Extensibility Reference Guide
Oracle Banking Payments
Release 14.4.0.0.0
[May 2020]

Contents

1 Preface

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

1.1 Audience

This document is intended for Oracle Banking Payments application Developers/Users who are authorized to perform the following tasks:

- Modify the layouts of existing Oracle Banking Payments 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

1.2 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

Oracle Banking Payments 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”

3 Extensibility Approach

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

3.1 Features

Oracle Banking Payments provides following additional handlers in the system:

- Contract Operation data base units

  These units are used to extend the Oracle Banking Payments module specific contract online operations.

- Maintenance of User Defined Fields at screen level

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

3.2 Layers

Oracle Banking Payments 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 Banking UBS.

- **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 Banking Payments 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 Banking Payments identifies and supports in extensible mode:

- **Kernel**: Oracle Banking Payments 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 Banking 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 Banking Payments:

- **Maintenance**: These screens are typically used to maintain static data used across the system. These screens include product definition function as well.
- **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 results 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 FCUBS 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_STDCIFCR_MAIN. Fn_Check_Mandatory
- STPKS_STDCIFCR_CUSTOM.Fn_Pre_Check_Mandatory
- STPKS_STDCIFCR_CLUSTER.Fn_Pre_Check_Mandatory
- STPKS_STDCIFCR_KERNEL.Fn_Pre_Check_Mandatory
- STPKS_STDCIFCR_MAIN .Fn_Sys_Check_Mandatory
- STPKS_STDCIFCR_KERNEL.Fn_Post_Check_Mandatory
- STPKS_STDCIFCR_CLUSTER.Fn_ Post_Check_Mandatory
- STPKS_STDCIFCR_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_STDCIFCR_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.

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_STDCIFCR_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_STDCIFCR_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_STDCIFCR_KERNEL.Fn_Pre_Default_and_Validate and STPKS_STDCIFCR_KERNEL.Fn_Post_Default_and_Validate. If the requirement is that only the validations and logic in STPKS_STDCIFCR_KERNEL.Fn_Pre_Default_and_Validate be skipped then the other

function STPKS_STDCIFCR_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.

Oracle Banking Payments Extensibility Reference Guide
[May 2020]
Version 14.4.0.0.0

Oracle Financial Services Software Limited
Oracle Park
Off Western Express Highway
Goregaon (East)
Mumbai, Maharashtra 400 063
India

Worldwide Inquiries:
Phone: +91 22 6718 3000
Fax:+91 22 6718 3001
www.oracle.com/financialservices/