissions. Menu options with this naming convention are reserved for administrator use. This has an option to use maker checker available.
TRAN	All commands that perform a transaction should have TRAN. A relationship manager logging in as a corporate or retail user cannot execute a command having a TRAN.

Custom Hooks

The Custom Hooks for Business Level Customization are called from the following modules:

- [User Menu Linkage](#)
- [Payments](#)

Funds Transfers

- [Role ID for an RM](#)

User Menu Linkage

To display messages to the user, based on the registration flag there is a custom hook available. This class will be called if the registration flag is 'Y'.

com\infosys\bankaway\util\custom\RegistrationCustom.java

This custom class has a method defined as

```
public static String getRegistrationUrl(String regUrl, String menuName)
```

and this will return the custom registration url. Here we are passing the default registration url now. Based on the menu name you can return customized registration urls.

Banks can customize the already available JSPs , to the desired requirement .The JSPs which are available for customization are:

web\L001\corporate\jsp\promotionMsg.jsp

web\L001\corporate\jsp\registrationMsg.jsp

They have default messages to be displayed to the customers. In case they want to display any other messages as per the business requirement , they can either customize this JSP or create their own JSP for the purpose.

Payments

Package structure of custom hooks for Payments Module are as follows:

- **com.infosys.bankaway.eipp.custom**
- **com.infosys.bankaway.common.custom**

The following custom hooks are available under the payments module:

- [BankDetails](#)
- [ChargeCalculator](#)
- [InitialNetworkValidations](#)
- [PaymentsLien](#)
- [ProcessPaymentsCustom](#)
- [RMApprovalRequirementStatus](#)
- [TransactionValidation](#)
- [MultiCurrencyHook](#)

Bank Details

Beneficiary Bank details and routing number related information is stored in BKDT and BKRT tables respectively, these details are accessed through the methods available in this Custom Hook.

Customization required on accessing the beneficiary bank information can be done by implementing the methods in this custom hook as per the requirement.

Class Files Used for Customization

The Class File that can be used for customization are:

com.infosys.bankaway.eipp.custom.BankDetails

--public Vector getBanksList(BankAwayUserInfo userInfo, BankDetailsListCrit bankDetailsListCrit, Connection dbConnection) throws BankAwayFatalException, BankAwayNonFatalException

The method is used to fetch the list of Banks, a Corporate or a Retail user tries to fetch based on the search criteria provided by the user. Default Implementation provided is to fetch the list of Banks from BKDT table based on the query criteria provided of the user.

--public Vector findRoutingNumber(BankRoutingNumberListCrit bankRoutingNumberListCrit, Connection dbConnection) throws BankAwayFatalException, BankAwayNonFatalException

This method is used to fetch the routing number based on the criteria provided by the Corporate or Retail User. Default Implementation provided is to fetch the list of routing number of Bank from BKRT table based on the query criteria of the user.

--public Vector findBankId(BankAwayUserInfo userInfo, BankDetailsInfo bankDetailsInfo, Connection dbConnection) throws BankAwayNonFatalException, BankAwayFatalException

This method is used to fetch the list of Banks based on a filter criteria provided by the user. Default Implementation is to fetch the list of Bank satisfying the search criteria from the BKDT table.

--public BeneficiaryInfo getBankRefNumberForBeneficiary(BankAwayUserInfo userInfo, long lBeneficiaryId, Connection dbConnection) throws BankAwayFatalException, BankAwayNonFatalException

This method is used to fetch the Bank Reference Number of the Beneficiary. Default Implementation is to fetch the Bank Reference information of a Beneficiary from PAYM table based on the Implementation Bank ID and Beneficiary ID.

--public BankDetailsInfo createBankDetails(BankAwayUserInfo userInfo, BankDetailsInfo bankDetailsInfo, Connection dbConnection) throws BankAwayFatalException, BankAwayNonFatalException

This method is to used to create Bank information. If the Bank, a user is trying to identify is not available in the list of Banks in the Implementation Bank's Database, then the Default Implementation is to create a record in the BKDT table.

ChargeCalculator

Charges and Commissions for transaction are not computed by the Finacle e-Banking application. This custom hook is provided to implement charge, commission calculating logic as required in the site, like interfacing the application with third party charge servers.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom. ChargeCalculator

--public static PaymentVO calculateCharges(BankAwayUserInfo userInfo, PaymentVO paymentVO)

throws BankAwayNonFatalException, BankAwayFatalException

This method takes PaymentVO and userInfo and uses this information to compute the charges and commission as required in the Implementation. Default Implementation provided is to compute Charges at 5% of the Transaction Amount and Commission Amount at 10% of the Transaction Amount.

--public static BatchVO calculatePenaltyCharge(BankAwayUserInfo userInfo, BatchVO batchVO)

throws BankAwayNonFatalException, BankAwayFatalException

This method takes the BatchVO and userInfo as inputs and computes the PenaltyCharge of a Presented Bill, when the bill is overdue. No Default Implementation provided for the same, Implementation for the same can be done as required.

--public static PaymentVO getChargeToBeAdded(BankAwayUserInfo userInfo, BatchVO batchVO, PaymentVO paymentVO) throws BankAwayNonFatalException, BankAwayFatalException

This method takes userInfo, PaymentVO and BatchVO as input. In case of Collection, if the Charge currency and Commission currency is not same as the transaction currency, then this method is used to convert Charge and Commission amount in the transaction currency. No Default Implementation provided.

InitialNetworkValidations

The Initial Network Validation custom hook has the functionality of determining Payment network, if the user has not chosen a network for payment based on the network priority set by the Bank. This custom hook also houses method which do the high and low level validations.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom.InitialNetworkValidations

```
--public static String determineNetwork(BankAwayUserInfo userInfo,NetworkPropsCrit networkPropsCrit,Connection dbConnection) throws BankAwayNonFatalException,BankAwayFatalException
```

This method is used for determining the network associated with a particular transactions and doing the high level validation on the selected networks. The database fetch is done based on transaction type which is passed in Crit Object. The selected network is passed for high level validation which returns the boolean true value on successful validation. This method returns the network type on successfully determining the network. Default Implementation is provided by selecting the network from NPRT table and executing the high level validation of the network.

```
--public static boolean doHighLevelValidations(StringsNetworkId) throws BankAwayNonFatalException
```

This method is used for doing high level validation for the network associated with a particular transaction. This method is called from the determineNetwork method for doing the high level validation, if the network is not known. It takes a networkid as a input and returns a boolean value depending on whether the network ID has undergone the validation successfully or not. In case of failure, exception is thrown to the user with the appropriate message. This method will be changed based on the NetworkId by including a new if condition and inside the if condition appropriate Logic for high level validations can be done. Returns a boolean value.

```
--public static boolean doLowLevelValidations(String strNetworkId) throws BankAwayNonFatalException
```

This method is used for doing low level validation for the network associated with a particular transaction. It takes a networkid as a input and it returns a boolean value depending on whether the network Id has undergone the validation successfully or not. In case of failure, an exception is thrown to the user with the appropriate message. This method will be changed based on the NetworkId by including a new if condition and inside the if condition appropriate Logic for low level validations can be done.

PaymentsLien

If the host supports online Lien marking functionality, this custom hook can be used to mark and remove Lien on an account.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom.PaymentsLien

```
--public          static          PaymentsLienOutputVO          removalOfLien(BankAwayUserInfo  
bwayUserInfo,PaymentsLienInputVO          paymentsLienInputVO)          throws          BankAwayFatalException,  
BankAwayNonFatalException
```

This method can be used to remove Lien on an account, if the host supports Lien marking and the implementation requires Lien removal functionality enabled.

```
--public static PaymentsLienOutputVO markLien(BankAwayUserInfo bwayUserInfo,PaymentsLienInputVO  
paymentsLienInputVO) throws BankAwayFatalException, BankAwayNonFatalException
```

This method can be used to mark Lien on an account, if the host support Lien marking and the implementation require Lien marking functionality to be enabled.

ProcessPaymentsCustom

Using this custom Hook, online payment processing can be customized. If online payment processing requires these methods, they have to be implemented to achieve the required functionality.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom.ProcessPaymentsCustom

```
--public      static      ProcessPaymentsOutputVO      processOfflinePayments(BankAwayUserInfo  
bwayUserInfo,ProcessPaymentsInputVO processPaymentsInputVO) throws BankAwayFatalException,  
BankAwayNonFatalException
```

This method is not supported in this version of the application.

```
--public static com.infosys.bankaway.eipp.custom.HostAccess getHostByMarkerForPymt(String bankId,  
String hostMarker) throws BankAwayFatalException, BankAwayNonFatalException
```

This method is not supported in this version of the application.

```
--public static HostAccess getClassForHost(String networkId) throws BankAwayFatalException
```

This method is used to return network specific bean instances. It takes in network id as input. The returned object implements HostAccess interface which gives access to the various method in Payment Services bean.

RMApprovalRequirementStatus

During Beneficiary addition, this custom hook can be used to decide whether for Beneficiary addition Relationship Manager approval is required or not.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom.RMApprovalRequirementStatus

```
--public RMApprovalRequirementInfo rmApprovalRequired(BankAwayUserInfo userInfo,CacheManager  
cacheManager) throws BankAwayNonFatalException, BankAwayFatalException
```

This method determines whether during beneficiary addition RM approval is required or not.

TransactionValidation

When all bill validation conditions are successfully completed, and the implementation bank still wants to do some Non-standard validation for a Bill, then this can be achieved using the following custom hook.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom.TransactionValidation

```
--public static BatchVO validateBill(BankAwayUserInfo userInfo, BatchVO batchVO)throws  
BankAwayNonFatalException, BankAwayFatalException
```

This method validates bill information with the extra validations as required by the Implementation Bank and returns the updated batchVO

MultiCurrencyHook

This Custom hook decides for two given currencies in a multi-currency transaction, whether RM approval is required for that particular transaction or not.

Class Files Used for Customization

Class File that can be used for customization are,

com.infosys.bankaway.eipp.custom.MultiCurrencyHook

This method decides whether RM approval is required for multi-currency transaction between two given currencies.

Interest Computation

This custom hook is responsible for getting the interest earned on the Pool and the applicable interest rates from the Bank. The report for sum of daily balances for a pool and the Pool Information is passed to the Custom Hook and a vector containing the pool information (with changed interest rates if applicable) and the interest earned by the pool for that period is received back. Using this vector the records in INTT table are created.

Custom Hook Details

InterestComputationCustom is the custom file which has to be implemented. The interest calculation part requires sending the message to the bank and receiving the response.

Other files to be used are:

AccountPoolMasterListData: - Contains the pool Information. The object of this class is passed as input parameter to the custom hook. The Interest rates should be set to the values which are applicable if there is a change.

The Vector object returned by this custom hook contains the following in the order given below.

1. Pool Information (AccountPoolMasterListData)
2. Interest Earned (Double)

Imaging

Package structure of custom hooks for Imaging Module are:

- **com.infosys.bankaway.accounts.custom.AccountCustom**
- **com.infosys.bankaway.accounts.bancsconnect2_0.BancsConnect**

This custom hook is used for getting the image, for the requested cheque's in case the third party image software, supports API interface instead of URL interface.

Customization required on returning the image data can be done by implementing the methods in this custom hook as per requirement.

Public Vector **getUrlForImage** (BankAwayUserInfo userInfo, AccountSummary accountSummary, String chequeNumber) throws BankAwayNonFatalException, BankAwayFatalException {

This method is used to return the list of images associated with the particular cheque's number which are captured.

File Upload Custom Hooks

Package structure of custom hooks for File Upload module are:

- `com.infosys.bankaway.admin.fileupload.custom`
- `com.infosys.bankaway.fileupload.custom`
- `com.infosys.bankaway.fileupload.custom.validation`
- `com.infosys.bankaway.batch.filesdpsch.custom.validation`

Admin custom package is used to validate **fileupload** relating to tables namely FUFU, FFPM, FUPT.

Fileupload custom package is used for decrypting and validating file contents. The way in which these processes are performed varies between banks. Hence, these files can be modified accordingly.

Batch custom package is used for validating the records/header in file, which have to be inserted into tables (BTDT & PYRQ) for payment.

Custom Hook Details:

Class files that can be used for customization are :

1.) **com.infosys.bankaway.admin.fileupload.custom** package

FUCustom.java

The validations related to admin use case for fileupload relating to tables namely FUFED, FFPM, FUPT are done in FUCustom. It has methods related to each table which are called from the ejb for validations during inserting a new record or updating a record.

2.) com.infosys.bankaway.fileupload.custom package

FileDecrypt.java

Implements `com.infosys.bankaway.fileupload.ejb.FileDecrypt.java` interface and contains methods to decrypt the file contents, if encrypted.

```
-- public Amount getWFamtForPTfiles(FileUploadDetailsVO fuDet, BankAwayUserInfo userInfo, Hashtable
crdCodes) throws com.infosys.bankaway.common.BankAwayNonFatalException,
com.infosys.bankaway.common.BankAwayFatalException
```

This method is used to get amount for workflow from header or from UI in case of pass-through files.

CustomFileUpload.java

Contains common methods to get decryption class to be used, transaction type of a product type and validation class to be used for a product type before validating the file.

```
-- public static boolean savePassThrFile(FileUploadDetailsVO fuDet, BankAwayUserInfo userInfo) throws
com.infosys.bankaway.common.BankAwayNonFatalException,
com.infosys.bankaway.common.BankAwayFatalException
```

This method is used to save the pass-through file contents in a specific directory. Currently no implementation is provided for this method. It has to be written during customization.

```
--public static String getRejectionInd(String mFileType, String mCorpld) throws
com.infosys.bankaway.common.BankAwayNonFatalException,com.infosys.bankaway.common.BankAwayFatalException
```

This method is used to get rejection indicator, when it is corporate dependent. Currently default return type is 'File Level (F)'

```
-- public static String getBrldFrmAcid(String mBankId, String mBranchId, String mAcid, String mRoutingNo,
String mNWId) throws com.infosys.bankaway.common.BankAwayNonFatalException,
com.infosys.bankaway.common.BankAwayFatalException
```

This method is used to derive branch ID from account ID. Currently "BRC" is being returned as default value.

CustomCmsUpload.java

This class contains methods which returns the data after decrypting, return validation class object to be used for validation (com.infosys.bankaway.fileupload.custom.validation.ValidateCmsTranUpId()).

CmsFileUpIdDecrypt.java

This class is used to decrypt MIS file contents.

3.) com.infosys.bankaway.fileupload.custom.validation package

CmsCloseBalRec.java

This class is used to store the closing balance information uploaded in the MIS File. This class is passed as a parameter to an internal method to validate the uploaded information.

FileCmsFormatStruct.java

This class is used to store the file format information for MIS Files. The class stores the field name, start and end position of the field in the file and the data type i.e., character or string. A vector of this class is used to store all the fields and its information. This vector is also used while parsing the file and interpreting its content.

fileUploadDecrypt.java

Class which implements **com.infosys.bankaway.fileupload.ejb.FileDecrypt.java** interface. Same as FileDecrypt.java.

HeaderDetails.java

This value object is used to hold header details after reading header from file.

McmdRecordDetails.java

This value object holds complete details of a single record read from DCT/DC2 product type file which is used during validation.

RecCmsStruct.java

This class is used to store the MIS transaction details, uploaded in the MIS file. It stores information about all the fields of MIS which are uploaded. This class is passed as a parameter to an internal method to validate the uploaded information.

RecMcmdStruct.java

This value object is used to hold details of record from DCT/DC2 product type file.

RecordCmsDetails.java

This class is used to store the file level information to be used later for validation of the uploaded MIS file. This is used in processCmsUpd and validate methods of ValidateCmsTranUpd.

RecordDetails.java

This value object holds complete details of a single record read from SAL/DDB/DDR/VND product type file which is used during validation.

RecSAIStruct.java

This value object is used to hold details of record from Salary Payments (SAI) product type file.

RecStruct.java

This value object is used to hold details of record from SAL/DDB/DDR/VND product type file.

SAIRecordDetails.java

This value object holds complete details of a single record read from Salary Payments (SAI) product type file which is used during validation.

ValAcctDetails.java

This value object is used to hold details of an account to be validated.

ValAcctLinkDetails.java

This value object is used to hold details externally linked account in case of DCT/DC2 product type files.

ValCmsAcctDetails.java

This class is used to store account related information present in a record of uploaded MIS file. This class is then passed to an internal method to validate the account.

ValidateBNFTranUpd.java

Validation class used to validate Beneficiary/Debtor(BEN) file.

ValidateBPYTranUpd.java

Validation class used to validate Payments(BPY) file.

ValidateCLITranUpd.java

Validation class used to validate Direct debit upload(CLI) file.

ValidateCmsStat.java

This class is used to store Field level validation information for the uploaded data in a MIS file. For instance, the information like if field name is mandatory, the type of justification, the pad character that is used etc.

ValidateCmsTranUpd.java

Validation class used to validate MIS/CMS file

ValidateMcmdStat.java

This class is used to store Field level validation information for the uploaded data in DCT/DC2 files. For instance, the information like if field name is a mandatory indicator, the type of justification, the pad character that is used etc.

ValidateMcmdTranUpd.java

Validation class used to validate Transaction upload type1 (DCT)/Transaction upload type 2(DC2) files.

ValidateSAIUpload.java

Validation class used to validate Salary payments (SAI) file.

ValidateStat.java

This class is used to store Field level validation information for the uploaded data in a SAL/DDB/DDR/VND files. For instance, the information like if field name is a mandatory indicator, the type of justification, the pad character that is used etc.

ValidateTranUpd.java

Validation class used to validate Salary Transfers(SAL)/VendorTransfers(VND)/Dealer debit (DDB)/Supplier debit(DDR) files.

ValidateVNIDCITranUpd.java

Validation class used to validate Vendor payments(VNI)/Inter-bank transactions upload(DCI) files.

XfrrCheckDetails.java

This value object is used to hold details of an account for External funds transfer rules check.

XfrrMcmdCheckDetails.java

This value object is used to hold details of an account for external funds transfer rules check in case of DCT/DC2 product type files.

4.) com.infosys.bankaway.batch.filedspch.custom.validation package

CommonValBatch.java

This class contains general methods used when batch programs are run on the uploaded file contents.

-- public String getRejInd(FileValidateDetailsVOfileDetailsVO,BankAwayUserInfo userInfo)

This method determines the rejection indicator if REJECT_IND in FUPT table is "C" i.e. corporate dependent. The function can return either "F" for file level rejection or "R" for record level rejection.

-- public static Vector getAccountList(BankAwayUserInfo userInfo, FileValidateDetailsVO fileDetailsVO)

It returns the list of all valid account ID associated with the corp ID.

-- public static HashMap readFFPMFmt(FileValidateDetailsVOfileValDetVO, BankAwayUserInfo usInfo)

It returns the hashmap containing all the fields that are part of the file format that is specified for a particular corporate. Along with the fields it also contains the sequence, field width, mandatory indicator etc.

-- public static HeaderDetails validateHeader(FileValidateDetailsVOfileDetVO, BankAwayUserInfo usInfo)

This method finds the header part in the data and calls getHeaderDetails() method to find out the individual values and also validates those values.

```
--private      static      HeaderDetails      getHeaderDetails      (HeaderDetailsheaderDet)      throws  
com.infosys.bankaway.common.BankAwayNonFatalException,com.infosys.bankaway.common.B  
ankAwayFatalException
```

It finds the individual field in the header data , validates those fields, forms an object and returns.

```
--public static boolean isValidDate(String dateStr, int dateFormat, String fieldSeparator) throws  
com.infosys.bankaway.common.BankAwayNonFatalException
```

It validates the date.checks for leap year also for valid months and days.

```
-- private static int getMonthNum(String monthShortName)
```

It returns the month number for given month name(short name).

```
-- synchronized public ValidateBnfVO validateBnf(ValidateBnfVOBnfVO,BankAwayUserInfo userInfo,String  
srecBankId,StringsrecBranchId) throws BankAwayFatalException, BankAwayNonFatalException
```

It validates the beneficiary if it exists or not, given the various details, also determines the bnf_id if not given and bnf list ID.

```
--private      Vector      validateRouteNetwork(String      sBankId,      String      sRouteNo,      String  
sNetworkId,BankAwayUserInfo userInfo) throws BankAwayFatalException, BankAwayNonFatalException
```

It validates the network ID, routing number and bank_id combination.

```
-- private void validateACH(String sfreeField1 , String sfreeField2,BankAwayUserInfo userInfo) throws  
BankAwayNonFatalException
```

If the network is ACH then checks if other necessary details like company code description, product type are given or not. if Yes, then validates them.

```
-- public static boolean isValidAmount(String localeValue, String currency,Locale userLocale)throws  
BankAwayFatalException, BankAwayNonFatalException
```

This method is used to validate the Amount. It takes the user Locale and currency code and ascertains the number of decimals allowed for that particular currency code. It then validates the amount by comparing the length of fraction of the amount value with the number of decimal places actually allowed for that particular currency code.

```
-- private static String getNonLocaleAmountString(String fieldValue, Locale userLocale) throws  
BankAwayNonFatalException
```

This function parses and validates the passed field value as per the locale specific format and then returns the amount value in String having standard decimal format.

```
-- public static String getRecords(String sfileData,String sEORInd, int ibegin,int iend)
```

It returns the string of record data which contains all those records which are processed by 1 thread.

```
-- public static boolean insertRecordInFULT(String sFileSeqNum, intiErrorCode, int iRecNum, String  
sRecData, String sErrMsg,BankAwayUserInfo usInfo)
```

It inserts the details of the records which are invalid or have failed insertion in BTDT or PYRQ.

CustomFileValidate.java

This class contains methods, which can be used to get transaction type for a product type, get the table into which payment details have to be inserted, validation class that has to be used to validate the product type, object into which data has to be set when VNI/DCI files are being validated.

FileValidateBatchInterface.java

This interface defines the required methods to be implemented by the validation classes.

ValidateBnfVO.java

This value object holds data of the beneficiary when Beneficiary file is being validated.

ValidateMCMDTranBatch.java

Validation class is used to validate Transaction upload type 1 (DCT)/ Transaction upload type 2 (DC2) files.

ValidadateSAITranBatch.java

Validation class used to validate Salary payments (SAI) file.

ValidateTranBatch.java

Validation class used to validate Salary Transfers(SAL)/VendorTransfers(VND)/Dealer debit (DDB)/Supplier debit(DDR) files.

ValidateTranBatchBNF.java

Validation class used to validate Beneficiary/Debtor(BEN) file.

ValidateTranBatchBPY.java

Validation class used to validate Payments(BPY) file.

ValidateTranBatchCLI.java

Validation class used to validate Direct debit upload(CLI) file.

ValidateVNIDCITranBatch.java

Validation class used to validate Vendor payments(VNI)/Inter-bank transactions upload(DCI) files.

Calculate Charge and Commission Amount

The Calculate Charge Amount and Commission Amount Hook is used to calculate the charge & commission amounts for a transaction.

Input to the custom method

The inputs are **ValidateFundsTransferVO** and **User Information**.

Output from the custom method

This method will return the **ValidateFundsTransferVO** object after assigning values to the following fields:

- Charge Currency
- Charge Amount
- Commission Currency
- Commission Amount

If not implemented, this method will return the **ValidateFundsTransferVO**, after assigning the following values:

- Charge Currency = Home Currency
- Charge Amount = 0.0
- Commission Currency = Home Currency
- Commission Amount = 0.0

Mark Lien on an Account

The Mark Lien on an Account Hook is used to mark lien on the amount if it is a future dated payment.

Input to the custom method

The input is **RestraintObjectVO** and **User Information**.

Output from the custom method

This method will return **RestraintReturnObjectVO**, which has two flags:

SuccessFailure and **EnterQueue**.

If **SuccessFailure** is true then in XFRQ, txn_status is updated to PEN.

If **SuccessFailure** is false, check for **EnterQueue** is performed.

If **EnterQueue** is True, the user is asked whether he/she wants to go offline:

If Yes then **markLien** method of the custom hook is called again for offline marking of lien. On return of success XFRQ is updated to WAT (Waiting for Authorization).

If No then in XFRQ, txn_status is updated to LMF (Lien marking Failed.).

If not implemented, this method will return true for both **SuccessFailure** and **EnterQueue**.

Removal of Lien on an Account

The Removal of Lien on an Account Hook is used to remove lien from an account.

Input to the custom method

The input is **RestraintObjectVO** and **User Information**.

Output from the custom method

This method returns **RestraintReturnObjectVO**

If not implemented, the value true is returned in all cases.

Offline Processing of a Transaction

The Offline Processing of a Transaction Hook is used to process a hot payment offline.

Input to the custom method

The input is **ProcessFTInputVO** and **User Information**.

Output from the custom method

ProcessFTOutputVO has a flag called **SuccessFailure**.

If true, then in XFRQ, txn_status is updated to WAT (Waiting For Authorization).

If false, then insert into XFRQ with status as E (failure) and update txn_status in XFRQ to CLO (CLOSED).

If not implemented, this method will return true.

Role ID for an RM

This custom hook is provided to get the Role ID for an RM .

By default the method returns an empty string .

Input

The input is RM ID

Output

The output is RM's Role ID.

Framework Level Customization

Framework Level Customization

This section provides details on the constructs and points of customization in the Finacle eBanking Platform. As a prerequisite, to get a good understanding of the customizable components, users should have familiarity with the architecture overview of Finacle eBanking.

This section lists the customization options available across each layer in the product.

- [Customization Infrastructure](#)
- [Achieving Customization in Type System Layer](#)
- [Customization in Formsgroup Layer](#)
- [Customization in Service Formsgroup Layer](#)
- [Customization in Service Layer](#)
- [Customization in Transaction Layer](#)
- [Customization in TAO/TMO Layer](#)
- [Customization in DAL Layer](#)
- [Customization in HIF Layer](#)
- [Context](#)
- [Error Code/ Incidence Code Customization](#)
- [Customization in UI Layer](#)
- [Multi Branding](#)
- [Customization of CorpAdmin Framework](#)
- [Customization of Formmanagement Framework](#)
- [Customization Toolkit](#)
- [Entity Reference](#)
- [Caching Framework](#)
- [Filtration Framework](#)
- [Common Event Registry Framework](#)

Customization Infrastructure

Finacle eBanking product strategy has been to focus on high productivity and consistent maintenance and implementation experience across installations and future releases. The product has prescribed a structured approach to building product components and means and points to adopt the product to regionalized specifications and requirements. While the out of the box scenarios and functions may work in many cases, it is inevitable that each enterprise implementation, may want to adapt to very specific styles and business requirements. Hence a proactive attention from product design perspective helps in ensuring a customization adopted at a specific site, does not hinder the product growth and upgrades in future. A disciplined and contractual adherence to customization prescriptions as laid down by the product helps the installed base to benefit from product upgrades and easy migration paths to later product releases.

Objectives

This section describes main goals of the Customization Infrastructure.

Extensibility

Below diagram depicts how e-Banking application may be customized to meet Bank's need:

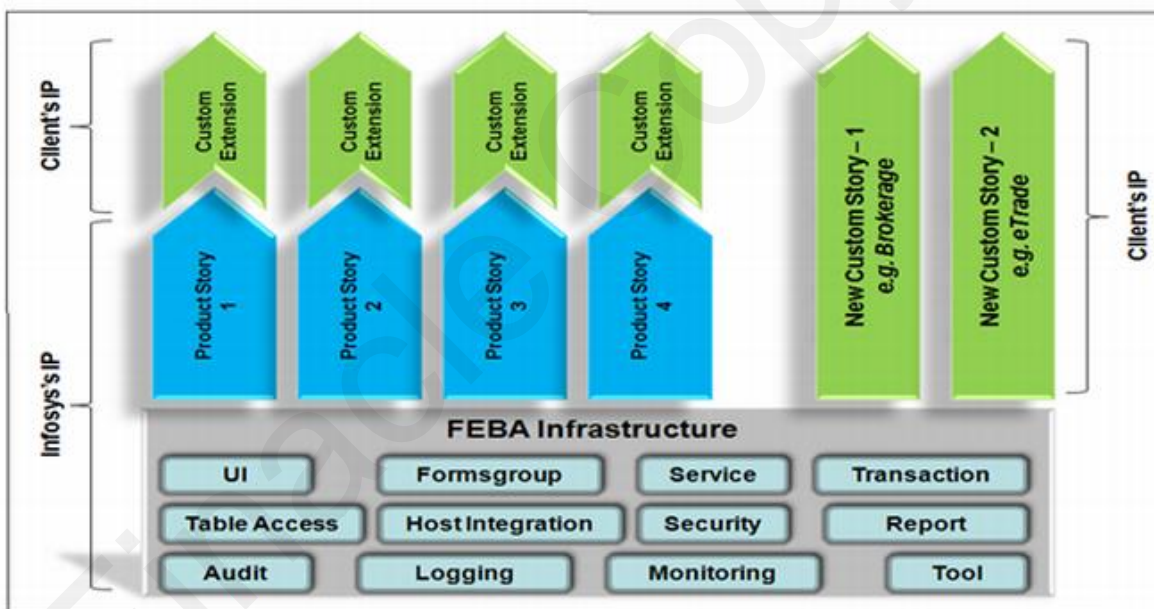


Figure 1 Enabling extensions around the Finacle eBanking product

As shown in the diagram, FEB Architecture provides a strong platform to build product stories. But based on Bank's requirement, some stories may need to be extended to support additional functionalities. At the same time new custom stories can be developed using the same platform. The customization infrastructure provides all required customization options to cater to all such requirements.

Upgradability

Product upgradability with minimum effort is one of the key concerns of the Customization Infrastructure.

To facilitate seamless upgradability, it enforces extension instead of customization in most of the layers. There are several hook points exposed that can be used to inject new functionalities in existing product flow. So, custom code has very little coupling with product code and can interact with product code only through these public hooks. Application of such Inversion-of-Control pattern helps to upgrade the product without impacting custom codes, so long as hook interface remains same.

Maintainability

It is assumed that the product binaries deployed at a site will remain un-tampered and isolated from the customization done therein. The product engineers are generally oblivious and with good intention, of additional components added or function enhancements achieved around the base product offering. Remaining independent allows the product team to plan for technical and product fixes and patches to be deployed at the site without impacting the customization code. An impact of change can now be considered at 2 levels, i.e.e. at product engineering, and customization code. As long as they needs not infringe on each other, sufficient maintainability is achieved. A well maintained installation can delay maturing and have a good stability over time.

Cost

Most of product features are created declaratively and allow for configuration based deployment at site. The impetus is to reduce customization cost by avoiding code incorporation and extensive re testing and regression. All customizations have been clearly categorized as configuration, minor or major, additions depending upon the level changes and the artifacts used for the change. By targeting customizability by configuration, much of the cost is absorbed by supporting teams without additional project costs.

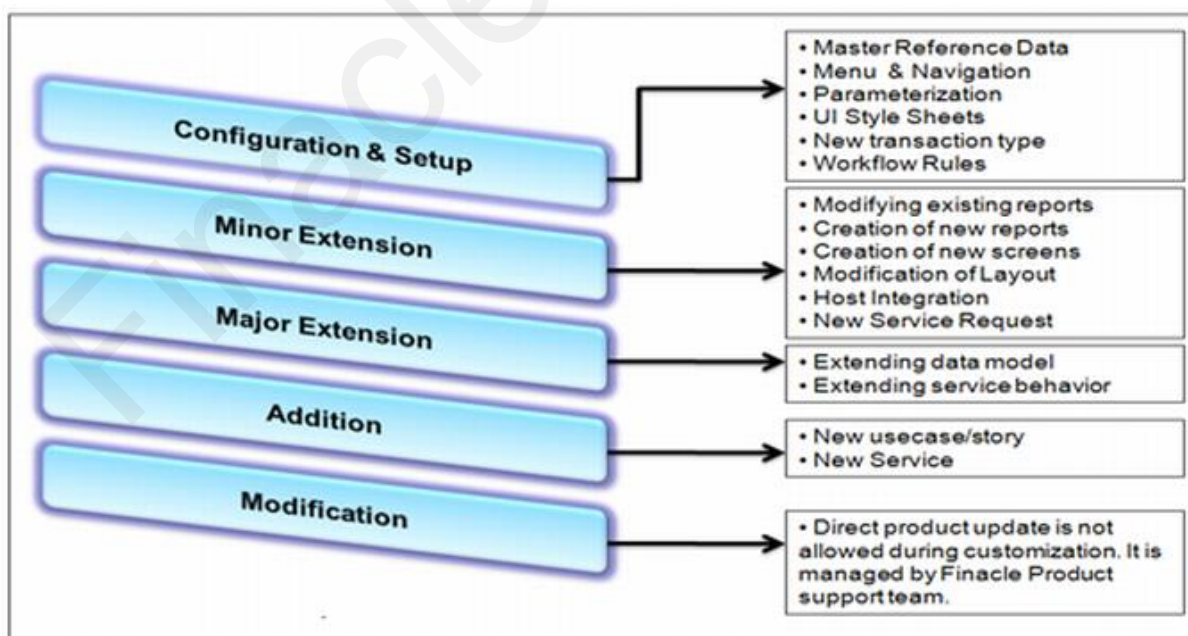
Ease of customization

Cost of customization will inversely grow with difficulty of customization. To control this, Customization Infrastructure enforces customization through configuration files wherever possible. The Finacle eBanking framework which is used by Customization Infrastructure abstracts out underlying platform from functionality developer. So developers can do customization easily without bothering too much about J2EE specifications.

Different levels of customization and their implications

There are different ways to execute customization on the base product. Each one of them has different impact on cost, maintainability, upgradability and complexity. This sections section describes all types of customizations and their impact.

All customization requirements in e-Banking can be classified into five types as shown in the image below:



As shown in the picture, there are grossly five types of customization done in e-Banking as per Bank's requirement. Below sections describe details of those types:

Configuration & Setup

Most of product functionalities delivered out-of-box are backed by configuration parameters and configuration files so that they can be tuned as per client's need just by assigning appropriate values to the parameters or changing the configuration files. Some examples of such customizations are:

- Setting up Menu Option & Menu tree
- Changing Navigation Logic
- Parameterization
- UI Style Sheets
- New transaction types
- Workflow rules
- Changing Backend host's IP/PORT
- Changing DB Server details
- Enabling/Disabling Audit
- Changing access control rules

There are many other things that can be customized by changing configuration files & parameters only. Refer to Product Release notes for complete details.

Minor Extensions

Extensions are customizations which needs code change. Minor Extensions are those customizations whose major part is configuration change, but it requires small code changes also. Some example of such extensions are-

- Modifying existing reports
- Creation of new reports
- Creation of new screens
- Modification of layout
- Host integration logic
- New service requests

There are many other things that falls in Minor Extension category. Refer to Product Release notes for complete details.

Major Extensions

Major extensions are those that mainly require code change. Configuration change may or may not be involved. This usually happens when we need to change data model of product service or we need to extend behavior of a service. Some example of such extensions are-

- Adding a new criteria field on inquiry screen/services
- Adding a new column on inquiry result/service response

Refer to Product Release notes for complete details on a list of major extension points.

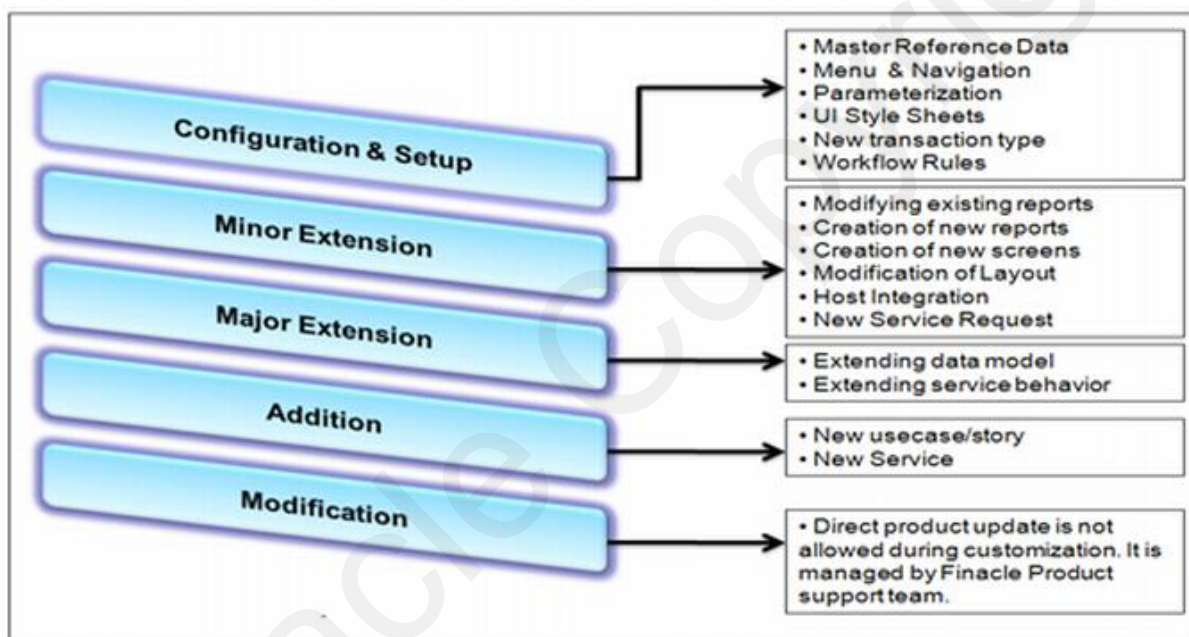
Addition

Sometimes one may need to add a completely new functionality which is not present in product. There may be different factors (like geography, constantly evolving market etc.) which may trigger such requirements. e-Banking supports writing a completely new functionality during customization. The Finacle eBanking is made available to implementation teams for such development.

Modification

Customization Framework doesn't allow direct modification of product sources to avoid branch lines which take off track routes from product roadmap. A modification should be worked out in conjunction with the product managers so that they can be supported in future releases.

All these five different customization choices, have different impacts on cost, upgradability, maintenance etc. Below picture depicts the same:



As displayed in the above picture, cost, complexity, maintenance effort, up gradation effort, technical skills will be least in configuration & setup based customization and would be highest in modification based customization.

The next set of sections focus on different product layers at which customizations are possible. All these are considered as spread across minor and major enhancements, without impacting product code base.

Achieving Customization in Type System Layer

Type System layer enables eBanking application to be strongly typed with respect to its domain. A type represents a logical unit of data along with operations and constraints applicable to it. Using type system framework all data structures, business entities are defined as one of the registered types in the system.

The platform provides a skeleton structure for all types using interfaces and abstracts classes. The constraints are validated when an attempt is made to instantiate or set a value to the type. All types existing in Type System are classified as mentioned below-

Basic or primitive types

- Numeric types like int, long, float and double.
- Char and String types.
- A special type for Amount – FEBAAmount.
- A special type for Date – FEBADate.
- A special type for Binary data - FEBABinary

List types

- An array list type - FEBAArrayList
- A FEBAHashMap type – which provides both key and index based access to its contents. This type can only hold a list of value objects.

Value object

- A composition of basic, list and other value objects.
- A simple composition of basic types which represent a reference to a database table, i.e. a data structure which has a foreign key but may have some more fields in the foreign table that accompany the key every time. These value objects are referred to as entity references.

FEBAHashMap

A simple wrapper over Hashmap which holds only objects of above FEBA types.

Creating new custom types

Customization infrastructure allows creation of new types (primitives as well as valueobjects). The type system defines a grammar vide an XSD. A new type can be created following the type definition schema.

Once the XML entries are made, a type generator tool is invoked to generate the type by using the spec.

Some considerations while adding a type to the system:

- The naming convention should be followed properly.
- Name of the new type cannot be same as any previous type.

- All new types generated during customization shall go to a registry file called CustomTypesCatalogue.xml. Product registry file viz. TypesCatalogue.xml should not get modified with new custom types. Type generator tool will take care of this.
- Product provided tool should be used for generation of types into classes.

How do I create a new custom Primitive?

To create a custom primitive, the tool setup should point to customization area. All the new custom primitives should be prefixed with “Custom” otherwise the tool will throw an error saying that the primitives should be prefixed with “Custom”. The grammar of custom specs is same as that of the product spec.

Code Alert:

```
<Primitive>

  <PackageName>com.custom.futurebank.types.primitives</PackageName>

  <ClassName>CustomMailID</ClassName>

  <Attributes>

    <Length>16</Length>

    <Justification>LEFT_JUSTIFIED</Justification>

    <CaseType>MIXED_CASE</CaseType>

    <ContentType>ALPHA_NUMERIC</ContentType>

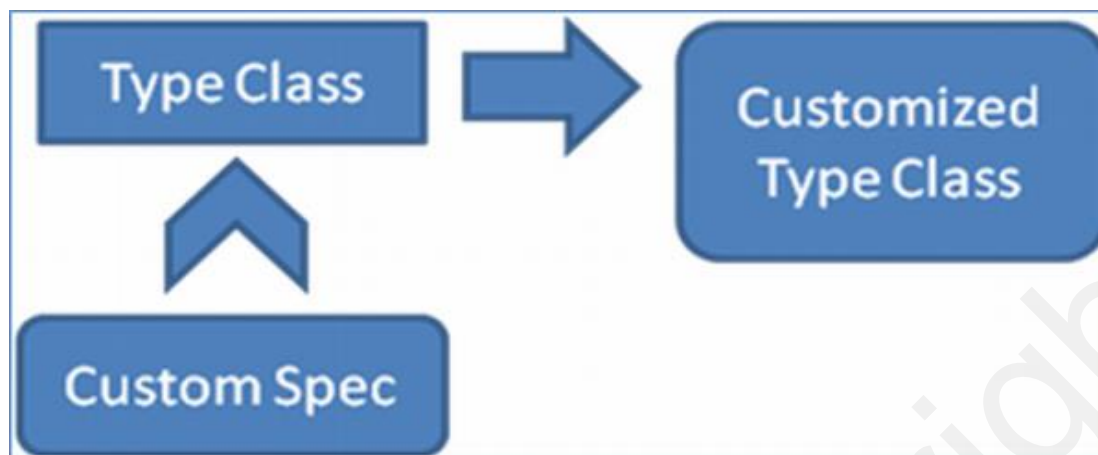
  </Attributes>

</Primitive>
```

Above code snippet is an example of an entry in CustomStringSpec.xml which demonstrates the grammar of the custom string spec.

Customizing existing primitive type

Customization of primitive types is supported by providing additional specification at run time. We make Custom Spec containing the types which are to be generated or used for customization.



The customization specification (custom spec) follows the same grammar as product (XML). The custom spec needs to be subset of product. Ex: FEBACharCustomSpec.xml, FEBAStringCustomSpec.xml etc.

If we want to overwrite any product type we just need to modify the type and need to place the spec in Data\type\extension.

For Example:

Data

|__types

|__extension

|__FEBACharCustomSpec.xml

|__FEBAStringCustomSpec.xml

|__FEBAFLOATCustomSpec.xml

|__FEBALongCustomSpec.xml

|__FEBADoubleCustomSpec.xml

|__FEBAIIntCustomSpec.xml

The abstract FEBA product types supported for customization are FEBAAString, FEBAAALong, FEBAAFloat, FEBAADouble, FEBAIInt & FEBAAALong.

The field values which are supported for customization based upon different types are mentioned in the table below:

Type	Supported
FEBA String Type	<ul style="list-style-type: none"> ▪ Default Value ▪ Length ▪ Justification ▪ Case Type ▪ Content Type ▪ Debug Flag

FEBA Char Type	<ul style="list-style-type: none"> ▪ Default Value ▪ Valid Char Set ▪ Debug Flag
FEBA Long Type	<ul style="list-style-type: none"> ▪ Default Value ▪ Min Value ▪ Max Value ▪ Debug Flag
FEBA Int Type	<ul style="list-style-type: none"> ▪ Default Value ▪ Min Value ▪ Max Value ▪ Debug Flag
FEBA Float Type	<ul style="list-style-type: none"> ▪ Default Value ▪ Min Value ▪ Max Value ▪ Debug Flag
FEBA Double Type	<ul style="list-style-type: none"> ▪ Default Value ▪ Min Value ▪ Max Value ▪ Debug Flag

What all customizations are possible in existing Product primitives?

The customization in existing product primitives can be done for the following primitives only:

- FEBAString
- FEBAChar
- FEBAFloat
- FEBADouble
- FEBAInt
- FEBALong

The specs provided in the custom XML for FEBA type should comply and be subset of the product specs except in case of FEBACChar.

For example:

Object Name	Type	Attribute	Product	Custom	Allowed
MailSubject	FEBAString	Type	Alphanumeric	Numeric	Yes
ProfileID	FEBAString	Length	32	50	No

YNFlag	FEBAACChar	Valid Set	YN	XY	No
YNFlag	FEBAACChar	Valid Set	YN	YNZX	Yes
YNFlag	FEBAACChar	Valid Set	YN	Y	No

In FEBAACChar the valid set attribute can be extended to allow for more characters to be added to valid set but we cannot remove the characters in the valid set entirely.

How to customize an existing Product primitive?

To customize an existing product primitive, you need to create a folder \WorkingDirectory\data\ types\extension and place the required PrimitiveCustomSpec(FEBACCharCustomSpec.xml etc..) xml under it. At runtime, this xml will be picked and will over write the concerned type object attributes mentioned in the spec.

Existing product primitive (FEBADoubleCustomSpec.xml):

```
<Primitive>

  <PackageName>com.infosys.e-Banking.types.primitives</PackageName>

  <ClassName>ExchangeRate</ClassName>

  <Attributes>

    <MinValue>1</MinValue>

    <DefaultValue>38.00000</DefaultValue>

  </Attributes>

</Primitive>
```

Customizing an existing product primitive (FEBADoubleCustomSpec.xml):

```
<Primitive>

  <PackageName>com.infosys.e-Banking.types.primitives</PackageName>

  <ClassName>ExchangeRate</ClassName>

  <Attributes>

    <MinValue>4</MinValue>

    <DefaultValue>38.00000</DefaultValue>

  </Attributes>

</Primitive>
```

Note:

The grammar of custom spec (FEBADoubleCustomSpec.xml, FEBACCharCustomSpec.xml etc..) will remain same as that of the product spec.

Creating a new Value Object

For creation of a new customVO, the tools setup has to point to customization sysarea. The name of the VO has to be prefixed with "Custom" otherwise the tool will throw an exception stating that the VO should be prefixed with "Custom" and also product names are not allowed as they already exists. The new customVO is capable enough to support existing product primitives along with newly created custom primitives.

Customizing existing value objects

Value Objects can be extended. This can be used to pass across additional custom data in a product service.

The VO extension mechanism is achieved through composition and not inheritance. Every FEBA VO contains a field called extension VO by default. Content of this field is injected from outside by means of an xml called VOExtensionRegistry.xml. This kind of Inversion-of-Control helps to achieve product VO extension without changing product binary.

To extend a product VO, following steps are to be executed-

- Create a new custom VO(EBFA Value object) using Type Generation Tool
- Make an entry in data/VOExtensionRegistry.xml to attach the custom VO to product VO. This configuration file acts as a glue to inject new custom VO

Find below example of an entry in VOExtensionRegistry.xml:

Example:

```
<valueobject name="StateCodeVO">
    <property name="extensionVO" value="CustomStateCodeVO" instantiate="YES"/>
</valueobject>
```

In the example above `StateCodeVO` is the productVO and `CustomStateCodeVO` is the extendedVO.

Extending a Product Value Object

Any productVO can be extended with any other VO provided the extendedVO has already been created. For extending a productVO, a configuration has to be done in “VOExtensionRegistry.xml” present in \WorkingDirectory\data\ folder. The configuration has to be done in the following ways:

```
<valueobject name="MailDetailsVO">
    <property name="extentionVO" value="CustomMailDetailsVO" instantiate="YES" />
</valueobject>
```

In the above code snippet, "MailDetailsVO" is the existing productVO which is extended with "CustomMailDetailsVO" (which has to be created first with customization tool setup). The attribute "instantiate" of <property> Tag is responsible for creating the instance of "CustomMailDetailsVO" for its usage in "MailDetailsVO".

The value of the property "instantiate" should be "YES" for creation of the extendedVO's instance for it's usage in the productVO. Ideally when a productVO is extended by a customVO, the new customVO created should be of the same entity as that of productVO i.e (expected behaviour is that a customVO

will encapsulate field/fields which are in relation and form an extension to the productVO rather than an accountsVO getting extended with a mailsVo).

When the productVO travels to the transaction layer, getExtensionVO() method can be invoked on it to get the instance of the extendedVO for further processing.

Customization in Formsgroup Layer

Formsgroup [FG] is a component which handles a group of related actions initiated in an UI/ Service. It can also be defined as a mechanism through which one defines navigation and transaction across business data. Essentially, it is a logical set of behavioral actions on a related set of data, generally modeled as domain objects.

This layer generally requires a lot of customization during implementation. For example, during implementations, screen or service navigation or orchestration is going to change or additional step of data transaction might need to occur between system and client system

So to accommodate all such requirements, below mentioned customization options are given:

Customizing Product Formsgroup

Product forms group should not be modified directly, but, one can easily extend the product forms group and add additional logic during customization.

As part of product forms group customization, below extensions are possible. Forms Group XSD specification knowledge is assumed here:

Attributes	Supported Customization
Event	<ul style="list-style-type: none"> Add a custom event Pre and Post events can be added based up on event requirement Over write a product event (Not recommended)
Mapping	<ul style="list-style-type: none"> Add a custom mapping Extension of product mappings
UI Controls	<ul style="list-style-type: none"> Add new UI controls Overwrite product UI controls
Fields	<ul style="list-style-type: none"> Add custom fields Use product fields
Global Variable	<ul style="list-style-type: none"> Add new global variables Use Product global variables

Extended FGs are placed in a “data/formsgroup/extension” folder. When user request comes, Formsgroup factory first loads the product formsgroup. Then it checks “extension” folder for custom extension formsgroup of the product formsgroup. If found it loads the extended FG and merges with product FG. After merge, both the FGs exist as a single entity. Extension FG can be independently compiled. It becomes aware of the product FG only at runtime.

Extended FG files have a naming convention so that they can be easily identified. As per this convention, all extended FG files must end with “_Extension”, for example MailFG_Extension.xml, AccountListFG_Extension.xml etc.

What all I can extend from a product FG?

- Form data (a new Form Field can be added).
- Declaration (a new global variable can be added).

- Can write a new event.
- Can overwrite an existing event.
- Can write a Pre-event and Post-event of an event of the productFG.
- Can overwrite existing UI control.
- Can write a new UI control.
- Can extend an existing mapping.
- Can write a new mapping.

How do I extend a Mapping sections in an extended Formsgroup?

In an extended Formsgroup we can extend a mapping just by adding the new field mapping for the particular enhancement we want to have. In this we have to change the ID which is given in the mapping section. Id name is same as that given in Product FG with a suffix “_CUSTOM” added to it.

Mapping has two types VO and LIST.

Example:

VO Mapping type:

```
<mapping type="VO" id="StateCodeCriteriaMapping_CUSTOM" datatype="StateCodeCacheVO">
<doc>Mapping for fields entered in the criteria screen</doc>
<field-mappings>
<field-mapping form-field="STATE_DESC"
vo-field="criteria.extensionVO.stateDescriptionCustom" />
</field-mappings>
</mapping>
```

LIST Mapping type:

```
<mapping type="LIST" id="StateCodeListMapping_CUSTOM" datatype="StateCodeCacheVO"
recordType="StateCodeVO">
<doc> Mapping of the results of a state code list query </doc>
<field-mappings>
<field-mapping form-field="LANG_ID_ARRAY[i]"
vo-field="resultList[i].extensionVO.languageID" />
</field-mappings>
```

```
</mapping>
```

In the above examples we have used a term “extensionVO” in vo-field under “field-mapping” tag. Here extensionVO is the name given to our customVO in which we have declared our new formfields and their fieldtypes. This extensionVO is declared in “VOExtensionRegistry.xml” file kept in data folder of WorkingDirectory. The declaration done in this file is as follows:

```
<extensions>

<valueobject name="StateCodeVO">

<property name="extensionVO" value="CustomStateCodeVO" instantiate="YES"/>

</valueobject>

</extensions>
```

How do I use product FORM FIELDS and Global variables in Extended FG?

We can use product formfields and Global variables in extended FG by firstly declaring them through “extern” tag.

Example:

The fields which are been declared in product FG can be called in extended FG just by writing their reference name and type as “field” and similarly the global variables can be declared by just writing type as “Global”.

```
<extern>

<reference name="TXN_USER_CATEGORY" type="Field" />

<reference name="TXN_CATEGORY_IMAGE" type="Field" />

<reference name="enquiryVO" type="Global" />

</extern>
```

So these fields mentioned above, under “extern” tag were declared in Product FG as Global variables and Form fields.

How do I write PRE or POST event hooks in extended FG?

If we want to call some events in our extension FG (MailFG_Extension.xml) which are being used in product FG (MailFG.xml), then we can write these events as PRE and POST accordingly as per the order of their calling.

Example:

Let us take “SEND_COMPOSED” be an event in Custom FG. So if we want to have PRE or POST event of this then we can use the following tags:

```
<events>

<event name="PRE_SEND_COMPOSED">
```

```

<doc>This is implementing pre Send Compose hook.</doc>

<audit-desc>Pre Send Compose Event</audit-desc>

    <event-handling-block>

        <step>FGI::SOP("In pre event")</step>

    </event-handling-block>

</event>

</events>

```

OR

```

<events>

<event name="POST_SEND_COMPOSED">

    <doc>This is implementing post Send Compose hook.</doc>

    <audit-desc>Post Send Compose Event</audit-desc>

        <event-handling-block>

            <step>FGI::SOP("In post event")</step>

        </event-handling-block>

    </event>

</events>

```

How do I overwrite LOAD event?

LOAD event of product formsgroup can't be overwritten by it's custom extension.

If custom extension doesn't want to execute the LOADing code of product FG, it needs to write a PRE event hook of LOAD and in that hook it needs use FGL::ENDEVENT. This FGL ends execution of the hook event and as well as its product event. And the POST hook event also.

Code Alert:

```

<events>

<event name="PRE_LOAD">

    <doc>This is implementing pre load hook.</doc>

    <audit-desc>Pre Load Event</audit-desc>

    <declaration>

        <variable name="customCallMode" datatype="FEBAUnboundInt"/>

```

```

</declaration>

<event-handling-block>

    <step>FGI::SetViewId("customMailDashboard");</step>

    <step>FGL::ENDEVENT</step>

    <step>FGI::SOP("Is this line reachable")</step>

</event-handling-block>

</event>

</events>

```

In the above code, one has coded PRE event hook of LOAD event. It is doing only 2 things. First, it is setting up the view ID to customized dashboard. Second, declaring end of event.

Because of the second step, FG framework would force end of execution of LOAD event and control would return from here. Even the SOP on third step won't be executed.

Note:

If EVENT_X called EVENT_Y and FGL::ENDEVENT was executed in EVENT_Y, it would terminate EVENT_Y and control will go back to EVENT_X. It will resume its execution from the step that made call to EVENT_Y.

New Custom Formsgroup

During customization, one can write new formsgroup as well, if required. The FG compiler tool provided would support new custom FGs. These new custom FGs have to follow same grammar as that of product FGs. But it must follow the naming convention for new custom FGs. The convention is that it must start with keyword "Custom".

New custom FGs would be residing in the folder "data/formsgroup/custom" folder.

Extension facility is provided for custom FGs as well. This would be useful in case of multi branding.

How do I have a new call mode in customization?

A new parameter has to be added in VMOT table for the column "FG_LINK_URL" known as "Custom_Call_Mode". For a retail user, if the user is linked with "RMAIL" the url that is configured in VMOT will be:

```
FORMSGROUP_ID=MailModuleFG&Call_Mode=0&CONTROL_ID=MAIL
```

If we want to invoke a new event on the click of the the above url as part of customization than the link has to be modified in the following ways:

```
FORMSGROUP_ID=MailModuleFG&Call_Mode=0&CONTROL_ID=MAIL&Custom_Call_Mode=1
```

As soon as the url is clicked the PRE_EVENT of the LOAD event in the ExtendedFG will have the access on custom call mode and the required processing can be done based upon the custom call mode.

Code Alert:

```
<step>FGI::CustomCallMode(customCallMode);</step>
```

A new FGI “CustomCallMode” is written which extracts the value from the url and stores it in a variable for further processing.

How can I add new reserved field in customization?

As part of customization, We can add the following types of Reserved fields:

- PageLevelTransientField
- PersistentReservedFields
- ReservedFields

For using a new Reserved field we have to first write a new custom reserve field registry which is configured in AppConfig.xml against the property “CUSTOM_RESERVEFIELD_REGISTRY” and this registry class need to implement

ICustomReservedFieldsRegistry interface.

Currently this property is configured with “DummyReservedFieldRegistry” value. This value is the fully qualified path of the CustomReservedFieldsRegistry class. The property is configured in the following way:

```
<Param name = " CUSTOM_RESERVEFIELD_REGISTRY" value = " com.infosys.e-
Banking.common.formsgroup.DummyReservedFieldRegistry"/>
```

Assume we want to add reserved field called NAVIGATION_OPTION__ .

Then we need define our reserved field in our new Registry class as a string array like one given below:

```
private static String[] NAVIGATION_OPTION_DETAILS=new
String[]{"NAVIGATION_OPTION__","BASIC","FEBAUnboundString","com.infosys.febe.framework.types.primitiv
es.FEBAUnboundString"}
```

and if you want make this field as normal Reservefield ,then loadReservedFields(List reservedFieldsList) method needs to be implemented and this field should be added in to the List like one given below:

```
public void loadReservedFields(List reservedFieldsList){

    reservedFieldsList.add(NAVIGATION_OPTION_DETAILS);

}
```

Like that if you want to add any field as PersistentReservedFields then loadPersistentReservedFields(List persistentReservedFields) method needs to be implemented And for PageLevelTransientField

, loadPageLevelTransientField(List pageLevelTransientFields) method needs to be implemented.

If you want to get the value of any custom reserved field in FG then you can use the following line of code.

```
<step>FGI::FormField("GET_VALUE",navigationOption," NAVIGATION_OPTION__")</step>
```

But suppose this step is inside an event which is having “isTxnPwdValidationReqd” flag as Y then you need to make entry of this reserved field in loadPersistentReservedFields(List persistentReservedFields) method of CustomReservedFieldRegistry.java else you would not be able to retrieve its value from cache.

Code Alert:


```
private static void loadPersistantReservedFields(List persistantReservedFields) {

    persistantReservedFields.add(NAVIGATION_OPTION);

}
```

Customization of Formsgroup Instruction

Formsgroup Instruction (commonly known as FGI) can also be customized to add new custom FGIs. If new FGIs are added, a mapping of FGI name to Interpreter class has to be provided. The mapping is loaded from custom file. Name of that file can be injected through a property in AppConfig.xml.

This kind of Inversion of Control strategy makes the FG Grammar extensible and thus it can grow up overtime without impacting existing grammar.

How do I write a new Custom Formsgroup Instruction (FGI)

For using a new custom FGI we have to first write a customFGI and this FGI is called through a Loader say "CustomFGILoader" which should be kept into the path as mentioned in AppConfig.xml file kept in data folder.

The path mentioned in the AppConfig file is as follows:

```
<Param name = "FGI_LOADER_CUSTOMIZATION" value=
    "com.infosys.feba.framework.formsgroup.interpretor.fgi.DummyFGILoader" />
```

This is just a dummy FGI path we can make the changes accordingly and change this path to:

```
<Param name = "FGI_LOADER_CUSTOMIZATION" value=
    "com.futurebank.feba.framework.formsgroup.interpretor.fgi.CustomFGILoader" />
```

So now we will have to create a new file as "CustomFGILoader" under this path, which will contain the name of the customFGI created. For this, a constant is to be used which is already defined in "CustomFGIConstants" file. CustomFGIConstants file contains declaration of the constant value(s) which are used in "CustomFGILoader". This constant file will be under the path

```
"com.futurebank.feba.custom.common.formsgroup.fgi"
```

Example:

A constant is declared in the following fashion in "CustomFGIConstants" file:

```
public class CustomFGIConstants {

    public static final String SOP = "SOP";

    private CustomFGIConstants(){}

}
```

And this is how it is called in "CustomFGILoader" file:

```
public class CustomFGILoader implements IFGILoader {
```

```
public void load(Map<String, IFGIStepInterpreter> fgiInterpreterMap) {  
  
    fgiInterpreterMap.put(CustomFGIConstants.SOP , new SOPInterpreter());  
  
}  
  
}
```

Now in our Custom FG, to call the concerned FGI, the constant's name has to be the same as mentioned in "CustomFGIConstants" file. Like in this case, the constant value is 'SOP' and is used thus in the custom FG:

```
<event-handling-block>  
  
<step>FGI::SOP("Hello. U are using Custom FGI.");</step>  
  
</event-handling-block>
```

Important:

Please note that while trying to generate an FG spec having a new custom FGI, the FG compiler tool should also refer to the files modified for new FGI. For example, FG compiler tool refers to a class say, CustomFGILoader. Now, while writing a new FG, this file has been modified. Hence this class needs to be present in the classpath of the FG compiler tool. Otherwise, it will give error saying that the newCustomFGI is not a valid FGI and the FG will fail to generate. Follow this approach for other similar errors occurring during generation of new FGI.

Customization in Service Formsgroup Layer

Service Formsgroup (aka SFG) layer, an extension of Formsgroup layer, is designed to compose and expose internal service(s) with minimal effort and cost in a technology agnostic way. It can be seen as a “CONTROLLER” layer which is responsible for producing the “MODEL” which framework then automatically converts to a data xml and exposes as a web service.

The core functionality of this layer is to accept all input fields coming in the request, call different services and finally return required output fields. By virtue of being an extension of Formsgroup layer, it leverages all the advantages provided by Formsgroup. The advantages are-

- Configuration driven – This layer is completely driven by XML configuration files.
- Customizability – It provides different customization points required to tailor the solution as per Bank's requirement.
- Upgradability – It completely abstracts product code from custom extension code, thus providing easy and low impact upgradability
- Audit – Incoming fields are sent to Audit Framework. Bank can decide what field to Audit via an administrative application.
- Access Control – The layer checks if user has access to requested functionality. Bank can choose user level access control via administrative application.
- Service contract generation – The framework can generate WSDL(WS-I Basic Profile 1 compatible) of the service and XSD of the payload automatically from the configuration files. This helps the developer to concentrate in service orchestration logic only and thus saves effort and cost.

In the below section, we'll discuss customizability of SFG in details.

SFG has same grammar as that of Formsgroup, but it contains only two events LOAD and PROCESS. The logic for invoking the service(s) which is(are) required to be exposed is placed under these events.

This layer can be customized during implementation to support customer specific requirements. It supports below options for customization –

Customization	Description
Change in input fields	<ul style="list-style-type: none"> ▪ Add a new input field ▪ Remove an existing input field ▪ Split an existing field to two ▪ Join two fields to one single field
Change in output fields	<ul style="list-style-type: none"> ▪ Add a new output field ▪ Remove an existing output field ▪ Split an existing field to two ▪ Join two fields to one single field
Service orchestration	<ul style="list-style-type: none"> ▪ It supports complex orchestration of fine grained services

Pre Process	<ul style="list-style-type: none"> One can extend product services to add pre processing logic to what product service does. This will help to achieve custom extensions with minimal coding.
Post Process	<ul style="list-style-type: none"> One can extend product services to add post processing logic to what product service does. This will help to achieve custom extensions with minimal coding.

To accommodate such customization requirements, one can customize below constructs of SFG specification-

Customizing Product Service Formsgroup

Customization Infrastructure doesn't support modifying the product service formsgroup directly. Service Formsgroup specification file would be released to customization team for direct modification. But, one can easily extend the product service formsgroup and add additional logic during customization.

As part of product service formsgroup customization, below extensions are possible

Attributes	Supported Customization
Mapping	<ul style="list-style-type: none"> Add a custom mapping Extension of product mappings
UI Controls	<ul style="list-style-type: none"> Add new UI controls as in Service FG
Fields	<ul style="list-style-type: none"> Add custom fields Use product fields
Global Variable	<ul style="list-style-type: none"> Add new global variables Use product global variables

Extended SFGs are placed in a "data/formsgroup/extension" folder. When the request receives, Formsgroup factory first loads the product service formsgroup. Then it checks "extension" folder for custom extension formsgroup of the product service formsgroup. If found it loads the extended SFG and merges with product SFG. After merge, both the SFGs exist as a single entity. Extension SFG can be independently compiled. It becomes aware of the product SFG only at runtime.

Extended SFG files have a naming convention so that they can be easily identified. As per this convention, all extended SFG files must end with "_Extension", for example RegisteredStandardPayeeSFG_Extension.xml, AccountSummarySFG_Extension.xml etc.

As part of product service formsgroup customization, below extensions are not possible/supported

What all I can extend from a product SFG?

- Form data (a new Form Field can be added).
- Declaration (a new global variable can be added).
- Can write a new UI control.
- Can extend an existing mapping.

- Can write a new mapping.

All the grammar of Formsgroup will be applicable for Service forms group in the above points

New Custom Service Formsgroup

During customization, one can write new service formsgroup as well, if required. The provided FG compiler tool would support new custom SFGs. These new custom SFGs has to follow same grammar as that of product SFGs. But it must follow the naming convention for new custom SFGs. The convention is that it must start with keyword "Custom".

New custom SFGs would be residing in the folder "data/formsgroup/custom" folder.

The eBanking workbench can be used to create a new service forms group which provides for a forms group template to be filled in an editor and invoke a generator tool for its validation and generation into java classes.

What all I should perform for invoking custom SFG?

- Write custom Service Forms group

Customized Operative Account Details Service.

- Identify a service ID for the new service

CROADT

- Make an entries in the VMOT and VMNP tables for mapping the newly identified service id and menu profile code of the user

Insert into VMOT

```
(DB_TS, BANK_ID, DEL_FLG, MNU_ID, MNU_TXT, FG_LINK_URL, PROMOTION_FLAG,
USER_ACCEPTANCE_REQUIRED, AC_TYPE, TYPE_OF_ACTIVITY, RM_PLUS_ENABLED, R_CRE_ID,
R_CRE_TIME, R_MOD_ID, R_MOD_TIME, MENU_TYPE)
```

```
Values (1, 'DBS', 'N', 'CROADT', 'Customized Operative Accounts',
'FORMSGROUP_ID=CustomizedAccountDetailsSFG', 'N', 'N', 'ALL', 'NFIN', 'Y', 'setup', TO_DATE('08/27/2010
12:17:51', 'MM/DD/YYYY HH24:MI:SS'), 'setup', TO_DATE('08/27/2010 12:17:51', 'MM/DD/YYYY HH24:MI:SS'), 'S');
```

Insert into VMNP

```
(DB_TS, BANK_ID, DEL_FLG, MNU_ID, MNU_PRF_CD, PARENT_MNU_ID, MNU_ID_TYPE, MNU_ID_ORDER,
IS_PART_OF_MENU_TREE, R_MOD_ID, R_MOD_TIME, R_CRE_ID, R_CRE_TIME)
```

Values

```
(1, 'DBS', 'N', 'CROADT', 'RUSER', 'NA', 'PARENT', 1, 'N', 'MANJUADM1', TO_DATE('08/26/2010 17:56:22',
'MM/DD/YYYY HH24:MI:SS'), 'MANJUADM1', TO_DATE('08/26/2010 17:56:22', 'MM/DD/YYYY HH24:MI:SS'));
```

Insert into VMNP

```
(DB_TS, BANK_ID, DEL_FLG, MNU_ID, MNU_PRF_CD, PARENT_MNU_ID, MNU_ID_TYPE, MNU_ID_ORDER,
IS_PART_OF_MENU_TREE, R_MOD_ID, R_MOD_TIME, R_CRE_ID, R_CRE_TIME)
```

Values

```
(1, 'DBS', 'N', 'CROADT', 'CUSER', '!', 'PARENT', 1, 'N', 'setup', TO_DATE('08/27/2010 11:33:03', 'MM/DD/YYYY  
HH24:MI:SS'), 'setup', TO_DATE('08/27/2010 11:33:03', 'MM/DD/YYYY HH24:MI:SS'));
```

Customization in Service Layer

Customization Infrastructure supports writing new services during customization. To write new services one has to write the service spec whose grammar is same as that of product services. These specs are to be kept in “data/services” folder. When a request comes for a service, corresponding spec is picked up from the path, parsed and a met info object is formed and loaded in memory. Services framework reads the Meta information and makes necessary service calls through client stub-service stub mechanism. While doing so, it also looks up for transmission mode chosen. It can be either EJB or POJO. This mode is controlled by a property in bootstrap configuration file. Thus service calls can be switched between EJB and POJO while doing the setup.

Finacle eBanking workbench provides for an XML on the service template. The workbench facilitates service creation.

How do I write a new custom service?

Unlike product, custom service spec will not undergo as a tool input for generation of client and server stubs along with service class. At runtime, when the custom service will be invoked, EBClientServerFactory will look for the custom service in the classpath and create metainfo of it and cache it. The custom service can also be contained in a jar and that jar can be present in the \WebContent\WEB-INF\lib folder or it can be imported in the server settings.

Note:

In short, the custom service spec should be present in the classpath for runtime execution of the custom service.

The grammar of the custom spec is same as that of a product spec. Custom service is XSD driven hence the custom service spec should adhere to “ServiceSchema.xsd” otherwise service exception will be thrown at runtime execution of the custom service.

How licensing works in custom services?

A new property has been added to the custom service spec which would identify whether the service invoked is a product service or a custom service.

Code Alert:

```
<service-properties>

  <param>

    <name>CUSTOM_SERVICE_INDICATOR</name>

    <value>Y</value>

  </param>

</service-properties>
```

Based upon the value of the above property as ‘Y’ or ‘N’ license check is disabled/enabled for the service invoked.

Note:

For disabling the license check for the custom service, its spec should contain the above property with its value as ‘Y’.

How audit works in custom services?

Since the Grammar of the custom service spec is same as that of product service, Audit framework works the same way as in the case of the product service.

Customization in Transaction Layer

All transactions have to follow one of five pre-defined transaction patterns. These patterns are actually abstract template classes that dictate the transaction process and expose couple of abstract methods that are implemented by different implementation classes. During customization one may need to intrude into the transaction process flow for a particular use case to add additional processing as required by Bank.

For this purpose, hooks are provided in each step of the transaction process defined in the abstract pattern. One can decide the hook class to be called for each hook point during customization. This is implemented by following Inversion-of-Control design pattern. There is a configuration file provided in where one can configure that for a given transaction IMPL class and given hook point, what hook class to be called, for example, one can say that “For “MailServiceComposeImpl” class and “preProcess” hook point, I want to invoke MailComposePreProcessor.java file which will do some additional processing before product defined processing happens”

Similar to “preProcess” as explained in the example, there are couple of hooks provided so that customization engineer can step into product process and add additional processing over and above product’s processing.

Below table describes couple of such hooks:

Attributes	Supported Customization
Custom_INITIALIZER	Initialize the custom objects
Custom Validate	Validate the custom fields passed
Custom Preprocess	Populate the object that needs to be passed to HIF and DAO Layer.
Custom Postprocess	Message the product /custom data
Custom Postprocess handler	Modify data before being passed back to the service layer
Custom Associate Query	Associate the value of new parameters added.
Custom Post Query	Manipulate data of each record.

Please refer to Customization Release Notes to get complete details of each hook for each transaction pattern.

What all hooks are exposed in transaction patterns?

Refer to Customization Release Note.

How do I write hook for a transaction impl?

For writing a hook for a transaction impl, first make an entry in custom.xml for that hook.

Code Alert:

```
<customVariationIdentifier id="MailCountServiceFetchCustomFoldersImpl">
```

```
<customVariationHook hookPoint="CustomPreProcess">
```

```

<customVariationHookClassName>

    com.infosys.e-Banking.custom.TransactionHelperHook

</customVariationHookClassName>

</customVariationHook>

</customVariationIdentifier>

```

Custom.xml has three elements. First element is customVariationIdentifier and value of id is the class name from where hook is exposed, second element is customVariationHook whose attribute is hookPoint id, the value of hookPoint id attribute can be any hook point exposed by the product class and third element is CustomVariationHookClassName. CustomVariationHookClassName is the actual hook class name which is implemented during customization.

How do I show success and error messages?

a. Success Messages-

Success message is shown by setting BusinessInfoVO with appropriate values.

1. Create an instance of BusinessInfoVO.
2. Use its setter methods to set code (the error code), logMessage, displayMessage, and additional parameters if any. For customization, you will need to set the nameSpace to "CUST"

b. Error Messages-

Custom error to be thrown using following constructor of BusinessException. plsCustomErrorCode to be set to true for custom error.

```

private void createSingleBusinessException(

    boolean pIsCustomErrorCode,

    IContext pContext,

    String pIncidenceCode,

    String pAdditionalMessage,

    int pErrCode,

    String pErrorFieldTag,

    Throwable pCause,

    AdditionalParam... pParam) { .... }

```

Customization in TAO/TMO Layer

How do I generate TAO classes for a new table?

We need to create a TSPC for that table which contains details of the columns of that table. Then we generate the TAO using that TSPC spec through the “TAO generator” tool. During the generation, TAO, DAO, INFO and INFOKEY files are generated.

How do I generate TAOs in custom package?

A file viz, Tools_Custom.properties will be used to specify the paths for TAO, DAO, Info, InfoKey, etc. Give the required paths for each of them in this file (Example: TAOPackage=com.cfs.e-Banking.tao).

1. In build setup, keep Tools_Custom.properties file in e-Banking-build\bin folder.
2. Similarly in local workspace, keep this file in the same directory where devrun bat file of tao generator exists (For example: e-Banking-Tool\bin)

Customization in DAL Layer

Customization Infrastructure doesn't allow direct modification of DAL specification files of product. It has instead provided an extension mechanism using which one can customize the query being fired to data base.

To avail this extension mechanism, one has to write the extension spec whose grammar is same as that of product spec and specifies only the part that is customized (where clause, select clause etc.). The naming convention for custom DAL spec is that its name will be same as product DAL spec but it will have "_custom" as suffix.

At the time of generation through the tool, we specify the name of product DAL spec only and not the custom DAL spec. While generating, the tool automatically checks whether there is any custom DAL spec in the same folder path of product spec, then it combines both and generates the DataBuilder and QueryBuilder files.



DAL Extension spec supports following:

Attributes	Supported Customization
Select Clause	Adding new fields in the select clause supported
Where Clause	Overwriting the where clause is supported For adding new field in the clause one needs to overwrite.
Order by Clause	Overwriting order by clause supported
Sub Query	The modification to sub-query is not supported.

What all I can do in DAL Layer?

- 1) Adding new fields in the select clause supported
- 2) Overwriting the where clause is supported. For adding new field in the clause one needs to over write
- 3) Overwriting the order by clause supported
- 4) Adding new tables in the from clause is supported.

How do I setup additional query parameters in QueryBuilder file?

The custom spec follows the same grammar as product. The customized DAL has to be saved as <ProductDAL>_Custom.xml in the same folder as the ProductDAL.xml spec file. The DAL generator should be run by specifying the input as the ProductDAL.xml.. The generated java files have to be placed in the correct path.

The package for the new java files can be mentioned in the custom DAL spec.

Code Alert:

```
<ObjectName>CPStateCodeListQuery</ObjectName>

<DataClassName>CPStateCodeListBuilder</DataClassName>

<DalPackage>com.futurebank.e-Banking.dbaccess.dal.querybuilder</DalPackage>

<DataPackage>com.futurebank.e-Banking.dbaccess.dal.databuilder</DataPackage>
```

Add new criteria field in the <CritList> section.

Code Alert:

```
<CritList>

  <CritFields sqlIdentifier="StateCodeListFetchQuery">
    <CritField>

      <name>stateDescriptionCustom</name>

      <column>STATE_DESC</column>          <datatype>com.futurebank.e-
      Banking.types.primitives.CustomStateDescription</datatype>

    </CritField>
  </CritFields>

</CritList>
```

Add condition in the <filter> section as a new <criterion condition= ">

```
</criterion>

<logical-operator>AND</logical-operator>

<criterion condition = "FEBATypesUtility.isNullOrBlank(stateDescriptionCustom)">

  <field>

    <STCD.STATE_DESC/>

  </field>

  <operator>=</operator>

  <value>

  <tablename></tablename>
```

```

<name> </name>

<variable>stateDescriptionCustom</variable>

</value>

</criterion>

```

Create a hook (.java file) which extends DefaultCustomQuery

Override the associateCustomQueryParameters() method for associating the additional criteria.

```

StateCodeVO codeVO = (StateCodeVO)objQueryCrit;

final CustomStateCodeVO customCriteriaVO = (CustomStateCodeVO) codeVO.getExtensionVO();

if (customCriteriaVO!=null){

    queryOperator.associate("stateDescriptionCustom", customCriteriaVO.getStateDescriptionCustom());

}

```

The entry of this custom hook is made in Custom.xml

```

<customVariationIdentifier id="StateCodeDetailsServiceFetchImpl">

<customVariationHook hookPoint="CustomAssociateQuery">

<customVariationHookClassName>com.futurebank.e-
Banking.CustomStateCodeAssociateQuery</customVariationHookClassName>

</customVariationHook>

</customVariationIdentifier>

```

How do I retrieve additional fields in DataBuilder file?

Declare new data field in the <GlobalFields> section of ProductDal_Custom.xml.

```

<GlobalFields>

<field>

<function></function>

<tablename>STCD</tablename>

<name>LANG_ID</name>

<mask></mask>

</field>

</GlobalFields>

```

Add it in the <addedFields> section.

```
<Fields>

    <addedFields>

        <STCD.LANG_ID/>

    </addedFields>

</Fields>
```

Declare it in the <DataBuilderList> section.

Define the fully qualified extensionVO name in the

```
vo-classname=" com.futurebank.e-Banking.types.valueobjects.CustomStateCodeVO".
```

Code Alert:

```
<DataBuilderList>

    <DataBuilder identifiers="StateCodeListBuilder">

        <ResultSetMap vo-classname="com.futurebank.e-Banking.types.valueobjects.CustomStateCodeVO"
isCookieReq="N" isExistingVO="N">

            <ParamInfo>

                <column>LANG_ID</column>

                <vo-field>languageID</vo-field>

                <field-type>CustomLanguageId</field-type>

            </ParamInfo>

        </ResultSetMap>

    </DataBuilder>

</DataBuilderList>
```

Extend the product List mapping in Extended FG to accommodate this extra field.

```
<mapping type="LIST" id="StateCodeListMapping_CUSTOM" datatype="StateCodeCacheVO"
recordType="StateCodeVO" >

    <doc>Mapping of the results of a state code list query</doc>

    <field-mappings>

        <field-mapping form-field="LANG_ID_ARRAY[i]" vo-field="resultList[i].extensionVO.languageID" />
```

```
</field-mappings>
```

```
</mapping>
```

PDL also needs to be changed to accommodate this extra field.

How do I skip a record from selected set?

Create a post custom query hook (.java file) which extends DefaultCustomQuery .

Override execute () method for skipping a particular method.

For each record this method will get called. Return type of this method is a Boolean.

For a particular record which is to be skipped, make the return value as 'false'.

The entry of this custom hook is made in Custom.xml

```
<customVariationIdentifier id="StateCodeDetailsServiceFetchImpl">  
  
  <customVariationHook hookPoint="CustomPostQuery">  
  
    <customVariationHookClassName>com.futurebank.e-  
      Banking.CustomStateCodeAssociateQuery</customVariationHookClassName>  
  
  </customVariationHook>  
  
</customVariationIdentifier>
```


Customization in HIF Layer

Host Integration Framework (HIF) enables Finacle e-Banking applications to integrate with different hosts having heterogeneous protocols. This component is meant to be customizable completely. The product provides special facilities for achieving host integration through declarative and in some cases implementation of abstract classes. It has the below customization options:

Customization of Request Specification File

Primary responsibilities of a Request Specification file are to perform orchestration, request splitting, response consolidation etc. During customization one may require to modify the way message orchestration, request splitting, response consolidation etc. are happening. So, customization is required in Request Specification XML file.

This file is provided by product team during product release. One can directly modify this file and put it in "data/hif/custom/request" folder. When request come for a specific Request Specification file, system first looks into the custom folder. If the file is not found there, it goes onto looking at "data/hif/product" folder. So files introduced during customization takes precedence over product

Customization of Message Descriptor File

Message Descriptor file contains details of a message, like the Message Transformer file for the message, the host details for this message, the offline handler for the message etc. Name of this file is HIF_Message.xml and it can be found at "data/hif" folder.

One can directly modify this file during customization to meet Bank's requirement.

Customization of Message Transformer file

Message transformer file is responsible for conversion of e-Banking VO to host specific format (String or another VO) and vice versa. Transformer files can be XML as well Java files. All transformer files coming out-of-box in product are XML. These files would be provided to customization team and they can modify these files to align to customer's needs. If modified during customization, it should be kept in "data/hif/custom" folder. When look up happens for a specific transformer file, framework first looks into "data/hif/custom" folder for the file and then looks into "data/hif/product" folder. Hence files present in custom folder always get preference.

Customization of Host Descriptor File

Host Descriptor file contains details of each host like IP address, port number, JNDI names etc. and also the endpoint adapter class name that knows how to open connections with host and do the communication. This file is a XML file and all these aspects are customizable as per implementation requirements.

What file I have to change for host configuration?

EBHIF_Host.xml has to be modified/changed for host configuration. Depending upon which Host (FI or BC) you want to change, their property attribute has to be modifies accordingly.

For BC the following property has to be modified:

Code Alert:

```
<propertyConfig>
```

```
<properties>
```

```
<property name="MessageFormat" value="ISO8583"/>
```

```

    <property name="Host" value="10.66.118.2" />
    <property name="Port" value="31980" />
    <property name="HeaderLength" value="4" />
    <property name="ControllerId" value="CMN" />
    <property name="retryCount" value="1" />
    <property name="TimeOut" value="99999" />

  </properties>
</propertyConfig>

```

For FI the following property has to be modified:

```

<propertyConfig>
  <properties>
    <property name="MessageFormat" value="XML"/>
    <property name="URL" value="http://vindya.ad.infosys.com:35680/FISERVLET/fihttp"/>
  </properties>
</propertyConfig>

```

What file I have to change for message-to-host mapping?

EBHIF_Message.xml has to be changed/modified for message-to-host-mapping.

```

<messageConfig messageName="LnAccountPenDmdMessage">
  <hostMessageConfigList>
    <hostMessageConfig hostName="FI" routeName="XML/HTTP">
      <messageTransformer>LnAccountPenDmdTransformer</messageTransformer>
    </hostMessageConfig>
  </hostMessageConfigList>
  <offlineHandlerImpl></offlineHandlerImpl>
</messageConfig>

```

In the above code snippet, "routeName" is the attribute of the <hostMessageConfig> Tag which can be configured so as to change the way the message are being sent and received from the host in the following way:

```

<messageConfig messageName="LnAccountPenDmdMessage">

```

```

<hostMessageConfigList>

    <hostMessageConfig hostName="FI" routeName="EJB">

        <messageTransformer>LnAccountPenDmdTransformer</messageTransformer>

    </hostMessageConfig>

</hostMessageConfigList>

<offlineHandlerImpl></offlineHandlerImpl>

</messageConfig>

```

How do I write a new custom XPI?

A CustomXPI will act as a new processing block required to meet some desired requirements. Its usage in a custom transformer spec is as follows (the following usage shows a dummy implementation of Custom system out)

```

<event-handling-block>

    <step>XPI:: CUSTOMSOP("In Tranformer, getting processed by a CustomXPI");</step>

</event-handling-block>

```

For using a new custom XPI we have to first write a customXPI and this XPI is called through a Loader which is configured in AppConfig.xml against the property "XPI_LOADER_CUSTOMIZATION".

Currently this property is configured with "CustomXPILoader" value. This value is the fullyQualifiedpath of the CustomXPILoader class. The property is configured in the following way:

```

<Param name = "XPI_LOADER_CUSTOMIZATION" value =

    "com.infosys.e-Banking.hif.interpreter.xpi.CustomXPILoader" />

```

A new constants class file "CustomXPIConstants has to be created in the package

"com.futurebank.feba.framework.hif.meta.(Dummy package)

which would contain the entry for the newly created XPI.

```

public class CustomXPIConstants {

    private CustomFGIConstants(){}}

    public static final String CUSTOMSOP                = "CUSTOMSOP";

}

```

The constant "CUSTOMSOP" will now be used in CustomXPILoader class to invoke the newly written XPI for processing whenever "XPI::CUSTOMSOP" is encountered in a custom transformer.

```

public class CustomXPILoader implements IXPILoader {

```

```
public void load(Map pXPIInterpreterMap) {  
  
    pXPIInterpreterMap.put(CustomXPIConstants.CUSTOMSOP, new CustomHIFSOPInterpreter());  
  
}
```

Important:

Please note that while trying to generate an HIF spec having a new custom XPI, the HIF compiler tool should also refer to the files modified for new XPI. For example, HIF compiler tool refers to a class XPIInterpreterRegistry. Now, while writing a new XPI, this file has been modified. Hence this class needs to be present in the classpath of the HIF compiler tool. Otherwise, it will give error saying that the newCustomXPI is not a valid XPI and the HIF will fail to generate. Follow similar approach for other similar errors occurring during generation of new XPI.

Context

How do I configure custom context VO?

Make an entry in custom.xml for the hook exposed by OpContext.java as shown below:

Example:

```
<customVariationIdentifier id="OpContext">

<customVariationHook hookPoint="CUSTOM_CONTEXT_VO_NAME">

    <customVaritionHookClassName>

        com.infosys.e-Banking.types.valueobjects.FutureBankContextVO

    </customVaritionHookClassName>

</customVariationHook>

</customVariationIdentifier>
```

In above example, FutureBankContextVO is configured as custom context VO and always customVariationIdentifier id attribute value will be OpContext and hookPoint attribute will be CUSTOM_CONTEXT_VO_NAME, but customVaritionHookClassName (VO name with full package structure) can be any FEFBAVO.

Error Code/Incidence Code Customization

How do I throw a new custom error code?

For throwing a new custom error code, always pass first parameter (Boolean) with value as “true” for any kind of exception.

Code Alert:

```
throw new BusinessException (true,Context,  
  
incidenceCode,  
  
"AdditionalMessage",  
  
ErrorFieldTag,  
  
errCode,  
  
param,  
  
cause);
```

When a user raises a business exception with Namespace parameter value as “true”, it throws a custom error code. If user does not pass first parameter (Boolean) then it will be a product error code by default. Similar to this other exceptions like FEBAException, CriticalException etc... can be raised.

How do I throw an existing product error code?

For throwing a new product error code, always pass first parameter (Boolean) with value as “false” for any kind of exception.

Code Alert:

```
throw new BusinessException (false,Context,  
  
incidenceCode,  
  
AdditionalMessage,  
  
ErrorFieldTag,  
  
errCode,  
  
param,  
  
cause);
```

When a user raises a business exception with Namespace parameter value as “false”, it throws a product error code. If user does not pass first parameter (Boolean) then it will be a product error code by default. Similar to this other exceptions like FEBAException, CriticalException etc... can be raised.

How do I create a new error code in customization?

New error codes can be created through Admin application.

The screenshot displays the Finacle Admin application interface. On the left is a navigation menu with categories like Application Maintenance, Maintenance, Code Maintenance, Error Codes, and various administrative functions. The main content area shows the breadcrumb path: Application Maintenance > Code Maintenance > Error Codes > Create Error Codes. The 'Create Error Codes' form is active, showing fields for Bank Id (DBS), Error Namespace (Custom), Error Code, Error Type, Description, Help Message, Image HTML URL, and Sound HTML URL. Each field has a corresponding input box. At the bottom right of the form are buttons for Submit, Cancel, and Back. The footer of the application window states: 'Finacle e-banking Solutions is a Trademark of Infosys Technologies Limited, India, 2007'.

Go to Application Maintenance > Code Maintenance > Error Codes > Create Error Codes. Here on the create screen enter all mandatory details and click on submit button to create a new custom error code.

Customization in UI Layer

UI layer is expected to undergo heavy customization during implementation. FEBA Customization Infrastructure provides enough options to customize UI layer easily and effectively.

Customization option available in UI Layer are-

UI Component	Creation Of New One	Modifying Existing One
Wireframes	Creation of new wireframes are supported	Product wireframes are given to customization team so that they can directly change.
Section	Creation of new section is supported	Product section XMLs are given to customization team so that they can directly change.
Component	Creation of new component is supported	Product component xml is given to customization team so that they can directly change.
Tag Library	Creation of new tag library is not supported	Product tag library is given to customization team for modification so that one can add new tags during customization.
Taghelper	<ul style="list-style-type: none"> Create a new tag helper in custom packages. Mention that custom package name in Appconfig.xml file under the parameter name CUSTOM_TAGHELPER_PACKAGE Make the entry in components.xml and in classname.properties files 	<ul style="list-style-type: none"> Create a new tag helper with the same name in custom packages and overwrite the logic. Mention that custom package name in Appconfig.xml file under the parameter name CUSTOM_TAGHELPER_PACKAGE
UI Filter	<ul style="list-style-type: none"> Create a new filter in custom packages. Mention that custom package name in Appconfig.xml file under the parameter name CUSTOM_UI_FILTER_PACKAGE The entries of this filter are to be done into two tables which are FDET and FCOT 	<ul style="list-style-type: none"> Create a new filter with the same name in custom packages and overwrite the logic. Mention that custom package name in Appconfig.xml file under the parameter name CUSTOM_UI_FILTER_PACKAGE
Formatter	<ul style="list-style-type: none"> Create a new formatter in custom packages. Mention that custom package name in Appconfig.xml file under the parameter name CUSTOM_UI_FORMATTER_PACKAGE. Use this formatter in pdl 	<ul style="list-style-type: none"> Create a new formatter with the same name in custom packages and overwrite the logic. Mention that custom package name in Appconfig.xml file under the parameter name CUSTOM_UI_FORMATTER_PACKAGE

Customization is supported for filters, tag helpers and formatters. For these one can write a custom filter or tag helper or formatter based upon the requirement.

The application configuration file has the following parameters to be specified:

- CUSTOM_TAGHELPER_PACKAGE

We can create a new custom tag helper for customization requirements and can place it in the path mentioned in Appconfig.xml file under the parameter name CUSTOM_TAGHELPER_PACKAGE. Also the entry for the same tag helper has to be made into components.xml, classname.properties files and EBUIConfig.xml

An existing tag can also be modified. Such tags should be kept in the path mentioned under parameter name CUSTOM_TAGHELPER_PACKAGE.

The UI Framework checks for the modified tag helper and uses it if found, if it doesn't find one it picks the original from the product path.

- CUSTOM_UI_FILTER_PACKAGE

We can create a new custom filter for customization requirements and can place it in the path mentioned in Appconfig.xml file under the parameter name CUSTOM_UI_FILTER_PACKAGE. Also the entries of this filter are to be done into two tables which are FDET and FCOT. Also, an entry should be made in EBUIConfig.xml.

An existing filter can also be modified. Such filters should be kept in the path mentioned under parameter name CUSTOM_UI_FILTER_PACKAGE. The UI Framework checks for the modified filter and uses it if found, if it doesn't find one it picks the original from the product path.

- CUSTOM_UI_FORMATTER_PACKAGE

We can create a new custom formatter for customization requirements and can place it in the path mentioned in Appconfig.xml file under the parameter name CUSTOM_UI_FORMATTER_PACKAGE. An entry should be made in EBUIConfig.xml.

An existing formatter can also be modified. Such formatters should be kept in the path mentioned under parameter name CUSTOM_UI_FORMATTER_PACKAGE. The UI Framework checks for the modified formatter and uses it if found, if it doesn't find one it picks the original from the product path.

How do I add a new link on sidebar?

For adding a new link on sidebar we have to make entries into two tables of our Database to which we are pointing. These tables are VMOT and VMNP.

Firstly we make the entries into VMOT table. In this we have to give the MNU_ID which is to be used for the link and FG_LINK_URL in which we give the name of the Formsgroup which we are going to call through that link and the call mode which is kept as 0 by default.

Example:

FORMSGROUP_ID=ComposeMailFG&Call_Mode=0.

Now in VMNP table we can link the profile (profile linked to the user) to MNU_ID which is created above. Hence link will be accessed to the user through this MNU_PROFILE_CD.

MNU_PROF_CD linked to MNU_ID as per our requirements can be RUSER, DEF_RET, CUSER etc.

Now we have to check whether this link is a Parent one that is Main Link or a Child one that is sub link.

If it is Parent link then we have to pass the value as "PARENT" to MNU_ID_TYPE and it will have "!" value in PARENT_MNU_ID as it is itself a Parent. Now consider that you want your link to be a sub link, we have to pass the value as "CHILD" to MNU_ID_TYPE and the Parent MNU_ID name to PARENT_MNU_ID which is having this link as its sub link.

How do I write a new custom UI formatter?

A custom UI formatter will format the values on screen according to the requirements that are not currently available in the product.

Package of a custom UI formatter is to be configured in AppConfig.xml.

Code Alert:

```
<Param name = "CUSTOM_UI_FORMATTOR_PACKAGE"
      value="com.futurebank.e-Banking.ui.util.formatter"/>
```

The custom UI formatter should be present in this package.

Custom UI formatter should extend AbstractFormatter.

Code Alert:

```
public class MyCustomFormatterClass extends AbstractFormatter {

    @Override

    public String format(String value, ICacheManagerView cache)

        throws Exception {

        return null;

    }

    @Override

    public String format(IFEBAType value, ICacheManagerView cache) {

        return null;

    }

    @Override

    public String format(Object value, IContext context) throws Exception {

        return null;

    }

}
```

As shown above, the formatter needs to implement the method "format", available for input types String / any FEBA type / any object type as applicable to the requirement.

It can be called from a PDL as follows:

```
<control          component="OutputTextbox"          location="DispFormWithListTable,Ra1,C2"
name="TranRequestManagerFG.REQUEST_ID"
fldFormatter="MyCustomFormatter"/>
id="TranRequestManagerFG.REQUEST_ID"
```

How do I write a new custom UI taghelper? [Applicable for 11.0.5 & above]

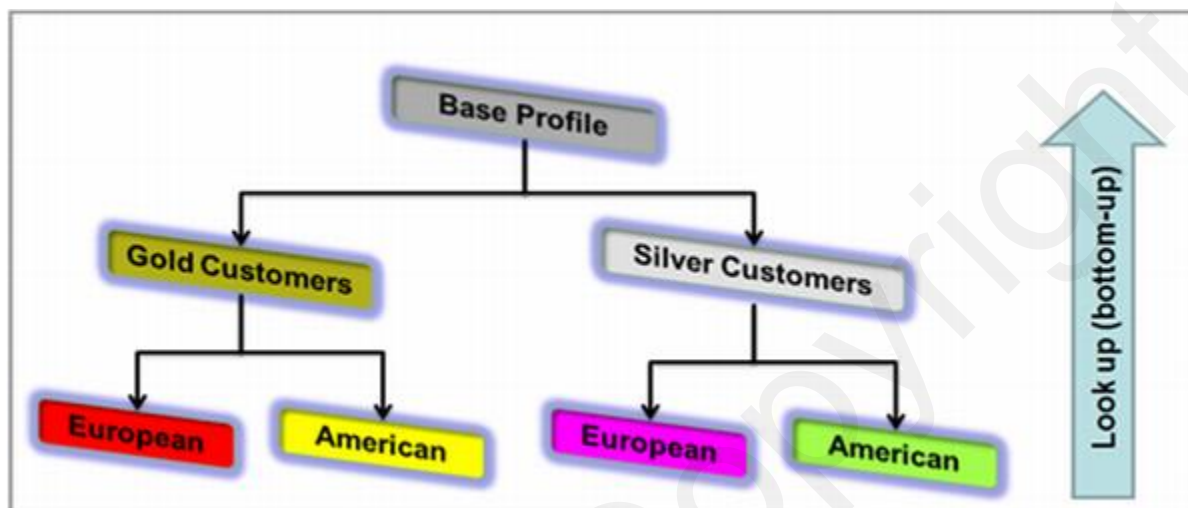
A custom UI taghelper is created to have its own functionality or to have a different UI [style] apart from product taghelper. Following are the steps to keep in mind while writing a new custom taghelper through VDT tool:

- Make sure that the entry for custom taghelper is made in classname.properties.
- Make sure that the entry of custom jar [which contains the custom taghelper] name is made in the "Add Additional jars for Customization" textbox of Window->Preferences-> Visual Developer Preferences.
- Make sure that the custom jar [which contains the custom taghelper] is present in the folder WebContent\WEB-INF\lib of your runtime project.
- Make sure that the entry for custom taghelper is made in UIConfig.xml file in the below format:

```
<Param    name="ButtonTagHelper"    value="com.infosys.feba.framework.ui.taglibs.helpers.ButtonTagHelper"
/>
```

Multi Branding

Multi branding aims at providing different user experience for different users within same application installation. For example, one application installation can differentiate and cater to different geographies. Moreover, it can be made to identify priority or high net worth customers and present the screens, navigations and backend processing accordingly. The same is depicted in the picture below:



This is implemented by assigning a profile to the user when logging in using an URL. The Profile Resolver component takes this responsibility. Out-of-box implementation of the resolver looks into the URL and finds out the correct profile by referring to a profile descriptor file (profileDetails.xml) present in the working folder. However this implementation itself can be changed as name of the implementation is decided by a property "PROFILE_IMPL" bootstrap properties file.

The entries which had to be made in the profileDetails.xml file will look like this-

```

<Entity ProfileID="PR1">

  <FIELD>

    <name>BankId</name>

    <value>FUTUREBANK</value>

  </FIELD>

  <FIELD>

    <name>BrandId</name>

    <value>Loan</value>

  </FIELD>

</Entity>

```

Now, let's consider a user logging in with below URL-

"http://localhost:8080/EBCustomWeb/AuthenticationController?__FORMSGROUP_ID__=AuthenticationFG&__START_TRAN_FLAG__=Y&__FG_BUTTONS__=LOAD&ACTION.LOAD=Y&AuthenticationFG.USER_TYPE=1&BANK_ID=ICICI&BRAND_ID=Loan".

In this case, since the BankId is "FUTUREBANK" and BrandId is "Loan" as per the URL, the user's profile would be resolved to PR1 and all resources like configuration files, fg spec files, hif spec files, CSS, script, JSP etc related to PR1 would be used for this user. Once the login URL is submitted and user profile is resolved a 'Context' will be updated with EntityProfileDetailsVO . This VO contains both profile id and parent profile id. At the end of this process, user's session becomes profile aware.

Note that the resource lookup process is intelligent enough to look into specific profile first, followed by parent profile and finally default to base product profile.

The profile awareness has been brought into each layer of customization. This enables entity specific customization.

Multi Branding in UI Layer

Below table describes different Multi Branding capabilities of UI Layer:

Layer	Attribute	Multi Entity Support
UI	Images	Image lookup is multi branding enabled. Images need to be kept in profile specific folder
UI	CSS	Style sheet lookup is multi branding enabled. Style sheets need to be kept in profile specific folder.
UI	Scripts	Script lookup is multi branding enabled. Scripts need to be kept in profile specific folder
UI	JSP	The profile specific customized JSP need to be placed in separate folder rest of the jsp would fall back to default structure.

A tool "devrun_jspcustompath.bat" is provided to generate a map for faster location of the JSP in multi entity environment. After running this tool the mapping of these jsps is done in ProfileBasedJsp.properties file which is present in the working folder.

Example:

PR1.MailsViewInFolder.jsp=PR1/L001/jsp/emails/MailListFG/MailsViewInFolder.jsp

PR2.MailsViewInFolder.jsp=PR2/L001/jsp/emails/MailListFG/MailsViewInFolder.jsp

So the JSPs are present in the profile folders that is PR1 and PR2 folder get mapped by the tool in to the ProfileBasedJsp.properties file.

Also the profile specific properties (files/folders) need to be made available in separate folders (profile specific), rest of the properties would fall back to default values. The folder name (profile specific) has a naming convention that is it will be "PROF_ProfileName".

Example:

For profile PR1 it will be "PROF_PR1".

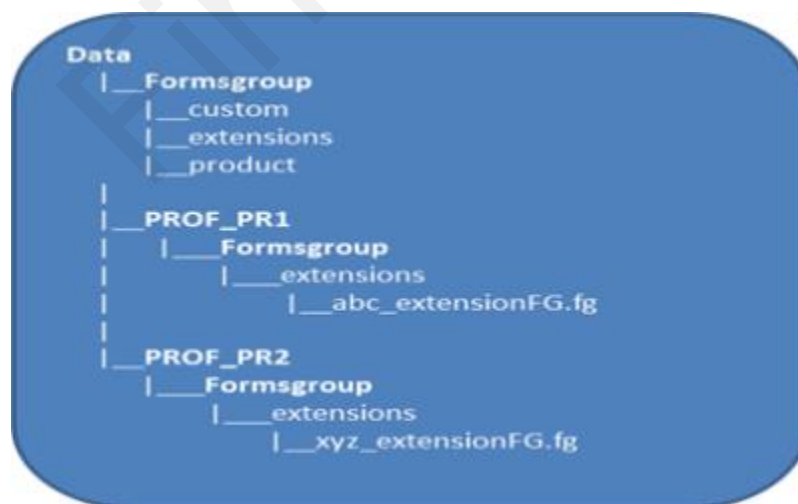
The structure of the folder having profile specific properties is as below:



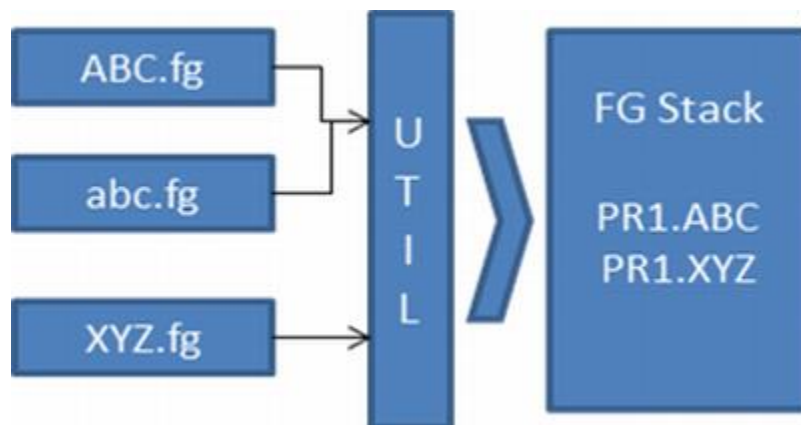
Multi Branding in Formsgroup Layer

The FG (.fg files) needs to be maintained separately under profile specific folder. They will integrate with product FG at runtime. So the profiles can have extensionFG based upon requirement.

Example:



The intelligence of integrating with the profile specific custom .fg and caching has been built into the FG layer.



Multi Branding in Transaction Layer

The custom hooks are executed on command pattern configured in Custom.xml. The same needs to be duplicated for each profile. So if we have some hooks which are to be implemented into profile specifically then we will provide the hook information in custom.xml file kept under profile folder.



Multi Branding in Database Layer

The DAL generated components Data builder and Query builder are configured in DAL.properties reading/caching of which is profile aware. So if we have some Data builder and Query builder which are to be implemented into profile specifically then we will provide their information in DAL.properties file kept under profile folder.



How do I Enable MultiProfile/MultiBranding in e-Banking?

Multi branding aims at providing different user experience for different users within same application installation. For example, one application installation can differentiate and cater to American Customer & European Customers. Moreover, it can be made to further go down and identify Silver and Gold customers and present the screens, navigations and backend processing accordingly.

1) Define the profile in ProfileDetails.xml

Make an Entry for an Entity like this-

```

<MultiEntitySpec>

    <ProfileDefinition>

        <fieldname>BANK_ID</fieldname>

        <fieldname>BRAND_ID</fieldname>

        <fieldname>Location</fieldname>

    </ProfileDefinition>

    <Entities>

    <Entity ProfileID="PR1">

    <FIELD>

    <name>BANK_ID</name>

    <value>FUTUREBANK</value>

    </FIELD>

    <FIELD>

    <name>BRAND_ID</name>

    <value>Loan</value>

    </FIELD>

```



```
</Entity>
```

```
</Entities>
```

```
</MultiEntitySpec>
```

Now, let's consider a user logging in with below URL-

```
"http://localhost:8080/EBCustomWeb/AuthenticationController?__FORMSGROUP_ID__=AuthenticationFG&__START_TRAN_FLAG__=Y&__FG_BUTTONS__=LOAD&ACTION.LOAD=Y&AuthenticationFG.USER_TYPE=1&BANK_ID=ICICI&BRAND_ID=Loan".
```

In this case, since his BANK_ID is "FUTUREBANK" and BrandId is "Loan" as per the URL, his profile would be resolved to PR1 and all resources like configuration files, fg spec files, hif spec files, css, script, jsp etc related to PR1 would be used for this user.

2) CSS ,images, java scripts, html files etc needs to be placed in profile specific folder under WebContent.

WebContent structure will look like this.

Here PR1 is the profile .So all images ,css, java scripts etc needs to be placed under this folder.

3) Run jspcustompath gen tool.

This tool is used to create Profilebasedjsp.properties file. This property file contains the absolute path of profile specific jsps. Before running this tool all profile specific jsps needs to be put in profile specific folder under webcontent folder. This tool will take absolute path of the webcontent folder as an input and creates Profilebasedjsp.properties file. The content of generated property file will look like one mentioned below.

```
PR1. MailsViewInFolder.jsp= PR1/L001/jsp/mails/MailListFG/ MailsViewInFolder.jsp
```

```
PR2. MailsViewInFolder.jsp= PR2/ L001/jsp/mails/ MailListFG/ MailsViewInFolder.jsp
```

```
PR2. MailViewSent.jsp= PR1/ L001/jsp/mails/ MailListFG/MailViewSent.jsp
```

Place this generated property file directly under data folder.

4) Keep profile specific FG, HIF etc in profile specific folder under data folder.

The FG (.fg files) needs to be maintained separately under profile specific folder. They will integrate with product FG at runtime. So the profiles can have extensionFG based upon requirement.

5) Keep profile specific property files (dal.properties ,custom.xml,appconfig.xml etc) in profile specific folder under data folder. If you don't have any profile specific properties file then no need to put this under PROF_PR1 folder.

Customization of CorpAdmin Framework

It covers all possible customization requirements in CorpAdmin from a technical aspect and explains how the framework can solve such requirements.



This document is arranged according to various elements present in CA config file. For each element it provides the customization solution in details. Code for the solutions can be found along the solution for given Extension. Following type of extensions are possible for Corp Admin Framework:

- FORM-FIELD EXTENSION
- MAPPING EXTENSION
- UI-DATA EXTENSION
- PRE-POPULATION EXTENSION
- FUNCTION EXTENSION

FUNCTION EXTENSION

- Error Code Mapping Extension
- Table Extension
- Val Array Extension
- Hook (Pre /Post) Extension

Legends

Legend	Meaning
	Information center. Important information would be mentioned in this section.
	Code alert. Raw codes would be displayed in this section. It may be pseudo code also.

Naming Conventions for CA Extension File

The CorpAdmin extension file should be named as "Product file name + _Extension.xml". For example, if the name of product CA config file is "ACM.xml" then the name of extended CA config file should be "ACM_Extension.xml".

Note:

So if we give extension file name other than as stated above, it will give error that Extension file name should end with "_Extension".

Form-Field Extension

In form-field extension, we can add new form-fields in Extension CA config file. The new form-fields introduced in extension CA config will be merged with the form-fields present in product CA config file. The new form-field can be added in extension CA config file as shown below:

Code Alert:

```
<form-data>

  <form-field>

    <doc>Holds account nickname value</doc>

    <field-id>AC_NIC_NAME</field-id>

    <datatype>AccountNickName</datatype>

  </form-field>

</form-data>
```

Note:

We cannot add same form-field as present in product CA config file in Extension CA config file. This will lead to Error at runtime stating "Form field exist in parent CA"

Mapping Extension

In Mapping extension, we can add new mapping in Extension CA config file. The new mapping introduced in extension CA config will be merged with the mapping present in product CA config file. We can also override same mapping as present in product CA config file. The new mapping can be added in extension CA config file as shown below:

```
<mappings>

  <mapping typename="AccountNicknameMaintenanceDetailsVO" id          ="AccountNumberMapping">

    <map name="accountNickname" value="AC_NIC_NAME"></map>

    <map name="accountType" value="ACCOUNT_TYPE"></map>

    <map name="accountTypeDescription" value="ACCOUNT_TYPE_DESCRIPTION"></map>

  </mapping>

</mappings>
```

Note:

So if we add same mapping as present in product CA config file in Extension CA config file. This will override the product mapping.

UI Data Extension

UI-data contains two elements, namely, "Main page" & "Confirmation Page". In UI-data extension, we can add new UI-data in Extension CA config file. The new UI-data introduced in extension CA config will be merged with the UI-data present in product CA config file. We can also override UI-data as present in product CA config file.

In case we want to override both main page and confirmation page then Extension file should contain below mentioned code.

```
<ui-data mainPage="CrpAccMaintenanceUpdate" confPage="CrpAccMaintenanceUpdateCfm"/>
```

For overriding only the Main Page,,we will specify the main page as shown below in the extension file.

```
<ui-data mainPage="CrpAccMaintenanceUpdate" />
```

For overriding only the Confirmation Page,,we will specify the confirmation page as shown below in the extension file.

```
<ui-data confPage="CrpAccMaintenanceUpdateCfm"/>
```

Note:

UI-data present in Extension CA config file will override the UI-data present in the Product CA config file.

Pre-Population Extension

In pre-population extension, we can add new pre-population in Extension CA config file. The new pre-population introduced in extension CA config will be merged with the pre-population present in product CA config file. We can also override same pre-population as present in product CA config file. The new pre-population can be added in extension CA config file as shown below:

```
<pre-population>

  <field>

    <id>AUTH_MODE</id>

    <mode-population>DEFAULT</mode-population>

    <default-value>S</default-value>

  </field>

</pre-population>
```

Note:

So if we add same pre-population as present in product CA config file in the Extension CA config file, This will override the product pre-population.

Function Extension

Functions in CorpAdmin are defined under "operations" tag. Each function may contain Error-Code Mapping, Val-array, Table, Hooks(Pre-hook and Post-hook). We can extend or override any of these elements.

Error Code Mapping Extension

In Error-Code Mapping extension, we can add new Error-Code mapping in Extension CA config file. The new Error-Code mapping introduced in extension CA config will be merged with the mapping present in product CA config file. The new Error-Code mapping can be added in extension CA config file as shown below:

```
<operations>

  <function type="modify">

    <errorCode-mapping>
```

```

    <businessInfo>

    <mapInfoCode cafwMsgCode="CAFW_MODIFY_SUCCESS" ucMsgCode="101167" />

    </businessInfo>

  </errorCode-mapping>

</function>

</operations>

```

Val-Array Extension

In Val-array extension, we can add new Val-array in Extension CA config file. The new Val-array introduced in extension CA config will be merged with the mapping present in product CA config file. The new Val-array can be added in extension CA config file as shown below:

```

<operations>

<function type="modify">

<val-array>

  <item>

    <error-tag>dataList['CORP_ID'].srFieldValue</error-tag>

    <validator>CorporateIDVal</validator>

    <map-valvo>CorpUserMapping</map-valvo>

    <mandatory>Y</mandatory>

    <dependent>N</dependent>

    <error-code>8505</error-code>

  </item>

  <item>

    <error-tag>dataList['ACID'].srFieldValue</error-tag>

    <validator>AccountIDVal</validator>

    <map-valvo>AccountNumberMapping</map-valvo>

    <mandatory>N</mandatory>

    <dependent>Y</dependent>

  </item>

</val-array>

```

```
</function>
```

```
</operations>
```

Note:

If we add any Val-Item in Extension CA config file, it will get merged into product Val-Array.

Table Extension

Under Table tag of CA config file, we define Key fields and Non-Key fields. In Extension CA config file, we can define a new table to be merged into the Product for a particular function or we can extend a table present in Product CA config file.

If we want to merge a new table in Product CA meta info then we define a new table in Extension CA config file as shown below:

```
<operations>

  <function type="selectForInsertOrUpdate">

    <process>

      <table name="ACMX">

        <associateKeyFields>

          <param tableColumn="BANK_ID" fieldId="BANK_ID"/>

          <param tableColumn="BRANCH_ID" fieldId="BRANCH_ID"/>

          <param tableColumn="ACID" fieldId="ACID"/>

          <param tableColumn="BAY_USER_ID" fieldId="CORP_ID"/>

          <param tableColumn="USER_BANK_ID" fieldId="BANK_ID"/>

        </associateKeyFields>

        <associateNonKeyFields>

          <param

            tableColumn="AC_NIC_NAME" fieldId="AC_NIC_NAME"/>

        </associateNonKeyFields>

      </table>

    </process>

  </function>

</operations>
```

If we want to extend a table already present in Product CA config then we define the non-key fields of that table in Extension CA config file as shown below:

```

<operations>

  <function type="selectForInsertOrUpdate">

    <process>

      <table name="ACMX">

        <associateNonKeyFields>

          <param

            tableColumn="AC_NIC_NAME" fieldId="AC_NIC_NAME"/>

          </associateNonKeyFields>

        </table>

      </process>

    </function>

  </operations>

```

Note:

We cannot Extend new Key fields for a table. If we define some key fields in Extension CA config file, then this will lead to Fatal Exception at runtime stating "Extension key fields can't be merged into product key fields"

Hooks Extension

There are two types of hooks that is Pre-hook and Post-hook. A function can have a single Pre-hook or Single Post-hook or both. If a function does not have Pre/Post hook then we can write it in Extension file and it gets merged with the function in Product CA config file. We cannot override a Pre/Post hook.

Code for adding Pre-Hook :

```

<operations>

  <function type="modify">

    <processhook type="pre" mapId ="PersonalProfileValuesMap" className="com.infosys.e-
      Banking.crpadmin.service.SetPreferencesCopyServiceImpl"/>

  </function>

</operations>

```

Code for adding Post-hook:

```

<operations>

  <function type="modify">

    <processhook type="post" mapId ="PersonalProfileValuesMap" className="com.infosys.e-
      Banking.crpadmin.service.SetPreferencesCopyServiceImpl"/>

  </function>

</operations>

```

</function>

</operations>

Note:

If we try to override any hook(pre/post) in Extension CA config file, then it will lead to Error at run-time stating that "The hook type is already present in Product CA config file".

Customization of Formmanagement Framework

Formmanagement customization framework provides flexibility to the developer that use case customization can be done with minimal configuration changes.

Pre-requisites:

The person doing customization should have the clear understanding of Form Management framework and should also know how to enable and use the macro. Using Formmanagement framework customization, users would get acquainted with:

- Tools Used - Custom Macro Tool Details
- Color coding used in customization Excel Workbook
- Naming Conventions-For New Forms and Existing Forms
- Scope For Customization In Form Management Framework
- Factory Class Customization Present In Form Management Framework

Tools to be used:

The different tools used for customization of Formmanagement Framework include:

1. Product Excel Workbook with macro
2. Customization Excel Workbook with macro
3. FM Compiler

User configures the FMF workbook excel with required details. XLS2XMLConverter and XML2XLSCConverters are enabled with macros and generates XML from excel and vice versa. Once it is done, FM Compiler generate .fm file from .xml.

FM Compiler Tool

FM Compiler tool is used to convert the configuration XML to an .fm Meta file. The XML is generated using the FM Macro tool from the excel sheet configuration done by the developer. FM Compiler also generates the audit seeds for form fields AFLT Seeds and events AUFC Seeds from the XML spec file. This tool is used for compiling both Product and Custom FM Specs.

Important terms:

FM_OUTPUT - The path where the .fm file has to be generated.

AUDIT_OUTPUT - The path where the audit seeds, AFLT.sql and AUFC .sql will be generated.

COMPILER_MODE - The mode in which Compiler is being run. It will be P in case Product spec is being generated and C in case custom extension spec is being generated.

PRODUCT_META_FILE - the path of the compiled product Meta file. This is mandatory in case the COMPILER_MODE IS C.

The path from which application will refer the .fm Meta file is configured in App Config.xml with the parameter name "FORM_MANAGEMENT_PATH". The path of custom .fm file (for new Form) will be defined with parameter name "FORM_MANAGEMENT_CUSTOM_PATH". The extension .fm files will be taken from the path given by parameter name "FORM_MANAGEMENT_EXTENSION_PATH". So FM_OUTPUT should be as per this configuration.

Validations

FM Compiler Tool performs a set of validations on the input XML spec to check for improper configurations done by the developer. This will help in reducing runtime issues because of wrong configurations. The validations performed by the tool are : RM_APPROVAL_PARAMS defined are in proper format.

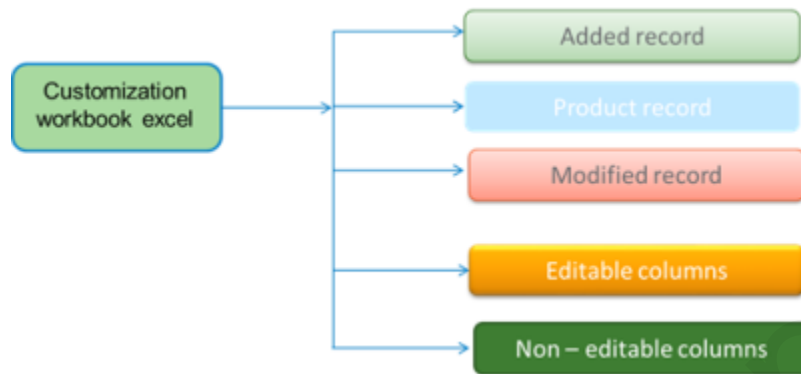
1. If IS_ARRAY_TYPE is Y, sourcefield is mandatory.
2. If PERSISTENCE_REQD is Y Extended Table field id is mandatory.
3. If IS_REQD_WF is Y Workflow field id is mandatory.
4. If IS_LIMITS is Y Limit field id is mandatory.
5. Workflow Field names can be only from a specific set(defined in developer guide)
6. Limit Field names can be only from a specific set(defined in developer guide)
7. A decision field cannot be an array field(If IS_DEC_FIELD is Y, IS_ARRAY_TYPE cannot be Y)
8. A decision field cannot have DEFAULT_VALUE.
9. Extended Field Id & FEBAAmount should be in format currency|amount
10. If HOST PERSISTENCE is Y, LOCAL PERSISTENCE should be Y
11. IS_MULTI_ROW_ACTION is Y and Multi row action should be given.
12. Multi Row action defined should be from the defined valid set only (ADD_ROW,EDIT_ROW and DELETE_ROW)
13. IS_VALIDATION_REQD is Y and Val scope should be given.
14. Dynamic View Params should be in the proper defined format.
15. Calculation equations should be in proper defined format.
16. Form Fields in calculation should be defined.
17. Output list cannot be empty for Data population configuration.
18. If the value of REQUEST_HOLD_RELEASE_REQD is "Y", then there should be at least one form field being present with the tagging as "holdRelease".
19. If the value of REQUEST_HOLD_RELEASE_REQD is "Y", then there should be at least one action being present with the Value of either "Y" or "N" in HOLD_RELEASE_REQ in Actions List.
20. The form Field having the tagging as "holdRelease" should be of the basic types.
21. Check for Workflow/Limits/Formfield Tags configuration(duplicate tags should not be present)
22. A form field can't be source of two arrays.

23. Compiler check to throw error if val id present in actions tag is not available in sections

24. Form Field defined in sections form field mapping not present in Form Fields List

Color coding used in customization Excel Workbook

Color coding helps in identifying or differentiating product or customization level configuration.



Naming Conventions - For New Forms and Existing Forms

This segment covers:

- Product xml
- Extension xml
- Custom xml
- Custom Extension xml

Product Xml:

This is the xml which gets generated when a form or use case is being developed by developer. The name of the file will be of the form formrequestid.xml.

Extension Xml:

This is the xml which gets generated when an existing form is customized by using customization excel workbook by customization team. The name of the xml file will be of the form formrequestid_Extension.xml

Custom Xml:

This is the xml which gets generated when a new form or functionality is developed by customization team during customization. The name of the file will be of the form Custom+ formrequestid.xml.

Custom Extension Xml:

This is the xml which gets generated when the custom xml is customized for different bank or different branches of the same bank. The name of the file will be of the form Custom+ formrequestid_Extension.xml.

Scope For Customization In Form Management Framework

The scope for customization in Form Management Framework extends to customizing existing use case of forms, and develop new forms in customization. It also covers customizing the customized form or form developed during customization process. This section will also cover what all can be customized under FMF customization.

Customize Existing Use Case or Forms

If customizing the product for the first time:

- Take the product Form_Request.xml
- Reverse the product Form_Request.xml using customization excel workbook
- Do customization using Buttons present in each sheet present
- Generate the Extension xml

If customizing the product for the Second or more time:

- Take the product Form_Request.xml
- Reverse the product Form_Request.xml using Customization excel workbook
- Reverse the Form_Request_Extension xml
- Do necessary customization by using buttons present in each sheet
- Generate the Extension xml

Develop new Forms in Customization

Developing new form or use case in customization is same as developing in product.

Configure the following information in product excel work book:

- Form definition
- Form request definition
- Form field definition
- Form section definition
- Form section validations
- Form action validations
- Form action list configuration
- Form calculation
- Pre- calculators

After configuration generate Form_Request_Custom.xml using the macro buttons provided excel workbook customization tab.

Customize the customized form or form developed in customization

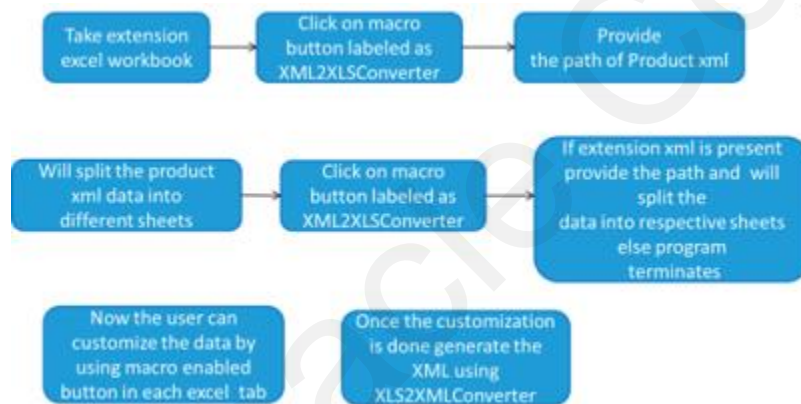
If customizing the Custom xml for the first time:

- Take the Form_Request_Custom xml.
- Reverse the Form_Request_Custom xml using Customization excel workbook.
- Do customization using Buttons present in each sheet present.
- Generate the Form_Request_Custom _Extension xml.

If customizing the Custom for the Second or more time:

- Take the Form_Request_Custom xml.
- Reverse the Form_Request_Custom xml using Customization excel workbook.
- Reverse the Form_Request_Custom xml.
- Do necessary customization by using buttons present in each sheet.
- Generate the Form_Request_Custom_Extension xml

Block Diagram of Customization Flow



If new use case is used to be developed or new for request then use the product excel workbook to develop the same and generate xml and from xml convert it into .fm file using product compiler and place the same in custom folder. During run time the custom.fm file will be used.

In order to customize the existing use case, the customization team has to use the Customization workbook. First of all they need to convert the product xml into the Customization workbook and do whatever customization they want to do as per the guidelines. Once done they have to generate the extension xml which will be only incremental one and need to convert the xml into .fm file using custom compiler and have to put the extension.fm file into extension folder. During run time the product .fm file will be merged with extension.fm file.

What all can be customized?

The following details can be customized in FMF:

- Form_Request
- Field_Definition

- Section_List
- Section_Definition
- Action_List
- Action_Validations
- Calculations
- Pre_Populators

Customizing Form Request Details

To Modify an existing form request details, customization excel workbook will be used. On click of "Modify Form Type" button, the selected row are copied. The columns which are editable are:

CUSTOM_EXT_MASTER_TABLE : If any custom extension table required mention the table name

CUSTOM_EXT_CHILD_TABLE: If any custom extension child table is required mention the child table name

RM_APPROVAL_PARAM : To enable RM approval functionality to the request configure the parameters in format like HelperClass(FormField)[confirmation code and to disable RM approval remove parameters

REQUEST_HOLD_RELEASE_REQ: To enable manual release functionality mark REQUEST_HOLD_RELEASE_REQ value as Y and to disable mark it as N

Customizing Form Field Definition

We will have 3 buttons to customize use case or form at field level

1. Adding a New Form Field:

In order to add a new form field the Customization workbook will be used. On click of the button "Add a new form field" all the mentioned columns in Customization excel workbook are editable in a new row in which the values are blank.

2. Modifying an existing form field:

While modifying, the selected row will be copied and only the allowed columns should be editable. Not allowed columns will be locked. When you click on "Modify an existing form field", the selected row is copied and the below mentioned columns are editable.

The allowed column are:

- **IS_REQ_FOR_WF** : To enable workflow in customization on particular action based on the form filed(AMOUNT ,ACCOUNT and CURRENCY) value, mark IS_REQ_FOR_WF field as 'Y' and to disable mark it as 'N'

- **WF_FIELD_NAME** : if workflow is enabled on form fields(AMOUNT ,ACCOUNT_ID and CURRENCY) provide the WF_FILELD_NAME like AMOUNT , DEBIT_ACCOUNT and CURRENCY AMOUNT , DEBIT_ACCOUNT and CURRENCY: These are tagging information which will be used by Formmanagement framework to capture the workflow enabled field values uniquely.

- **IS_REQ_FOR_LIMIT**: The Entry /Approval limit validations needs to be performed on specified field(TXN_AMOUNT,ACCOUNT_ID,TXN_DATE and ENTRY_ID) during initiation or approval of a transaction by a

corporate user provide IS_REQ_FOR_LIMIT as “Y” and disable Entry/Approval limits on specified provide IS_REQ_FOR_LIMIT as “N”

- **LIMIT_FIELD_NAME:** The Limit form field tags that needs to be given for those fields which are enabled for Entry/Approval validations are TXN_AMOUNT,ACCOUNT_ID,TXN_DATE and ENTRY_ID.
- **DEFAULT_VALUE:** In customization we can remove or provide default value to specific form field.
- **FORM_FIELD_TAG:** To identify the user selected record in case of check box and radio buttons form field tagging is required.

Example:

ENTRY ID : entryId
 MARK FOR DELETE: markForDelete
 SELECTED INDEX : selectedIndex

We can add/remove tagging field information to a specific form field.

- **AUDIT_DESC :** To add/modify audit description for newly/modified form field
- **IS_DECISION_FIELD:** To enable auto action completion based on form field value. This field is used in case of Multi Row Actions like ADD_ROW ,EDIT_ROW .

Example:

A form field which is used in ADD_ROW action and its IS_DECISION_FIELD configured as 'Y'. Then ADD_ROW action completion will not happen until you provide value for that form field

In customization we can enable or disable a field as decision field by using IS_DECISION_FIELD.

3. Deleting a newly added form field:

“Delete a new form field” button we can use only for deleting newly added records.

Customizing sections List

FMF customization allows to add a new section modifying existing section and deleting newly added section

- Adding new section

On click of “Adding a new section “ a new row will be created to enter the section details

- Modifying existing section

The row which the user selects will be copied and the column SECTION_REQ_OR_NOT Having the dropdown with the value of “Y” and “N” will be allowed to modify

- Deleting a newly added section

The row which the user selects gets deleted only if it is newly added.

Customizing Section Definition

Formmanagement framework allows to customize section definition by

- Mapping newly added form field to a section
- Mapping existing form field to other sections
- Adding mandatory and business validations
- Removing newly added mandatory and business validation
- Changing validations order

We will have 3 buttons, one for adding a new Section definition, one for deleting new added section definition by mistake and another for modification of the section definition.

Add New Section definition:

This allows the user to map newly added form field to particular section and to plugin mandatory and business validation.

A row with the dropdown available for SECTION_ID containing the valid sets of values and a dropdown for FORM_FIELD_ID containing the valid set of form fields from which the user can select any of the value.

- **SECTION_ID:** Contains all the section available in section list. So that user can map newly added or existing form field to any section
- **FORM_FIELD_ID :** All form definition fields available as dropdown, so that user can select any form field and map any section

Modify an existing Section:

The row which the user selects will be copied at the end of all the rows and "SECTION_ID" column will be permitted to change with the values contained in the dropdown along with the columns.

VALIDATOR_CLASS : To modify existing business validator calss

- **IS_MANDATORY:** To Modify mandatory validations
- **IS_DEPENDENT :** To change IS_DEPENDENT value 'Y' to 'N' and vice versa
- **IS_LIST_VALIDATOR_FLAG:** if modified business validator accepts FormmanagementVO as the input configure IS_LIST_VALIDATOR_FLAG as 'Y' else 'N'
- **VALIDATION_ORDER:** This allows the user to change validation order
- **IS_ACROSS_SECTION:** This allows the user to changes IS_ACROSS_SECTION values 'Y' to 'N' and vice versa.

Delete Modified Record:

User can delete newly added section definition or modified section definition. Product section definition will not allow to delete

Customizing action list and action definition

Action list customization allows the user to add new action modify existing action and delete newly added/modify action

Customizing action definition allows user:

- To enable/disable validations and changing validation scope of an action
- To enable/disable workflow
- To enable/disable local and host persistence
- To enable/disable limits
- To plugin custom hooks
- To modify the request status
- To enable/disable Hold/Release functionality

FMF customization workbook provides three buttons, one for adding a new action, one for deleting a newly added action, one for modifying an existing action.

Adding New Action:

While adding a new record that is by clicking on button “Adding a New Action”, all fields column are editable.

Modifying Existing Action :

On click of “Modifying an Existing Action” button, the selected row will be copied and the below mentioned columns will be allowed to customize.

IS_VALIDATION_REQD : This field allows user to override action validations by enabling or disabling(‘Y’ and ‘N’)

VALIDATION_SCOPE : This field allows user to modify the validation scope FC to FP or SC to SP and vice versa

IS_LOCAL_PERSISTENCE_REQD : If user wants to enable local inquiry or local persistence provide IS_HOST_PERSISTENCE_REQD field value as ‘Y’. Similarly to disable provide field value as ‘N’.

IS_HOST_PERSISTENCE_REQD : If user wants to enable host inquiry or host persistence provide IS_HOST_PERSISTENCE_REQD field value as ‘Y’. Similarly to disable provide field value as ‘N’.

IS_WF_REQD : For selected action id user can enable or disable workflow functionality by changing IS_WF_REQD ‘Y’ to ‘N’ or ‘N’ to ‘Y’.

WF_ACTION : If workflow is enabled for this action specify the WF_ACTION (I,A and R etc.) and if IS_WF_REQD disabled remove the WF_ACTION

FORM_REQ_STATE: user can specify the customized workflow status

VIEW_ID: User can provide the customized JSP name to be loaded on execution of this action

IS_MULTI_ROW_ACTION: User can change the multi row action behavior by providing IS_MULTI_ROW_ACTION as Y and N

MULTI_ROW_ACTION_NAME: If user modified IS_MULTI_ROW_ACTION as Y specify the multi row action name like ADD_ROW,EDIT_ROW,DELETE_ROW. If it is modified to N remove multi row action name.

HOST_REQUEST_NAME : User has to provide host request name if HOST_PERSISTENCE_REQD is enabled

LIMIT_MODE: User can enable consumption or reversal limits by providing field value as C – consumption R – reversal. Same can be disabled by removing field values

LIMIT_TYPE: user can override the action behavior by providing or removing LIMIT_TYPE value.

If LIMIT_TYPE value is E limit check happens during request initiation on the field enabled for limit check in field details section of customization work sheet

If LIMIT_TYPE value is A limit check happens during request approval on the field enabled for limit check in field details section of customization work sheet

E - Entry Limits

A - Approver Limits

DYNAMIC_VIEW_PARAMS: In customization user can override the dynamic or conditional display of view like IF REQUEST_STATUS is SUC show success screen or if REQUEST_STATUS is FAL show Failure screen

ACTION_DESC: user can modify the action description

PRE_LOCAL_CUSTOM_HOOK: Pre local custom hook performs business processing before local processing method of this action if any business processing needs to be performed , user can plugin Pre-local custom hook.

POST_LOCAL_CUSTOM_HOOK: Post local custom hook performs business processing after local processing method of this action if any business processing needs to be done on local processed data, user can plugin Post-local custom hook

PRE_HOST_CUSTOM_HOOK: Pre host custom hook performs business processing before making a call to host system If any business processing to be done before making host call , user can plugin pre host custom hook

POST_HOST_CUSTOM_HOOK: Post host custom hook performs business processing after host call processing. If any processing to be done after host call. User can plugin Post host custom hook

Customizing Action validations

We will have only two button in Action_validation section of customization workbook ,one for adding a new validation, one for deleting a newly added validation.

Adding New Validation

While adding a new record that is on the click of “Adding a new action validation” new record get added with ACTION_ID field having list actions configured in action list section and VAL_ID field having list validation configured in section validations.

User can configure new validation to an action by selecting action id from action list and Val item from Val item list

Deleting new validation

User can remove new validation configured by using Delete new added link button.

Calculations Customization

Calculation section of customization workbook have 3 buttons, one for adding a new Calculation, One for deleting a newly added Calculation, one for modifying an existing Calculation.

Adding new Calculation

While adding a new record that is on the click of “Adding a New Calculation”, a new record gets added in which user can configure calculation to specific action.

Modifying Product Calculation

“Modifying an existing Calculation” button creates a copy of selected row allowing CALCULATION_REQUIRED to modify.

User can switch off the product calculation by changing CALCULATION_REQUIRED to “N”

Deleting New Calculation

“Deleting a newly added Calculation” button we can use only for deleting newly added action by mistake.

Pre_Populators Customization

Pre populator customization allows user to

- Configure new pre populator to an action
- Modifying the INPUT ,OUTPUT parameters to product populator
- Switching off product populators
- Deleting newly added populators

We will have 3 buttons in customization section of pre populators, one for adding a new Pre-Populator, One for deleting a newly added Pre-Populator, one for modifying an existing Pre-Populator.

Adding new populator

On click of “Adding a New Pre Populator” a new row will be there with all the columns value being blank.

Modifying existing populator

On click of “Modifying an Existing Pre-Populator” the selected row will be copied and the below mentioned columns will be allowed to customize.

POPULATOR_CLASS: we can override product populator with customization populator

INPUT_PARAMS: we can add or remove fields to input parameters

OUTPUT_PARAMS: we can add or remove fields to output parameters

POPULATOR_REQUIRED: In customization we can switch off product pre populators

Deleting new populator

On click of “Deleting a newly Added Pre-Populator” only the pre-populator which is newly added will be removed.

Factory Class Customization Present In Form Management Framework

This is classified as:

- RMHelperFactory
- OnlineMessagePopulatorFactory
- BTAOFactory

RMHelperFactory

If a request has to be sent for RM Approval, then the condition on which the request should be sent for RM approval can be configured in RM_APPROVAL_PARAMS column of "Form_Requests" tab in the data seed excel sheet.

By default the product is providing two helper type implementations:

- ThresholdAmount
- NotNull

Below are the key points to be noted while configuring new RMHelperFactory.

1. Configure the CustomRMHelperFactory in the AppConfig.xml for the Param name "CUSTOM_RM_HELPER_FACTORY".

A dummy implementation of CustomRMHelperFactory has been provided in AppConfig.xml.

Param name = "CUSTOM_RM_HELPER_FACTORY" value = "com.infosys.e-Banking.formmanagement.util.DummyCustomRMHelperFactory"

2. The new Helper class should extend the class AbstractRMHelperRegistry
3. The new Helper class should override the method getRMConfirmationHelper() as per the requirement and should return the helper class.

OnlineMessagePopulatorFactory

Form Management Framework provides some generic success/error messages after event processing.

Below are the key points to be noted while configuring custom Message Populator.

1. Configure the CustomMessagePopulator in the AppConfig.xml for the Param name "CUSTOM_MESSAGE_POPULATOR ". A dummy implementation of CustomMessagePopulator has been provided in AppConfig.xml

Param name = "CUSTOM_MESSAGE_POPULATOR" value = "com.infosys.e-Banking.formmanagement.util.DummyCustomMessagePopulator"

2. The new Populator class should extend the class AbstractFMMessageRegistry
3. The new Populator class should override the method getPopulatorClassName() as per the requirement and should return the MESSAGE_POPULATOR.
4. Message populators can be configured at module level.

EBTAOFactory

If a newly added form field in customization needs to be persisted into database, then we are not allowed to alter the product table specs. For this, we need to define new extended tables for customization which will be configured as the extended master table in "CUSTOM_EXT_MASTER_TABLE" and extended child table in "CUSTOM_EXT_CHILD_TABLE" in the Customization excel sheet.

So for getting the TAO instance of the custom tables, we need to customize FMTAOFactory class. A default implementation will be there for product. For Customization, follow the below key points:

1. Configure the CustomTAOFactory in the AppConfig.xml for the Param name "CUSTOM_TAO_REGISTRY". A dummy implementation of CustomTAOFactory has been provided in AppConfig.xml

Param name = "CUSTOM_TAO_REGISTRY" value = "com.infosys.e-Banking.tao.DummyCustomTAOFactory"

2. The new TAO class should extend the class AbstractFMTAORegistry

3. The new TAO class should override the method getTAOClass() as per the requirement and should return the TAO ClassName.

Customization Toolkit

A set of tools are provided to customization team to ease out customization process. Refer to documentation on Finacle eBanking workbench.

Listed below is a summary of these tools:

Visual Developer Tool

Usage: This tool is used to design PPDs and DPDLs. PDL stands for Page Definition Language.

Customer deliverable: YES

Description: This is an eclipse based UI designing tool. Using this tool it is possible to design both PPDLs and DPDLs.

AuditConfigGen tool

Usage: To generate audit scripts

Customer deliverable: YES

Description: This tool is used to scan and generate audit scripts from service spec files and formsgroup spec files so that developer need not write AUDIT related scripts. This tool would be shared with customer. In customization mode we can automate the audit script generation.

Formsgroupgen tool

Usage: To compile and create .fg files

Customer deliverable: YES

Description: This tool is used to scan and generate formsgroup Meta files from fg spec files. This tool is used for compiling new formsgroup and extended formsgroup. This tool would be shared with customer. In customization mode we may need to create new formsgroup or we may need to extend a product formsgroup.

DAL Generator tool

Usage: To create databuilder and query builder

Customer deliverable: YES

Description: This tool is used to generate data builder and query builder from DAL spec files so that developer need not write data builder and query builder. Using this tool we can provide an extension to the product DAL. This tool would be shared with customer. In customization mode we may need to extend a product DAL for fetching additional data or we may need to write a new DAL.

UI Generator tool

Usage: To create JSP files from PDL spec.

Customer deliverable: YES

Description: This tool is used to generate JSP files from PDL Spec. This tool also does css validation and it creates JSP equivalent HTMLs .This tool would be shared with customer. In customization mode we may need to modify a product PDL or we may need to write a new PDL.

Jspcustompathgen tool

Usage: To create Profilebasedjsp.properties file.

Customer deliverable: YES

Description: This tool is used to create Profilebasedjsp.properties file. This property file contains the absolute path of profile specific JSPs. This tool will be used in profile enabled environment. This tool will take absolute path of the web content folder as an input and creates Profilebasedjsp.properties file. The content of generated property file will look like one mentioned below.

Where PR1 and PR2 are profile ids. At runtime if the incoming request is profile enabled and if it finds a corresponding entry in this file, Request dispatcher will dispatch the request to this value. This way look and feel of the same JSP can be made different for various profiles.

Jasper generator tool

Usage: To create jasper files for report generation.

Customer deliverable: YES

Description: This tool is used to create Jasper files from JRXML files. JRXML files will get generated as part of JSP generation. Jasper files are used for report generation.

This tool would be shared with customer. During customization we may need to generate new JSPs or we may need to modify product JSPs. Both cases we need to have new jasper file for report generation.

TAO generator tool

Usage: To generate TAO, DAO, Info and Info key objects.

Customer deliverable: YES

Description: This tool is used to create TAO, DAO, ER, Info and Info key objects from tpsc spec. As part of customization we may need to have new tables.

This tool would be shared with customer.

TMO generator tool

Usage: To generate TMO objects.

Customer deliverable: YES

Description: This tool is used to create TMO objects from TSPC spec. As part of customization we may need to have new shadow tables.

So this tool would be shared with customer.

Primitives generator tool

Usage: To generate Primitives.

Customer deliverable: YES

Description: This tool is used to create primitives from their spec files. These primitives can be of following types:

- Binary

- Char
- Double
- Float
- Int
- Long
- String
- Text

So these different types can be generated by passing respective arguments.

These tools would be shared with customer. During customization we may need to generate new primitives.

VO generator tool

Usage: To generate Value Objects.

Customer deliverable: YES

Description: This tool is used to create Value objects from VO spec. This tool would be shared with customer. During customization we may need to generate new value objects for different reasons like creating new use cases, new services etc.

ER generator tool

Usage: To generate Entity Reference.

Customer deliverable: YES

Description: This tool is used to create Entity Reference from ER spec. This tool would be shared with customer. During customization we may need to have new table so we may have to generate new ER for that.

SR Compiler tool

Usage: To generate Service Request Meta files.

Customer deliverable: YES

Description: This tool is used to create Service Request Meta files from SR spec. This tool would be shared with customer.

UI Builder tool

Usage: To generate PDL spec from HTML spec.

Customer deliverable: YES (not used after arrival of Visual Developer)

Description: This tool is used to create PDL spec from HTML spec. By using this tool a new screen can be generated from an HTML spec.

This tool would be shared with customer until workbench is released.

BC Validator tool

Usage: To validate sections, CSS and checks Browser Compatibility.

Customer deliverable: YES

Description: This tool is used to validate sections, CSS and checks for browser compatibility. This tool would be shared with customer. During customization we may have new sections and CSS which are needed to be validated. Also we may need to check different browsers compatibilities for newly created JSPs.

DBSeedConverterTool

Usage: This tool is used to create non English database seeds.

Customer deliverable: YES

Description: For Internationalization of the product, the data in the tables for which seeds are provided must also be present in the user specific language. This tool would be shared with customer

Entity Reference

How do I create a new custom Entity Reference?

To create a custom entity reference, the tool setup should point to customization area. All the new custom entity references should be prefixed with "Custom" other wise the tool will throw an error saying that the entity references should be prefixed with "Custom". The grammar of custom specs is same as that of the product spec.

Code Alert:

```
<EntityReference xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.infosys.org FEBAEntityReferences.xsd"
  xmlns="http://www.infosys.org/EntityReferences">
  <PackageName>com.futurebank.e-Banking.types.entityreferences</PackageName>
  <ClassName>CustomCountryCodeER</ClassName>
  <RelatedDatabaseTable>COMMON_CODES:COCD</RelatedDatabaseTable>
  <Fields>
    <Field>
      <FieldName>code</FieldName>
      <FieldType>CountryCode</FieldType>
      <FieldDefaultValue></FieldDefaultValue>
      <IsKeyField>true</IsKeyField>
      <IsNullable>false</IsNullable>
    </Field>
  </Fields>
</EntityReference>
```

Above code snippet is an example of an entry in Custom Entity Reference which demonstrates the grammar of the custom spec.

Caching Framework

How do I create a new custom Cache?

A CustomAppCacheLoader will act as a new processing block required to meet some desired requirements. Its usage in a caching framework is as follows(the following usage shows a dummy implementation):

```
public class CustomAppCacheLoader {

    static {

        try {

            AppCacheRegistryManager.registerCache(

                CustomAppDataConstants.COMMONCODE_CACHE, CustomCommonCodeCache

                .getInstance());

        } catch (CriticalException e) {

            LogManager.logDebug(null, FEBAIncidenceCodes.LOAD_FOR_CACHE);

        }

    }

}
```

For using a new CustomAppCacheLoader we have to first write a Cache and this Cache is called through this Loader which is configured in AppConfig.xml against the property "CUSTOM_APP_CACHE_LOADER_CLASSNAME".

The Cache here which we used is CustomCommonCodeCache which is similar to CommonCodeCache.

Currently this property is configured with “DummyAppCacheLoader” value. This value is the fully qualified path of the CustomAppCacheLoader class. The property is configured in the following way:

```
<Param name = "CUSTOM_APP_CACHE_LOADER_CLASSNAME"
value="com.infosys.feba.framework.cache.DummyAppCacheLoader"/>
```

A new constants class file "CustomAppDataConstants" has to be created in the package "com.futurebank.feba.framework.cache.(Dummy package)" which would contain the entry for the newly created Cache:

```
public class CustomAppDataConstants {

    public static final String CUSTOM_COMMON_CODE_CACHE = "CommonCodeCache";

}
```

The constant “CUSTOM_COMMON_CODE_CACHE” will now be used in CustomAppCacheLoader class to invoke the newly written Cache for processing whenever is encountered in a custom caching.

Filtration Framework

How do I create a new custom Filter Loader?

A CustomFilterHelperLoader will act as a new processing block required to meet some desired requirements. Its usage in a filtration framework is as follows (the following usage shows a dummy implementation):

```
public class CustomFilterHelperLoader implements IFilterHelperLoader {

    public void loadHelpers(Map<String, IFilterHelper> pRegistryMap) {

        pRegistryMap.put(

            CustomFilterConstants.ENTITY_SKINCODES,

            new SkinFilterHelper());

    }

}
```

For using a new CustomFilterHelperLoader we have to first write a Filter and this Filter is called through a Loader which is configured in AppConfig.xml against the property "CUSTOM_FILTER_HELPER_LOADER_IMPL".

Currently this property is configured with "DummyFilterHelperLoader" value. This value is the fully qualified path of the CustomFilterHelperLoader class. The property is configured in the following way:

```
<Param      name      =      "CUSTOM_FILTER_HELPER_LOADER_IMPL"      value      ="com.infosys.e-
Banking.filter.DummyFilterHelperLoader"/>
```

For creating a new filter Helper, we will have to write a class which will contain all the details defining the filtration part.

FilterHelper class will do the following:

Based upon the TransactionContext or OpContext will perform the following the operations

Based upon the attribute of filter (eg. from FDET table), it will return the attribute value.

Note:

- **Package for writing the custom FilterHelper class is com.futurebank.e-Banking.filter.helper**
- **FilterHeler Class should extends AbstractEBFilterHelper**
- **User has to map the FilterHelper objects in EBFilterHelperLoader**

A new constants class file "CustomFilterConstants" has to be created in the package "com.futurebank.e-Banking.filter.(Dummy package)" which would contain the entry for the newly created Filter:

```
public class CustomFilterConstants {

    public static final String ENTITY_SKINCODES      = "SkinType";

}
```

Common Event Registry Framework

How do I create a new custom Common Event?

A Custom Common Event will act as a new processing block required to meet some desired requirements. Its usage is as follows (the following usage shows a dummy implementation).

For using a new Custom Common Event we have to first write a new Event and this is called through a Registry which is configured in AppConfig.xml against the property "CUSTOM_COMMON_EVENT_REGISTRY".

Currently this property is configured with "DummyCommonEventRegistry" value. This value is the fully qualified path of the Custom Common Event Registry class. The property is configured in the following way:

```
<Param          name          =          "CUSTOM_COMMON_EVENT_REGISTRY"    value          =
com.infosys.feba.framework.formsgroup.DummyCommonEventRegistry "/>
```

To call this new Custom Common Event we need to make its entry in:

```
public class DummyCommonEventRegistry implements ICommonEventRegistry{

    private static Map customCommonEvent=new HashMap();

    static{

        customCommonEvent.put(FEBAConstants.LOOK_UP_EVENT,"com.futurebank.custom.common.formsgrou
p.commonevent.LookUpEvent ");
    }
}
```

So firstly, in application it will check for the Custom Common Event, if it doesn't find any custom common event then it checks the Product Common Event.

Interface methods like `getCommonEventHandler(String eventId)` and `isCommonEvent(String eventId)` in "DummyCommonEventRegistry" may have same logic present in "Commoneventregistry" for Custom Common Events.

List of Abbreviations

JSP	Java Server Page
SMS	Short Messaging Service
WAP	Wireless Application Protocol
UI	User Interface
URL	Universal Resource Locator
COCD	Common Codes
STCD	State Codes
BRCM	Branch Customer
JDBC	Java Database Connectivity
IP	Internet Protocol

About Infosys Finacle

Finacle is the industry-leading universal banking solution from EdgeVerve Systems, a wholly owned subsidiary of Infosys. The solution helps financial institutions develop deeper connections with stakeholders, power continuous innovation and accelerate growth in the digital world. Today, Finacle is the choice of banks across 84 countries and serves over 547 million customers – nearly 16.5 percent of the world's adult banked population.

Finacle solutions address the core banking, e-banking, mobile banking, CRM, payments, treasury, origination, liquidity management, Islamic banking, wealth management, and analytics needs of financial institutions worldwide. Assessment of the top 1000 world banks reveals that banks powered by Finacle enjoy 50 percent higher returns on assets, 30 percent higher returns on capital, and 8.1 percent points lesser costs to income than others.



For more information, contact finacle@edgeverve.com

www.finacle.com

©2018 EdgeVerve Systems Limited (a fully owned Infosys subsidiary), Bangalore, India. All Rights Reserved. This documentation is the sole property of EdgeVerve Systems Limited ("EdgeVerve"). EdgeVerve believes the information in this document or page is accurate as of its publication date; such information is subject to change without notice. EdgeVerve acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. This document is not for general distribution and is meant for use solely by the person or entity that it has been specifically issued to and can be used for the sole purpose it is intended to be used for as communicated by EdgeVerve in writing. Except as expressly permitted by EdgeVerve in writing, neither this documentation nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior written permission of EdgeVerve and/ or any named intellectual property rights holders under this document.