Oracle FLEXCUBE Investor Servicing®
Extensibility Reference Guide

Release 12.0

April 2012

Oracle Part Number E51528-01

ORACLE

FINANCIAL SERVICES

Contents

1 Preface....3
1.1 Audience....3
1.2 Related documents....3
1.3 Conventions....4
2 Introduction....4
2.1 How to use this Guide....4
3 Extensibility Approach....4
3.1 Features....4
3.2 Layers....5
3.3 Release hierarchies....5
4 Extensible units....6
4.1 Application Server Layer....6
4.1.1 Language xml....7
4.1.2 SYS Java Script File....7
4.1.3 Kernel JavaScript File....7
4.1.4 Cluster JavaScript File....8
4.1.5 Custom JavaScript File....8
4.2 Database layer – Maintenance....8
4.2.1 Function ID Main Package....8
4.2.2 Hook Packages....9
4.2.3 Kernel Package....11
4.2.4 Cluster Package....12
4.2.5 Custom Package....12
4.3 Database layer – Bypassing base functionality....12
4.4 Database layer – Online....12
5 Contract online extensibility....13
5.1 Message flow....13
5.2 Handlers list....**Error! Bookmark not defined.**

1 Preface

This document describes the approach to FLEXCUBE extensibility and acts as reference for a various handlers provided for extensibility.

1.1 Audience

This document is intended for FLEXCUBE Application Developers/Users who are authorized to perform the following tasks:

- Modify the layouts of existing FLEXCUBE Screens
- Modify the existing functionality by adding new fields/tabs/data blocks
- Extend the existing screen to have fields based on customer specific table/fields
- Add customer specific validations at extension hooks
- Add customer specific processing logics in batch processing
- Add customer specific notifications
- Add customer specific calculation elements
- Add customer specific reports

To Use this manual, you need conceptual and working knowledge of the below:

Proficiency	*Resources*
FLEXCUBE IS Development overview	FCIS-FD01-01-01-Development Overview Guide
RAD function ID development getting started	FCIS-FD02-01-01-RAD Getting Started

1.2 Related documents

For more information on RAD development and extensibility, refer the below documents:

- *FCIS-FD03-01-01-Extensibility Getting started*
- *FCIS-FD01-01-01-Development Overview Guide*
- *FCIS-FD02-02-01-RAD Function ID Development*
- *FCIS-FD02-03-01-RAD Web Service Development*
- *FCIS-FD02-04-01-RAD BIP Report Integration*
- *FCIS-FD02-05-01-RAD Notification Development*
- *FCIS-FD05-02-01-RAD-Reference*
- *FCIS-FD03-03-01-Extensibility By Example Volume 1*
- *FCIS-FD03-03-02-Extensibility By Example Volume 2*
- *FLEXCUBE_IS_Extensibility_Framework.doc*
- *FCIS-FD04-02-01-Generic Interface Configuration Guide*
- *FCIS-FD04-03-01-Upload Adapter Development Guide*

1.3 Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements (for example, menus and menu items, buttons, tabs, dialog controls), including options that you select.
italic	italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates language and syntax elements, directory and file names, URLs, text that appears on the screen, or text that you enter.

2 Introduction

FLEXCUBE IS base product development is performed by Kernel team and the units that are developed are called as Kernel software units. Other teams that requires the product extensions are required to use the "extension units" applicable for respective teams.

Product extension required for the following teams:

- Cluster release teams
- Customer release teams
- Partners/Customers

2.1 How to use this Guide

This document contains the below chapters describing the approach of extensibility in different areas of the system.

- Chapter 3, "Extensibility Approach"
- Chapter 4, "Extensible Units"
- Chapter 5, "Contract Online Extensibility"

3 Extensibility Approach

This section describes the various extensibility features, layers that impact the extensibility and release hierarchies involved.

3.1 Features

FLEXCUBE IS provides following additional handlers in the system:

- Contract Operation data base units

These units are used to extend the FLEXCUBE module specific contract online operations.

- Interest & Charges SDE Definition

  These SDE components used to define specific IC interest calculation rule.

- Consumer Lending SDE Definition

  These SDE components used to define specific CL module interest calculation

- Message Generation

  This feature used to define customer specific advice tags and populate the tag during runtime.

- Maintenance of User Defined Fields at screen level

  UDF feature is used to define the additional fields required for extensibility to capture extra data

- Generation of notifications for operations done in the system

  Notifications are the specific event related messages to be propagated to external systems.

3.2 Layers

FLEXCUBE IS provides handlers at the following layers to extensibility teams to extend the business logic:

- **Screen extensibility**

  Screen extensibility is provided to add data blocks , fields and other graphical elements buttons, LOVs to the screens. Extensibility design also helps upgrade of the extended logic in further release of FLEXCUBE IS.

- **Screen – Java script extensibility**:

  Java script files extensibility provides 'Pre' and 'Post' handlers to add the code at logical stages in front end processing.

- **Back End Units**:

  Database extensibility provides 'Pre' and 'Post' handlers to add code at logical stage in back end processing

3.3 Release hierarchies

To enable extensibility, Oracle FLEXCUBE identifies the release type both during design and in runtime thereby restricting the development teams to add business logic in designated units only. This is to ensure the development teams of different release types use corresponding units to add business logic.

Below are the release types Oracle FLEXCUBE identifies and supports in extensible mode:

- **Kernel**: Oracle FLEXCUBE base product release
- **Cluster**: Customized base for a specific region or a specific functionality

- **Custom**: Customized release for customers

Kernel is the main product release and Cluster releases are made using Kernel as the base to develop Cluster specific functionality. This Cluster release can be further enhanced based on the customer specific requirements to develop a Custom release.

In such case, hierarchy of Release types would be as below:
Kernel → Cluster → Custom

In some cases where the final set of requirements are not very different from Kernel release or if there are not many common requirements across the customers of a particular region, Kernel itself will be taken as base for Custom releases.

In such case, hierarchy of Release types would be as below:
Kernel → Custom

In all these cases, it is required for the Kernel release to provide place holders for adding additional business logic both in Cluster and Custom releases.

Oracle FLEXCUBE will be enhanced to support extensibility in the below areas:

- Screen Design
- Front End Scripting
- Code Generator
- Back End PL/SQL Programming

The approach is to divide the programs (Java Script and PL/SQL Packages) into several logical stages and to provide 'Pre' and 'Post' handlers to Customization teams.

4 Extensible units

There are basically the following four types of screens in Oracle FLEXCUBE IS:

- **Maintenance**: These screens are typically used to maintain static data used across the system. These screens include product definition function as well.
- **Online**: These screens are typically used to capture contract related data. Any operations related to contracts are performed in these screens.
- **Batch**: These screens are used to initiate Oracle FLEXCUBE Batch End or Day batch operations.
- **Reports**: These screens are used to capture data required to generate a BI Publisher canned reports.

4.1 Application Server Layer

As a part of RAD function ID generation, following units are generated for application layer:

- RAD XML
- Language / UI XML
- Java Script files
    - SYS JS files
    - Kernel JS files
    - Cluster JS files
    - Custom JS files

4.1.1 Language xml

Language XML file, also called as UIXML is generated by RAD tool during function ID (screen) development. This file is contains following elements:

- Screens
- Sections and Partitions
- Blocks
- Field sets
- Fields and their properties

During run time, XSL Transformation is applied to this XML file by linking it to an XSL file. This result in screen rendering at the browser.

4.1.2 SYS Java Script File

As a part of Function ID development, RAD tool generates the SYS Java script files. These SYS JavaScript file mainly contains a list of pre declared variables:

- msgxml: - This variable is used by the system to build FCIS Request XML
- dataSrcLocationArray: - This variable is an array of DATA BLOCKS
- relationArray:-This array contains relation and relation type details of blocks.
- Databinding
- retflds and bndFlds:- These arrays contains LOV information
- CallFormArray, CallFormRelat, CallRelatType:- These arrays contains callform details, call form relation and relation type
- actionsAmmendArray: - This array contains information for enabling fields based on actions

4.1.3 Kernel JavaScript File

As a part of Function ID development, RAD tool generates the Kernel Java script files. These Javascript file allows developer to add functional code and is specific to KERNEL release. The functions in this file are generally triggered by screen events. A developer working in kernel release would add functions based on two categories:

- Functions triggered by screen loading events
  Eg: fnPreLoad_KERNEL(),fnPostLoad_KERNEL()
- Functions triggered by screen action events
  Eg: fnPreNew_ KERNEL (),fnPostNew_ KERNEL ()

4.1.4 Cluster JavaScript File

As a part of Function ID development, RAD tool generates the Cluster Java script files. These Javascript file allows developer to add functional code and is specific to CLUSTER release. The functions in this file are generally triggered by screen events. A developer working in CLUSTER release would add functions based on two categories:

- Functions triggered by screen loading events
  Eg: fnPreLoad_CLUSTER(),fnPostLoad_CLUSTER()
- Functions triggered by screen action events
  Eg: fnPreNew_ CLUSTER (),fnPostNew_ CLUSTER ()

In case if any function in KERNEL javascript file has to be modified,this can be achieved by overriding the function in CLUSTER javascript file.

4.1.5 Custom JavaScript File

As a part of Function ID development, RAD tool generates the Custom Java script files. These java script file allows developer to add functional code and is specific to CUSTOM release. The functions in this file are generally triggered by screen events. A developer working in CUSTOM release would add functions based on two categories:

- Functions triggered by screen loading events
  Eg: fnPreLoad_CUSTOM(),fnPostLoad_CUSTOM()
- Functions triggered by screen action events
  Eg: fnPreNew_ CUSTOM (),fnPostNew_ CUSTOM ()

In case if any function either in KERNEL javascript file or CLUSTER javascript file has to be modified,this can be achieved by overriding the respective function in CUSTOM javascript file

4.2 Database layer – Maintenance

As a part of function ID development, RAD generates following database packages:

- Function ID MAIN Package
- Hook Packages
  - KERNEL Package
  - CLUSTER Package
  - CUSTOM Package

4.2.1 Function ID Main Package

The Main Package contains the basic validations and backend logic for the Maintenance function id. The Main package contains the mandatory checks required. It will also contain function calls to the other packages generated by RAD.

The main package has the below stages:

- Converting Ts to PL/SQL Composite Type
- Checking for mandatory fields

- Defaulting and validating the data
- Writing into Database
- Querying the Data from database
- Converting the Modified Composite Type again to TS

Each of these stages has a 'Pre' and 'Post' hooks in the Kernel, Cluster and Custom Packages. These Hooks are called from the Main Package itself. Main Package has the system-generated code and should not be modified by the developer Kernel, Cluster and Custom Packages are the packages where the respective team can add business logic in appropriate functions using the Pre and Post hooks available.

4.2.2 Hook Packages

The Main Package has designated calls to these Hook Packages for executing any functional checks and Business validations added by the user. The structure for all the Hook Packages are the same, like:

- Fn_Post_Build_Type_Structure
- Fn_Pre_Check_Mandatory
- Fn_Post_Check_Mandatory
- Fn_Pre_Default_and_Validate
- Fn_Post_Default_and_Validate
- Fn_Pre_Upload_Db
- Fn_Post_Upload_Db
- Fn_Pre_Query
- Fn_Post_Query

These Functions are called from the Main package using the Pre and Post Hooks available in the Main Package. The 3 Hook Packages namely Kernel, Cluster and Custom Packages have similar structure and are for the respective teams to work on.

In the Table SMTB_PARAMETERS, the parameter RELEASE_TYPE indicates the deployed release. The system uses this flag to determine the hooks to be called. Depending on the deployed release type system skips calling these hooks.

For examples if the deployed release is Kernel, Cluster and Custom hooks need not be called. Similarly in case the deployed release type is Cluster, system does not call custom hook as it is not needed.

The Complete Flow for a sample function, say Fn_Check_Mandatory is as follows:

- STPKS_STDCIF_MAIN. Fn_Check_Mandatory
- STPKS_STDCIF_CUSTOM.Fn_Pre_Check_Mandatory
- STPKS_STDCIF_CLUSTER.Fn_Pre_Check_Mandatory
- STPKS_STDCIF_KERNEL.Fn_Pre_Check_Mandatory
- STPKS_STDCIF_MAIN .Fn_Sys_Check_Mandatory
- STPKS_STDCIF_KERNEL.Fn_Post_Check_Mandatory
- STPKS_STDCIF_CLUSTER.Fn_ Post_Check_Mandatory
- STPKS_STDCIF_CUSTOM.Fn_ Post_Check_Mandatory

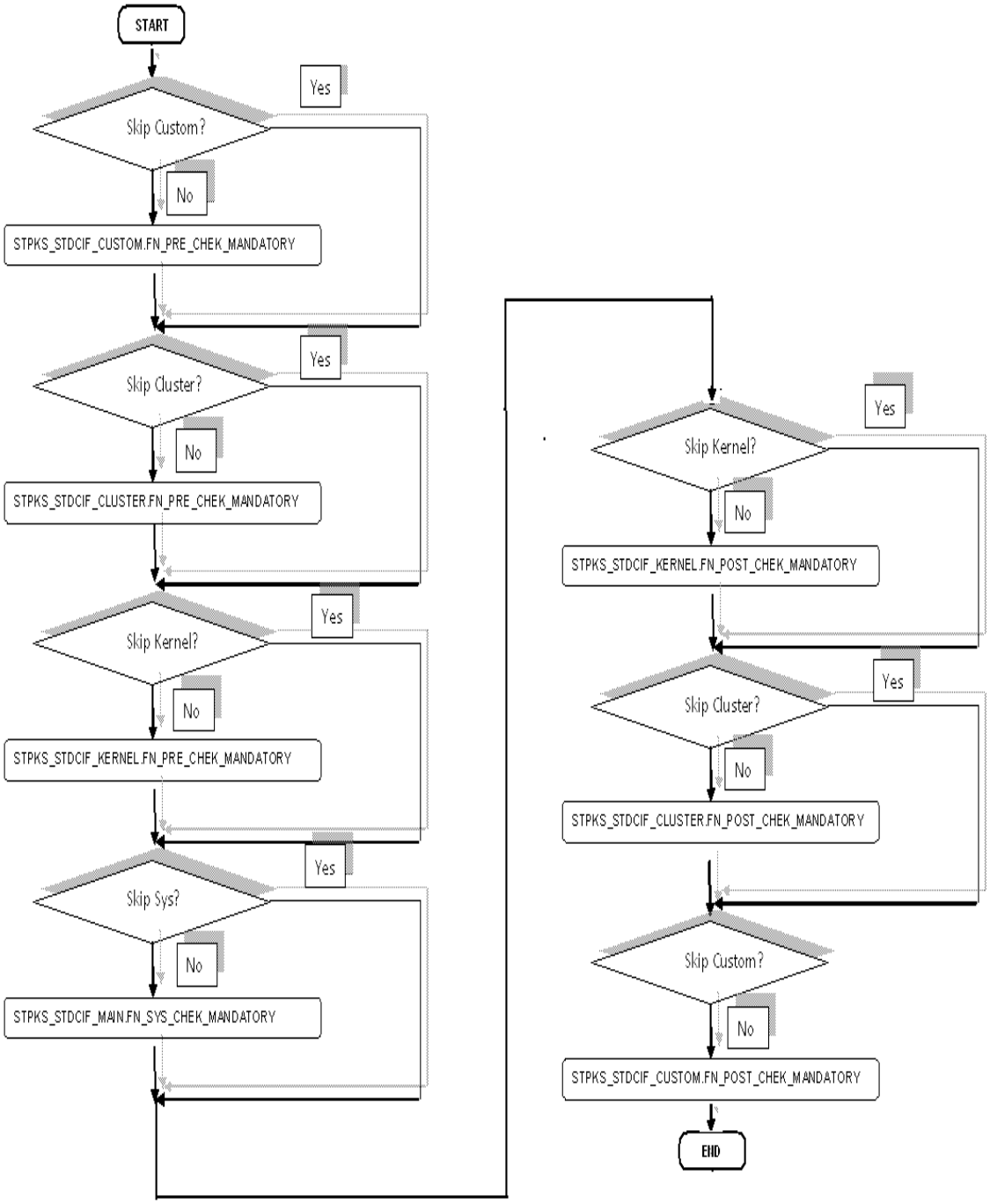
There are auto generated functions like FN_SKIP_<RELEAE_TYPE> which would determine whether or not a particular hooks needs to be called.

Developer also has an option to bypass the base release hook if need be. For example if the validations written in STPKS_STDCIF_Kernel.FN_PRE_CHECK_MANDATORY are not required or not suitable for the Cluster release, system provides an option to bypass the code written by Kernel team.

Similarly a Custom release can also bypass the code written by Kernel and Custom Releases. This can be achieved by calling procedures PR_SET_SKIP_<RELEASE_TYPE> and PR_SET_ACTIVATE_<RELEASETYPE>. These procedures will be made available in the main package and the development teams of Customization teams can use these procedures to skip and re-activate the hooks of parent release.

The Developer should avoid adding validations or Checks in the Pre Stage of any function, like Fn_Pre_Check_Mandatory, etc and should aim to add all the validations in the Fn_Post_Default_and_Validate.

For Example let us see the flow for the Mandatory Stage for STDCIF:



4.2.3 **Kernel Package**

The Kernel package is solely for the Kernel Team to modify. The Main package has designated calls to the Kernel package for executing any functional checks or validations included in the Kernel Package. All the user level validations and conditional operations should be included in Fn_Post_Default_and_Validate. This function is called from the Main Package after the execution of Fn_Default_and_Validate. User should avoid putting validations or code in any other function.

In case user needs to add a separate function, the existing RAD generated structure should not be changed. Instead the user can create a new package e.g. STPKS_STDCIF_UTILS package. The desired function can be included in this package and the call can be made from the Kernel Package.

4.2.4 Cluster Package

The Cluster package is available to the Cluster Team to add any validations or Checks specific to the Cluster Team over and above the Kernel Team. The Kernel Team or the Custom Team should not modify the contents of this package.

4.2.5 Custom Package

The Custom package is available to the Custom Team only to add any validations or Checks over and above those already present in the Kernel and Cluster Packages.

4.3 Database layer – Bypassing base functionality

In cases where the functionality of child release, either cluster or custom like to override base functionality, there might be a need to skip the base functionality. RAD Generated code provides handlers to this as well and the kernel functionality can be skipped from Cluster and kernel/cluster can be skipped from custom releases.

For Example, let us say that the business logic in the function STPKS_STDCIF_KERNEL.Fn_Pre_Default_and_Validate is contradicting the business logic for Cluster, then the user has the option to skip the validation present in the Kernel. For this the user needs to call PR_SET_SKIP_KERNEL. After it bypasses, the user again needs to activate this flag by calling PR_SET_ACTIVATE_KERNEL. Else all the following functions in KERNEL will be bypassed.

Once the Skip is set in cluster and again activated, it skips both the functions in kernel namely, STPKS_STDCIF_KERNEL.Fn_Pre_Default_and_Validate and STPKS_STDCIF_KERNEL.Fn_Post_Default_and_Validate. If the requirement is that only the validations and logic in STPKS_STDCIF_KERNEL.Fn_Pre_Default_and_Validate be skipped then the other function STPKS_STDCIF_KERNEL.Fn_Post_Default_and_Validate needs to be called explicitly from the Cluster Package.

Similarly from Custom Package the validations in Kernel as well as Cluster can be bypassed.

4.4 Database layer – Online

The Backend Units generated by RAD for Online Functions are:

- Main Package
- Kernel Package
- Cluster package

- Custom Package
- Services package

The Basic structure and function of the Main, Kernel, Cluster and Custom Packages are the same as in case of Maintenance Functions with the addition of a few extra functions specific to Online Functions like FN_ENRICH, FN_PRODUCT_DEFAULT.

For Online functions, RAD generates an extra package called the Services Package. The naming convention followed by the services package is <Module>pks_<Function id>_SERVICES.

All the user specific business validations and checks need to be added in the services package. Some of the Functions in the Services package are:

- Fn_Resolve_Ref_Numbers
- Fn_Check_Mandatory
- Fn_Default_and_Validate
- Fn_Upload_Db
- Fn_Query
- Fn_Process
- Fn_Product_Default
- Fn_Subsys_Pickup
- Fn_Enrich
- Fn_Unlock

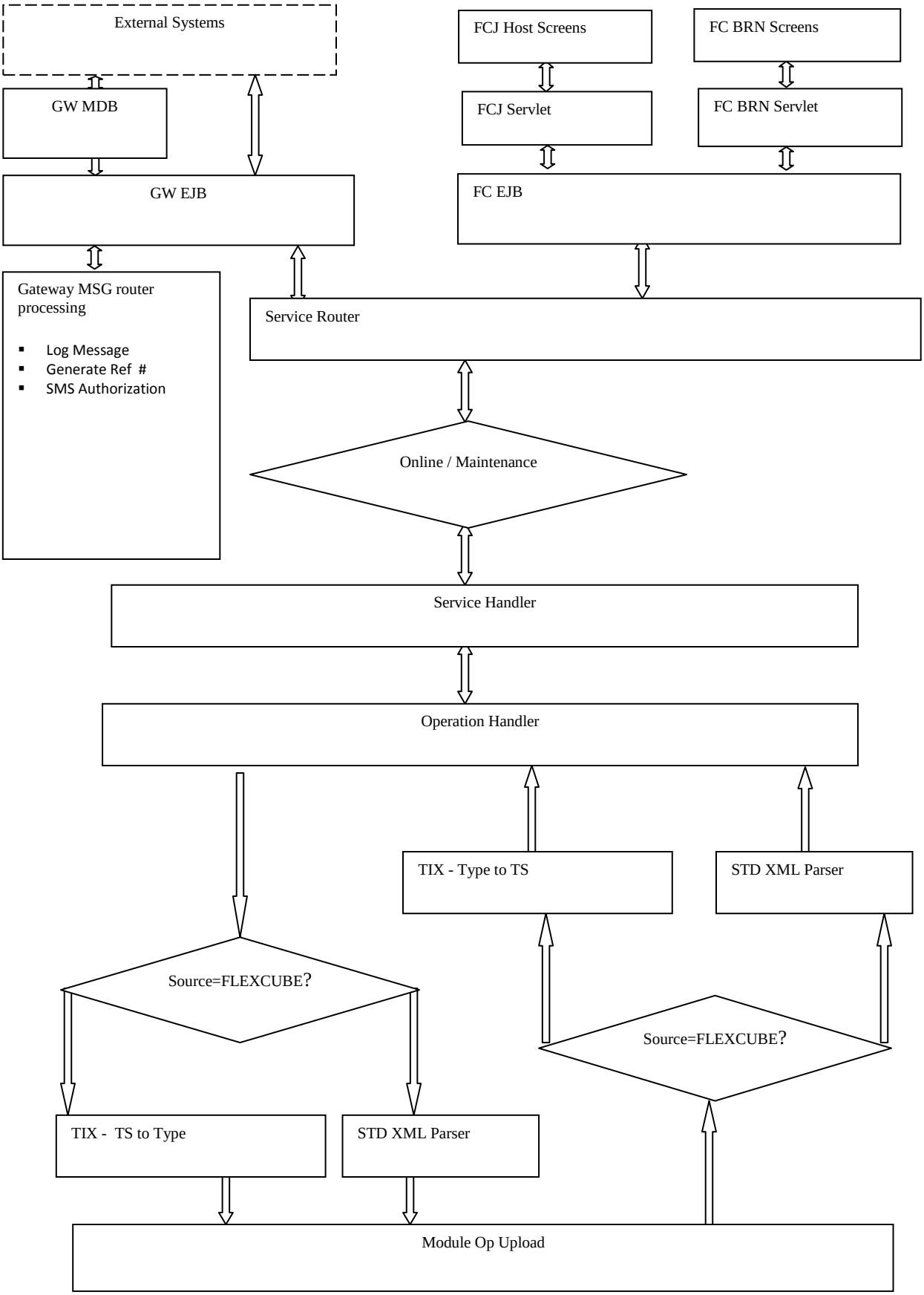
Online extensibility is further explained in section 5.

5 Contract online extensibility

5.1 Message flow

Typically, the usage of contract online screens tends to be highest and these are the screens where some additional defaulting or validations are requested. To address these requirements, it is envisaged that additional handlers are provided in contract online screens. These handlers need to be provided at individual operation level to increase the flexibility.

To understand the additional handlers, it is important to first know the processing logic of online operations in the back-end. Whenever, any operation is done in a contract online screen, the system builds the data in an XML format and sends the request to the back-end. The requests are then parsed in the database and the existing routines are invoked to process the requests. On successful processing, a response message is built with enriched data and sent to the front-end. The following schematic highlights the various stages in an online request processing:


External Systems
FCJ Host Screens
FC BRN Screens
GW MDB
FCJ Servlet
FC BRN Servlet
GW EJB
FC EJB
Gateway MSG router processing
Log Message
Generate Ref #
SMS Authorization
Service Router
Online / Maintenance
Service Handler
Operation Handler
TIX - Type to TS
STD XML Parser
Source=FLEXCUBE?
Source=FLEXCUBE?
TIX - TS to Type
STD XML Parser
Module Op Upload

The following is the detailed flow for processing any request related to a contract online operation in the database:

- Oracle FLEXCUBE New UI middleware (General EJB) invokes the service router (gwpks_service_router) for every request from a contract online screen.
- Each request contains the function id and the action. Based on these details, the system derives the service name and the operation that is being processed. For this, the system refers to GWTM_FCJ_FUNCTIONS. Any subsequent processing of the message is performed based on the service name and the operation.
- There is a specific service handler for each module in Oracle FLEXCUBE. When a request is received, the system identifies the service handler based on the service name derived and invokes the required service handler package. The logic for this available in gwpks_service_router
- Thereafter the control is handed off to an operations handler based on the operation.

For example, when a request to amend an existing contract is received from Money Market Contract Online screen, the function is passed as MMDCONON and the action code is sent as AMEND. Based on these, the system derives the service name as FCUBSMMService and the action code as ModifyMMContract

The service handler for processing Money Market contract online operations is Gwpks_Mmservice.Pr_Service_Handler. The system invokes this package from gwpks_service_router. The invocation of this function is done as follows

```
IF p_Rec_Msg_Header.Service_Name = 'FCUBSMMService'
THEN
                Gwpks_Mmservice.Pr_Service_Handler(p_Is_In_Msg_Clob
                                                   ,p_In_Msg_Str
                                                   ,p_In_Msg_Clob
                                                   ,p_Rec_Msg_Header
                                                   ,p_Is_Out_Msg_Clob
                                                   ,p_Out_Msg_Str
                                                   ,p_Out_Msg_Clob
                                                   ,p_Instr_Rec
                                                   ,p_Err_Code
                                                   ,p_Err_Param);

 END IF;
```

The control is then routed to the operation handler using the following logic

```
IF p_rec_msg_header.operation_code = ' ModifyMMContract'
THEN
      dbg('About to call the function
Gwpks_ModifyMMContract.pr_process_msg');
      gwpks_modifymmcontract.pr_process_msg(p_is_in_msg_clob,
                                            p_in_msg_str,
                                            p_in_msg_clob,
                                            p_rec_msg_header,
                                            p_is_out_msg_clob,
                                            p_out_msg_str,
                                            p_out_msg_clob,
```

```
                                                p_instr_rec,
                                                p_flag,
                                                p_err_code,
                                                p_err_param);
 END IF;
```

The operation handlers then carryout the following tasks:

- Convert the message from XML to tilde separated
- Build PL/SQL type variables based on the tilde separated list of values
- Invoke the module specific processing function to process the request
- Validate the response received from the module processing function. In case the response is an error, then build an error message by appending the errors to the request message received.
- In case the processing is successful, then build a response message with an enriched data

In order to enable extensibility, additional handlers have been provided before and after every operation. These handlers are made available in a new package. These packages do not contain any additional logic in the Kernel version. As part of customizations, these handlers can be populated with the required logic.

For example, the following handlers are available for Money Market amendment operation

```
FUNCTION fn_pre_modifymmcontract
( p_rec_msg_header    IN   gwpkss_service_router.ty_biz_process_header
                           ,p_instr_rec            IN OUT
                           gwpkss_service_router.ty_processing_instructio
                        ns
                           ,p_ty_mmcont_details            IN OUT
                           ldtbs_upload_master%ROWTYPE
                           ,p_ty_settlements                IN OUT
                           ispks_upload.ty_tbl_istbs_upld
                           ,p_ty_rec_misdetails             IN OUT
                           mitbs_upload_class_mapping%ROWTYPE
                           ,p_ty_schds_details              IN OUT
                           ldpks_create_contract.ty_tb_contract_schedets
                           ,p_ty_linkages                   IN OUT
                           ldpks_create_contract.ty_tb_linkages
                           ,p_tb_hol_ccy_master            IN OUT
                           ldpks_create_contract.ty_hol_ccy_master
                           ,p_tbl_upl_charges               IN OUT
                           cfpks_upload.ty_tbl_upl_charges
                           ,p_ty_interests                  IN OUT
                           cfpks_upload.ty_tbl_upl_interest
                           ,p_ty_upl_taxrule                IN OUT
                           tapks_upload.typ_tax_rule
                           ,p_tbl_udf_det                   IN OUT
                           uvpks_udf_upload.ty_upl_cont_udf
                           ,p_ty_brktxnbook                 IN OUT
                           brtbs_upload_maintxn%ROWTYPE
```

```
                              ,p_tb_setbs_upl_repo_blks  IN OUT
                              ldpks_create_contract.tb_setbs_upl_repo_blk
                              ,p_tb_setb_upl_revrepo_dets IN OUT
                              ldpks_create_contract.ty_tb_setb_upl_revrepo_d
                            ets
                              ,p_ty_split_tag_details_upld IN OUT
                              ldpks_split_stlmts_upload_type.ty_split_tag_de
                            tails_upld
                              ,p_tbl_upl_advice                IN OUT
                              mspks_upload_advices.ty_advice_details
                              ,p_tb_relationship               IN OUT
                              cspks_utils.ty_tbl_relation_upld
                              ,l_contract_ref_no               IN OUT
                              cstb_contract.contract_ref_no%TYPE
                              p_err_code                 IN OUT VARCHAR2
                            ,p_err_param               IN OUT VARCHAR2
) RETURN BOOLEAN AS
FUNCTION fn_post_modifymmcontract
(
   ……..
) RETURN BOOLEAN AS
```

These handlers are available as part of the package GWPKS_EXT_MMSERVICE. Currently, these functions do not contain any processing logic. If any default mechanism is required or additional validations are to be performed, then they should be built in these handlers as part of the customization.

Similar to this, each contract operation has been provided with pre and post handlers. These handlers can be used to do some specific customizations. The following table gives the list of extensible handlers available for various contract operations in Oracle FLEXCUBE IS.

FCIS-FD03-02-01-Extensibility Reference Guide
April 2012
12.0

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com/ financial_services/

Copyright © 2012 - Oracle Financial Services Software Limited. All rights reserved.

No part of this work may be reproduced, stored in a retrieval system, adopted or transmitted in any form or by any means, electronic, mechanical, photographic, graphic, optic recording or otherwise, translated in any language or computer language, without the prior written permission of Oracle Financial Services Software Limited.

Due care has been taken to make this document FCIS-FD03-02-01-Extensibility Reference Guide and accompanying software package as accurate as possible. However, Oracle Financial Services Software Limited makes no representation or warranties with respect to the contents hereof and shall not be responsible for any loss or damage caused to the user by the direct or indirect use of FCIS-FD03-02-01-Extensibility Reference Guide and the accompanying Software System. Furthermore, Oracle Financial Services Software Limited reserves the right to alter, modify or otherwise change in any manner the content hereof, without obligation of Oracle Financial Services Software Limited to notify any person of such revision or changes.

All company and product names are trademarks of the respective companies with which they are associated.