Oracle® FLEXCUBE Investor Servicing

Development of Online Forms

Release 14.8.0.0.0
G32133-02
April 2025

Oracle FLEXCUBE Investor Servicing Development of Online Forms, Release 14.8.0.0.0

G32133-02

Contents

Preface

Purpose i
Audience i
Documentation Accessibility ii
Critical Patches ii
Diversity and Inclusion ii
Conventions ii
Screenshot Disclaimer iii
Prerequisite iii
Related Resources iii

1 Overview of Online Form

2 Screen Development

2.1 Header Information 1
2.2 Preferences 3
2.3 Data Sources 4
2.4 Data Blocks 5
2.5 Screens 8
2.6 Field Sets 10
2.7 Actions 10
2.8 Launch Forms 11
2.9 Call Forms 12
2.9.1 Sub System Pickup/Processing 13
2.10 Summary 14
2.11 Preview 15

3 Attach Call Form to Main Function Id

4 Generated Units

4.1 Front End Units 1
4.1.1 Language xml 1
4.1.2 SYS JavaScript File 1
4.1.3 Release Type Specific JavaScript File 1
4.2 Data Base Units 2
4.2.1 Static Scripts 2
4.2.2 System Packages 2
4.2.3 Hook Packages 3
4.3 Other Units 3
4.3.1 Xsd 3

5 Extensible Development

5.1 Extensibility in JavaScript Coding 1
5.2 Extensibility in Backend Coding 1

Preface

Oracle FLEXCUBE Investor Servicing is a comprehensive mutual funds automation software from Oracle® Financial Servicing Software Ltd.©.

You can use the system to achieve optimum automation of all your mutual fund investor servicing processes, as it provides guidelines for specific tasks, descriptions of various features and processes, and general information.

This topic contains the following sub-topics:

- Purpose
- Audience
- Documentation Accessibility
- Critical Patches
- Diversity and Inclusion
- Conventions
- Screenshot Disclaimer
- Prerequisite
- Related Resources

Purpose

This manual is designed to help FLEXCUBE Application developers/users to familiarize with ORACLE FLEXCUBE Development Workbench for Investor Servicing.

Audience

This document is intended for FLEXCUBE Application developers/users that use Development Workbench to develop various FLEXCUBE components.

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

Table 1 Proficiency and Resources

Proficiency	Resources
FLEXCUBE Functional Architecture	Training programs from Oracle Financial Software Services.
FLEXCUBE Technical Architecture	Training programs from Oracle Financial Software Services.
FLEXCUBE Object Naming Conventions	Development Overview Guide

Table 1 (Cont.) Proficiency and Resources

Proficiency	Resources
Working knowledge of Web based Applications	Self-Acquired
Working knowledge of Oracle Database	Oracle Documentations
Working knowledge of PLSQL developer	Respective vendor documents
Working knowledge of PLSQL and SQL Language	Self-Acquired
Working knowledge of XML files	Self-Acquired

Documentation Accessibility

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

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Critical Patches

Oracle advises customers to get all their security vulnerability information from the Oracle Critical Patch Update Advisory, which is available at Critical Patches, Security Alerts and Bulletins. All critical patches should be applied in a timely manner to ensure effective security, as strongly recommended by Oracle Software Security Assurance.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Screenshot Disclaimer

Personal information used in the interface or documents is dummy and does not exist in the real world. It is only for reference purposes.

Prerequisite

Specify **User ID** and **Password**, and log in to **Home Screen**.

Related Resources

The functions of ORACLE FLEXCUBE Development Workbench for Investor Servicing system is organized into various guides, each discussing a component.

For more information, see these Open Development Tool documents:

- *Open Development Tool Installation*
- *Development Workbench - Getting Started*
- *Development Workbench - Administration*
- *Development Workbench - Screen Development I*
- *Development Workbench - Screen Development II*
- *Development Workbench - Screen Customizer*
- *Development Workbench - Notifications*
- *Development Workbench - Bulk Generation*
- *Development Workbench - Source Upgrade*
- *Development Workbench - Tracking Changes*
- *Child and Screen Childs - Concept and Design*
- *Development of Maintenance Form*
- *Development of Online Form*
- *Development of Call Form*
- *Development of Launch Forms and Other Screens*
- *Development of Dashboard Form*
- *Development Workbench Service XML Development*
- *Development Workbench Performance Tuning Enhancements*
- *Development Workbench - Rest Services Development*

1

Overview of Online Form

This topic provides an overview on Online Forms.

Online Forms are function Id’s (screens) which is used for creating Contracts for respective modules. Same contracts can be processed further for Payments, Availments, Amendments, Reassignments and Authorizations also using Online forms.

All the transaction processing in FLEXCUBE is carried out through Online screens Online form screens should be launched independently.

On launching the Online form screen, user has to input the respective values to create the contract. Form may have the different user-defined actions like Product-Default, Enrich, and Subsystem-Pickup while creating contract. Once all the user-defined actions performed finally user has to save the contract.

ExampleLetter Of Credit (LC) contract
An LC contract is an instruction wherein a customer requests the bank to issue, advise, or confirm a credit letter for a trade transaction. An LC substitutes a bank’s name and credit for the parties involved. The bank thus undertakes to pay the seller/beneficiary even if the remitter fails to pay.

Thus for each module, we should develop different function Id’s for creating contracts and other online forms for other operations like Payments, Availments, Amendments, Reassignments, and Authorizations.

Below are the list of LC contract function ID's:

- LCDTRONL - Contract Input
- LCDAMEND - Amend Confirmation Input
- LCDAVMNT - Availment Input
- LCDTRPAY - Payment Input
- LCDTRANF - Transfer Input
- LCDEPMNT - Manual Liquidation Input
- LCDTREAS - Contract Reassign
- LCDTRAUT - Amend Confirmation Input

> **Note**
>
> The above function ID's are given as an example only.

2

Screen Development

The design and development of an Online Form function id are similar to any other function Ids.

This topic describes the following sub-topics:

- Function Generation
- Preferences

For more information, refer to the topic *Development Workbench - Screen Development I*.

- Header Information
  This topic describes about defining the header information for Online Forms.
- Preferences
  This topic describes about defining the preferences for Online Forms.
- Data Sources
  This topic describes about defining the data sources for Online Forms.
- Data Blocks
  This topic describes about defining the data blocks for Online Forms.
- Screens
  This topic describes about designing the screens for Call Forms.
- Field Sets
  This topic describes about defining the field sets for Online Forms.
- Actions
  This topic describes about the actions screen for Online Forms.
- Launch Forms
  This topic describes about the launch forms.
- Call Forms
  This topic describes about the call forms.
- Summary
  This topic describes about the summary screen.
- Preview
  This topic describes about the preview.

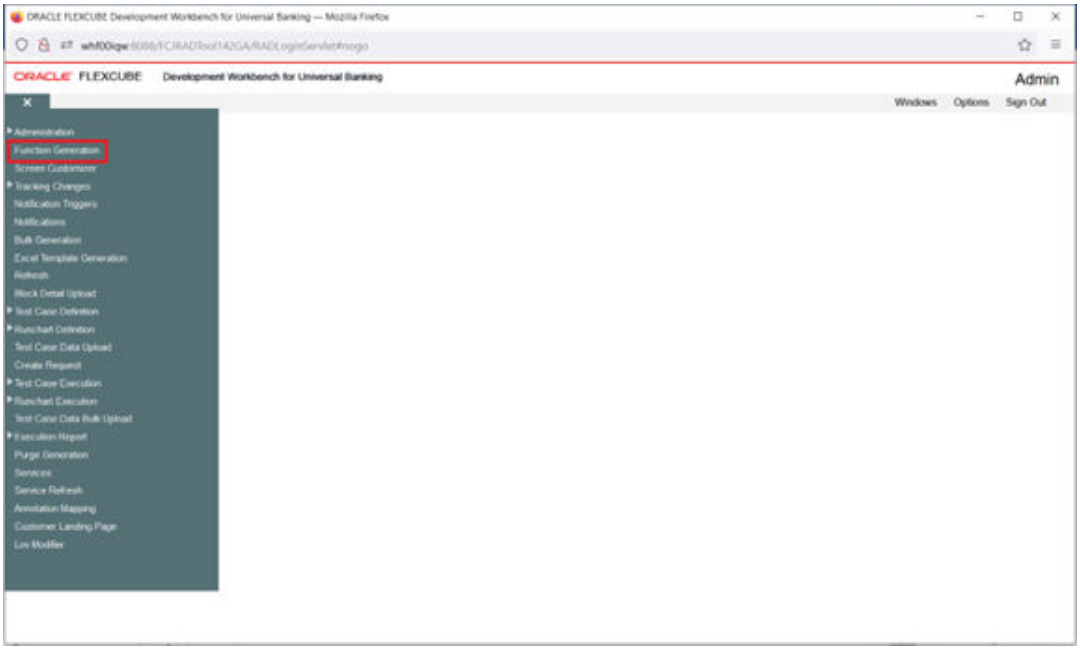
2.1 Header Information

This topic describes about defining the header information for Online Forms.

1. On **Expand Menu** of the Development Workbench for Universal Banking, click **Function Generation** node.

   The **Function Generation** screen displays.

Figure 2-1 Function Generation



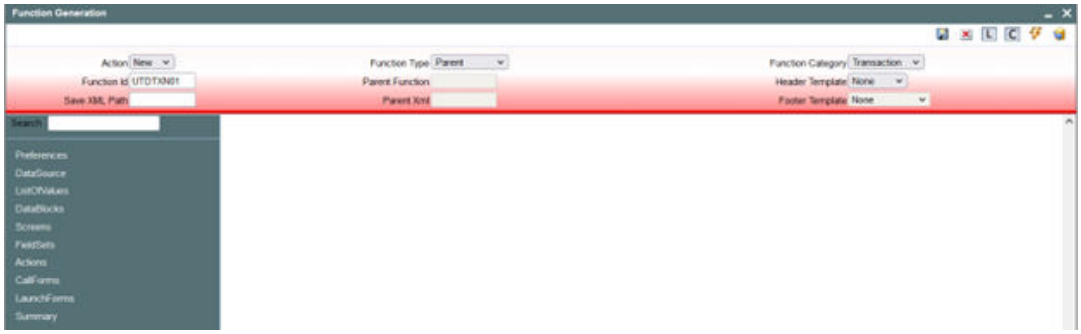
2. On the **Function Generation** screen, specify the following fields in the **Header** section for **Online Forms**.

   For more information on fields, refer to the field description table.

Table 2-1 Function Generation - Field Description

Field	Description
Function ID	It is the name of the Online Form. Online Form name has to have the third character as **D**. Ideally, the length of thename should be 8 characters Example: UTDTXN01 etc. are valid online form names.
Function Category	It is the Online Form Category. It has to be **Transaction**.
Footer Template	Select the footer template from the drop-down list. • **None** • **Maint Audit** • **Maint Process** • **Process** Footer template can be provided as required. Note for **Transaction** screens, footer template has to be selected as **None**. System does not provide any default template for transaction screens; hence developer has to design thefooter portion of the screen manually. Developer has to make sure that footer designedhas generic fields like transaction status (TXNSTAT), authorization status (AUTHSTAT) etc., For Online Process Flow Screens footer template should be selected as **Process**.
Function Type	Parent and child functionality is supported for online forms.

Figure 2-2 Online Form header Information

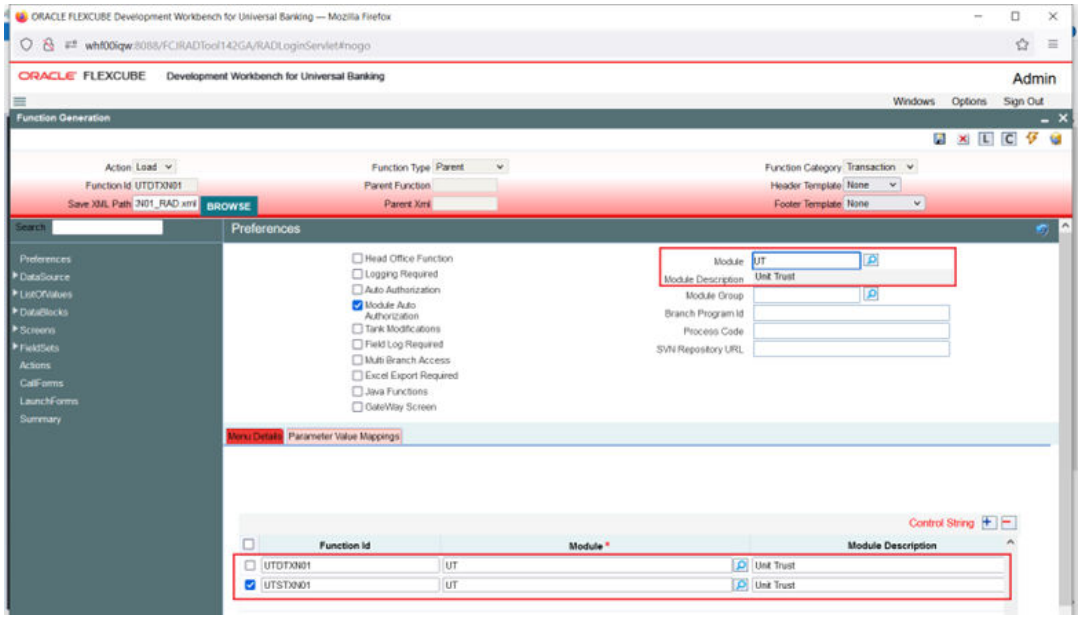


2.2 Preferences

This topic describes about defining the preferences for Online Forms.

1. Specify the menu details in the **Preferences** screen.

Figure 2-3 Online Form Preferences



2. On the **Preferences** screen, specify the following fields in the **Header** section for **Online Forms**.

   For more information on fields, refer to the field description table.

Table 2-2 Preferences - Field Description

Field	Description
Module	Module name is a mandatory field and has to be provided. It is recommended that the first two letters of the function id is kept as same as the module name. Naming of the generated package will be derived from the module code maintained.

3. Script for the following tables will be generated by Workbench (menu details) which are essential for launching of an Online screen.
   - SMTB_MENU
   - SMTB_FCC_FCJ_MAPPING
   - SMTB_FUNCTION_DESCRIPTION
   - SMTB_ROLE_DETAILS

   Type string of the Online screens will be generated as **O** in smtb_menu table.
4. Transaction specific action codes has to checked in the control string whichever applicable.

   Example: LIQUIDATE, ROLLOVER, REVERSAL etc.

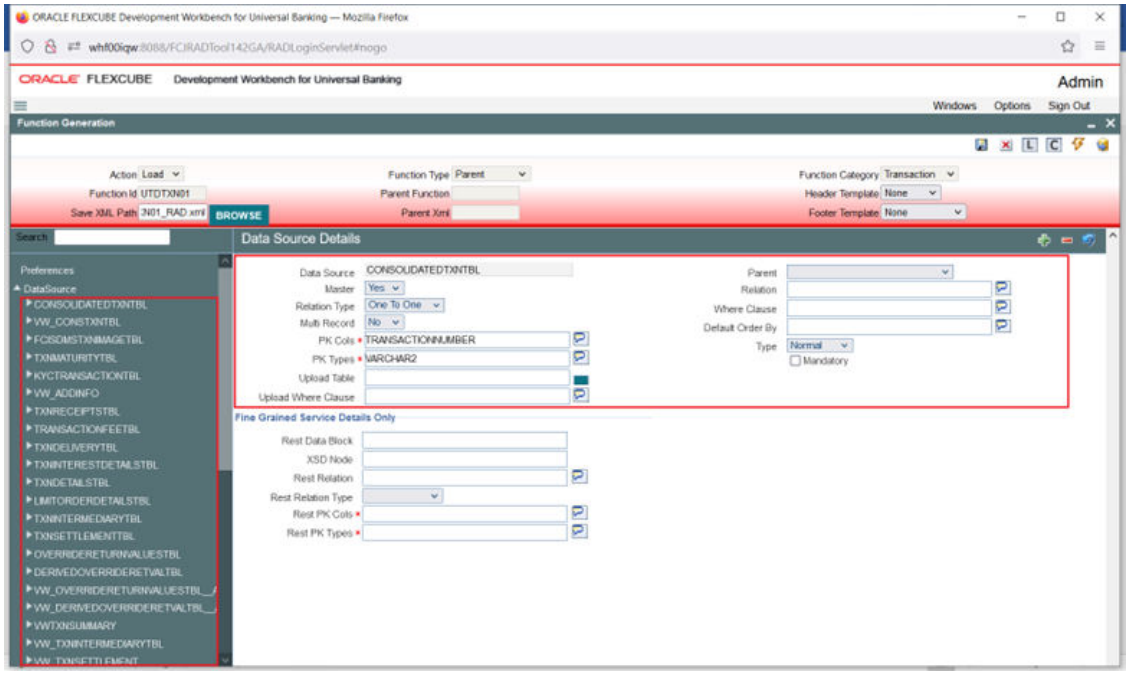
2.3 Data Sources

This topic describes about defining the data sources for Online Forms.

On the **Function Generation** screen menu, click **DataSource**.

Identify the tables/views for the Online form. Define data sources and add data source fields as required.

Figure 2-4 Adding data sources and maintaining properties

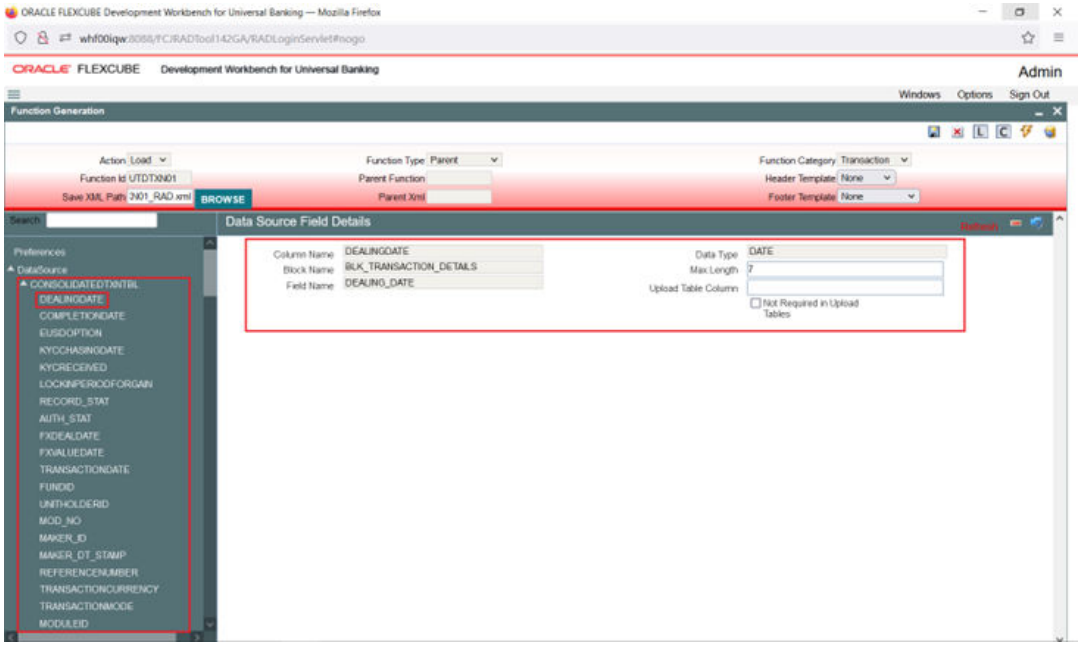


Note the following while creating data sources:

- Master Data Source has to be a single entry data source.
- Logical Relationships has to be maintained for all data sources except the parent.
- Provide **PK Cols** and **PK Types** for all data sources. If the data source is a multi-record block, then make sure it has at least one more pk than its parent which helps to identify each record of multi-record block uniquely.

- Minimize the use of views in the data sources. For transaction screens, system generated upload logic (fn_sys_upload_db) is not called within the system package. It is up to the developer to decide whether the system generated code can be used or not. If views are used in data sources, then this function should not be used by the developer.
- Usually for Online forms, a separate view can be used for summary purpose. This view will have all the fields required to be displayed in the summary. Example: **UTDTXN01_SUMMARY**.

Figure 2-5 Adding data sources fields and its properties



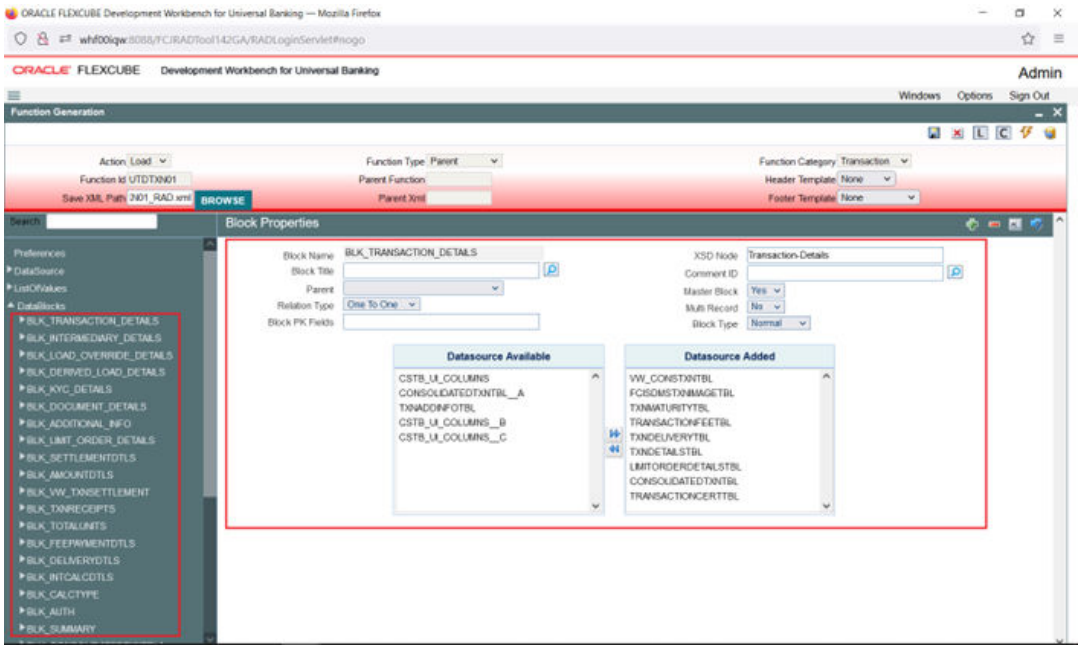
> **Note**
>
> Max length of the data source field can be modified as per requirement.

2.4 Data Blocks

This topic describes about defining the data blocks for Online Forms.

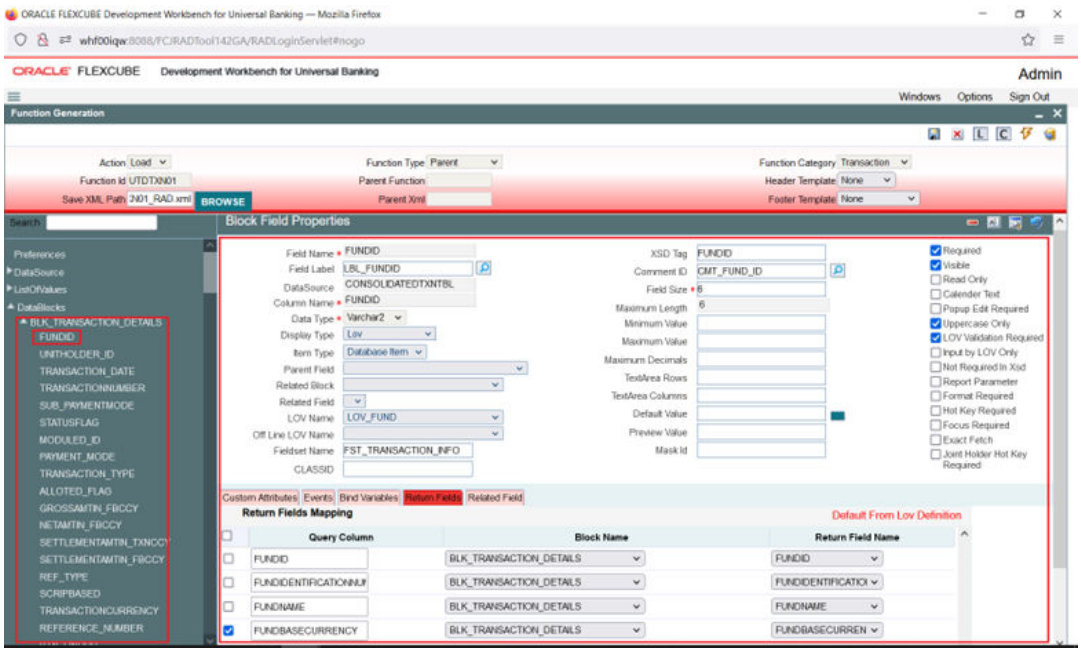
1. Determine the block structure for the function id. Define Data Blocks as per the design in the **Block Properties** screen.

Figure 2-6 Defining Data Blocks and maintaining its properties



2. Master Data Block has to be a single entry data block.
3. Provide **XSD Node** name if the block is normal and is required in gateway request.

Figure 2-7 Attaching Block Fields and maintaining its properties



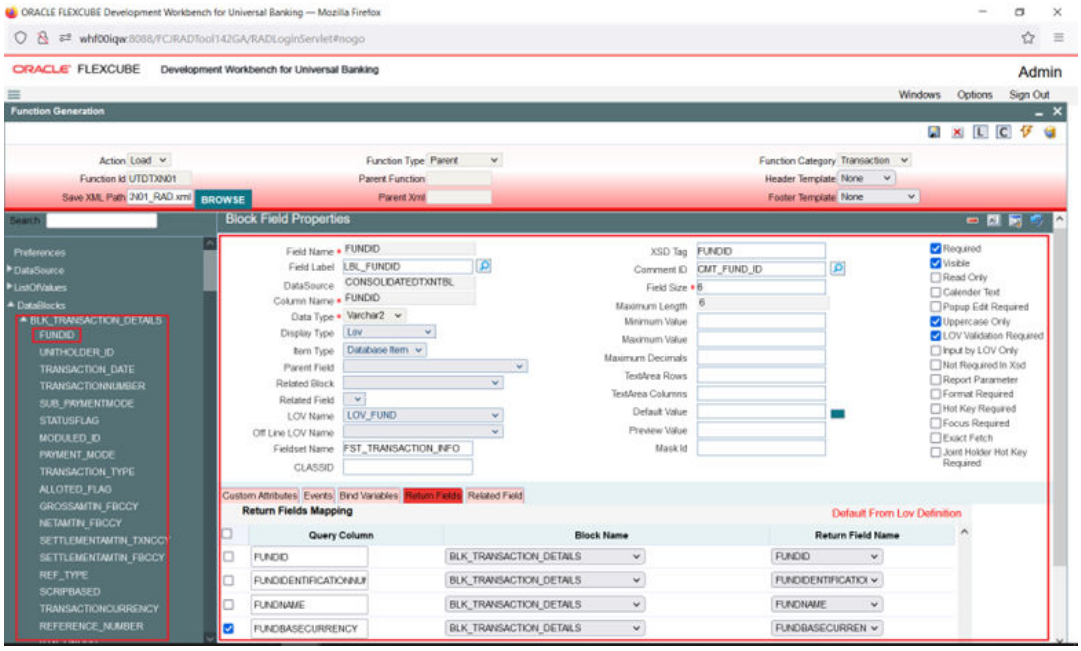
4. Block order and block field order can be changed by rearranging blocks and block fields in the browser tree (drag and drop).

> **Note**
>
> Note that all units will have to be regenerated if block or block field order is changed (including xsd's).

5. Related currency fields should be placed above the amount field in the tree.

   Add block fields to the data block as required.

Figure 2-8 Attaching Block Fields and maintaining its properties

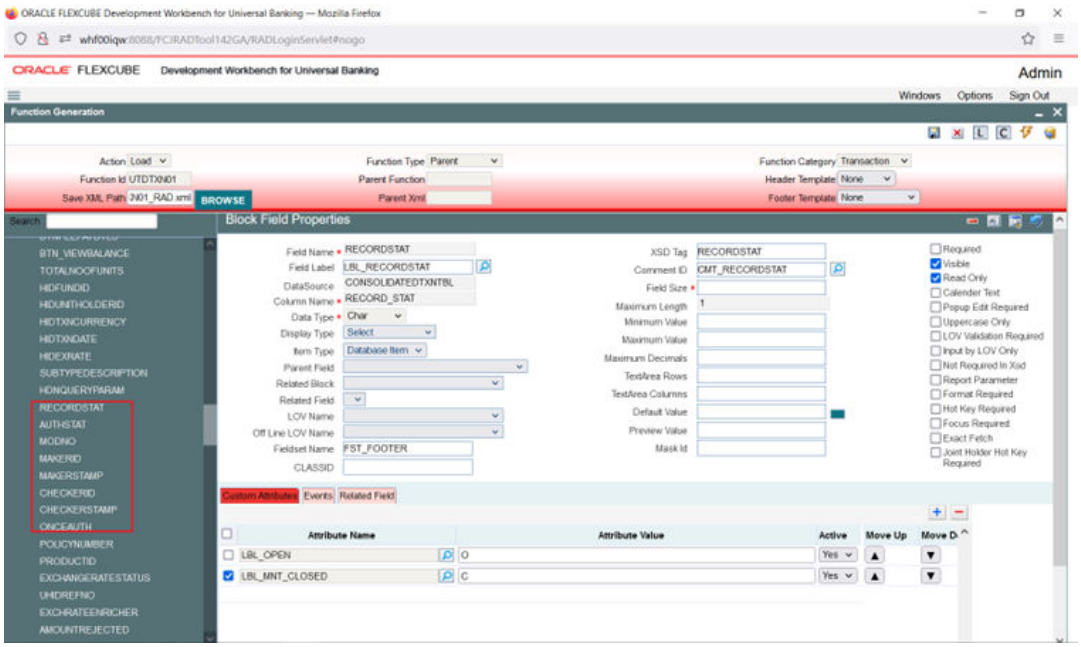


6. In case the block is not required in XSD, select the **Not Required in XSD** checkbox.
7. Ensure that Related Block and Related Field are given for Amount Fields.
8. Minimize the use of query data sources by using DESC fields wherever possible.

> **Note**
>
> Query data sources is rarely required for a Online Form screen; as launch form can be used for query only screens.

9. Master block should contain reserved field names like AUTHSTAT, RECORDSTAT, ONCEAUTH, MODNO, MAKERID, CHECKERID, MAKERDTSTAMP and CHECKERDTSTAMP are added as part of the footer of the screen.

Figure 2-9 Master Block

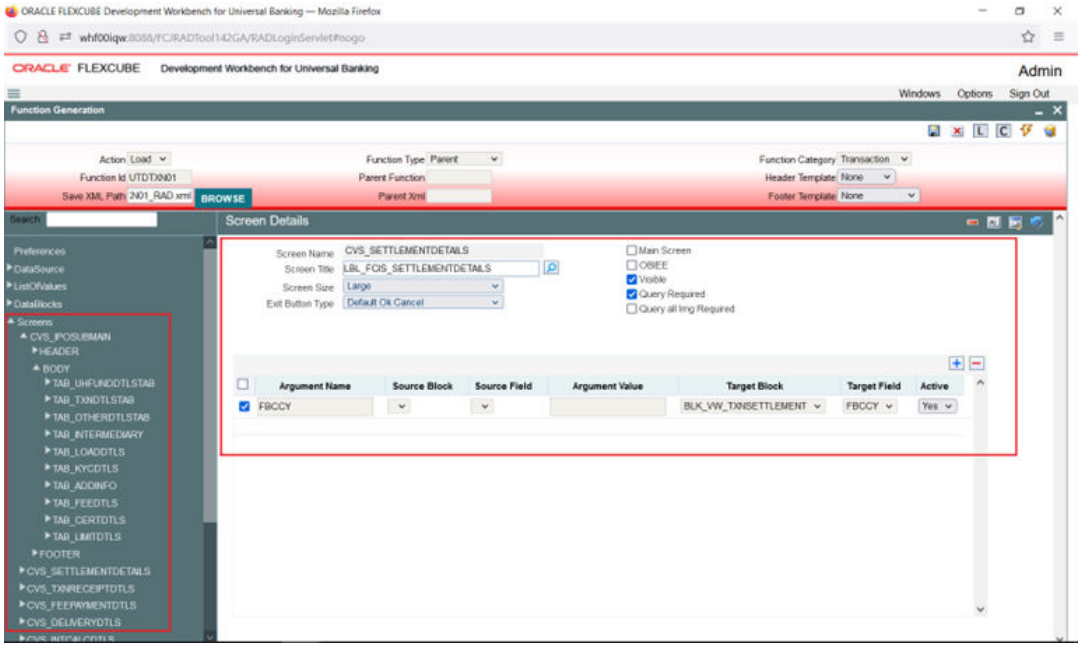


2.5 Screens

This topic describes about designing the screens for Call Forms.

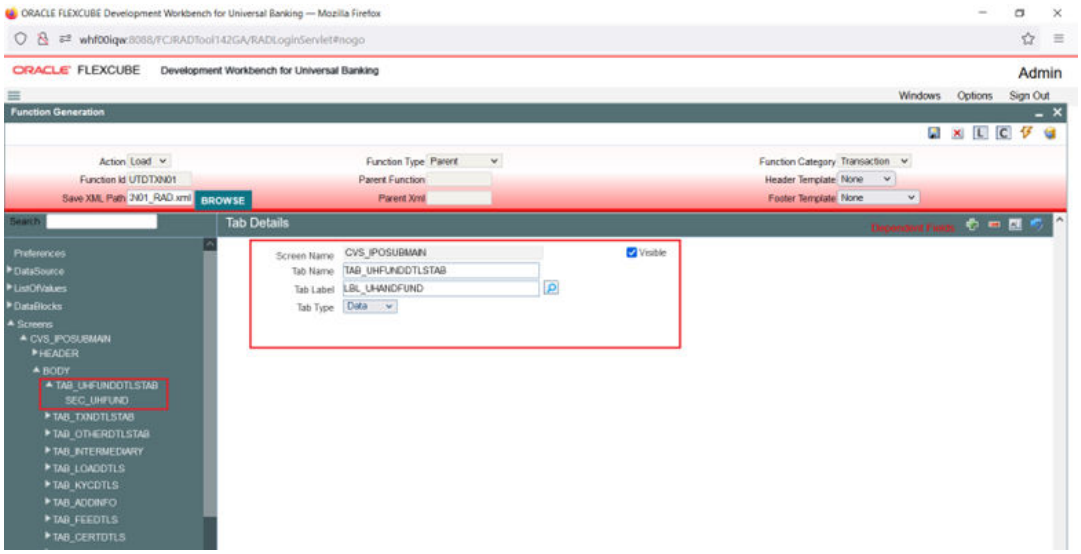
1. Design the screen layout based on the requirement in the **Screen Details**.

Figure 2-10 Designing Screens and providing Screen Properties



2. Identify one screen as the main screen; if multiple screens are present.
3. Add Tabs, sections and partitions as per the screen design.

Figure 2-11 Creating Tabs and maintaining Properties



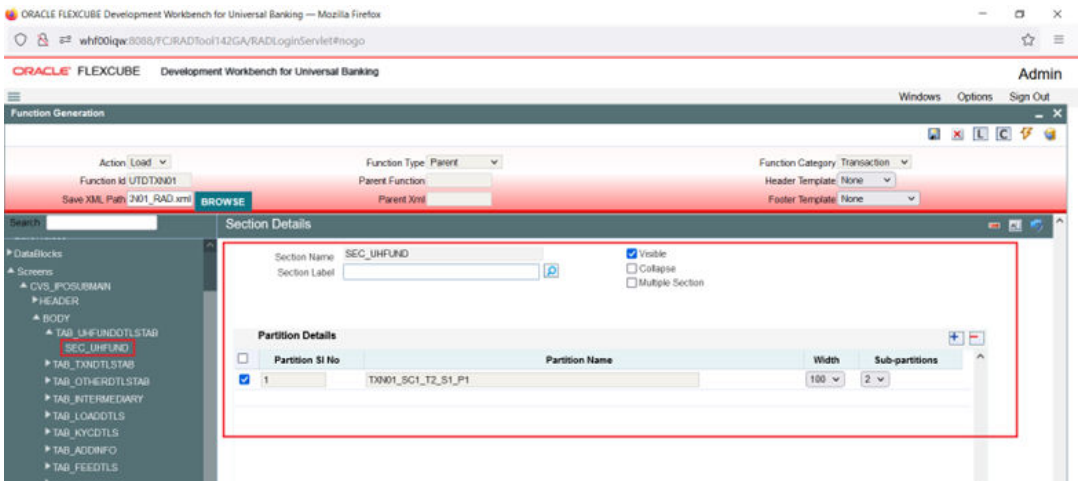
4. When creating tabs and sections for the screen, if the screen does not have multiple tabs, then only the TAB_MAIN needs to be used. TAB_HEADER should not contain any sections in this scenario.
5. Use all the tabs for Online forms. Since online forms are large screens with multiple tabs.

   TAB_HEADER should contain the header information. TAB_MAIN should be the first tab in the body. Other tabs have to be added in the body portion as required.
6. Provide sections in TAB_FOOTER as required.

   The developer often designs footers for Online forms.

> **Note**
>
> In large screens, footer supports four partitions while other portions support three partitions.

Figure 2-12 Section Properties



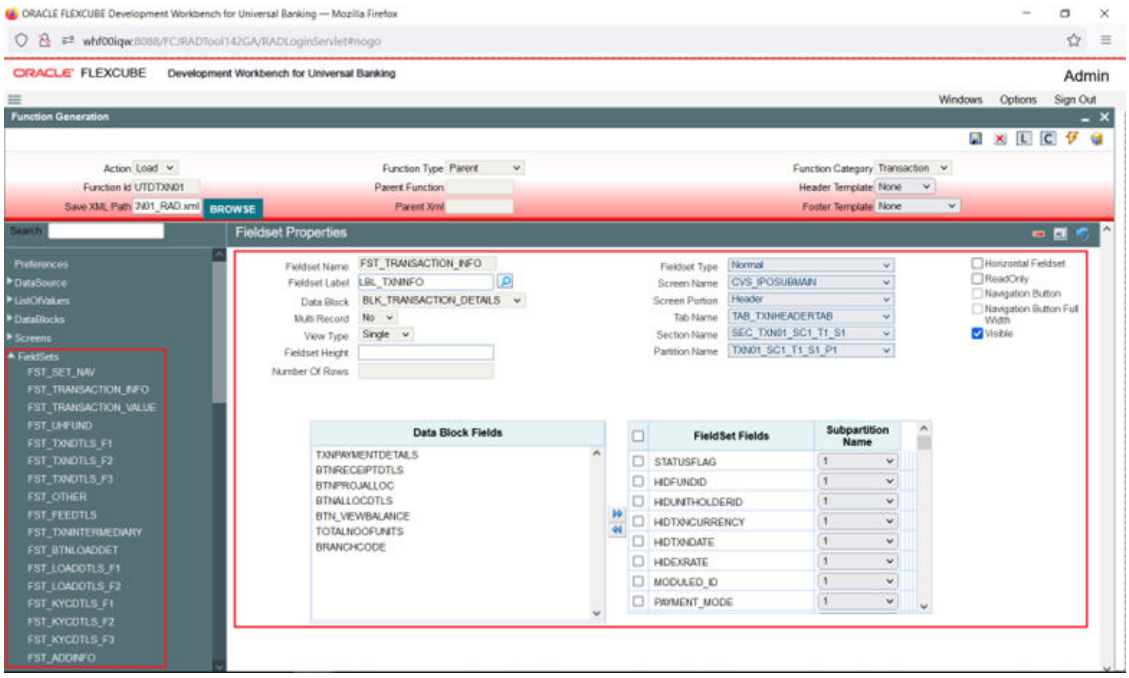
Multiple Screens can be designed if required.

2.6 Field Sets

This topic describes about defining the field sets for Online Forms.

In the **Fieldset Properties** screen, create Fieldsets and attach the fields to the field sets as required.

Figure 2-13 Field Sets



Note the following when attaching field to a field set:

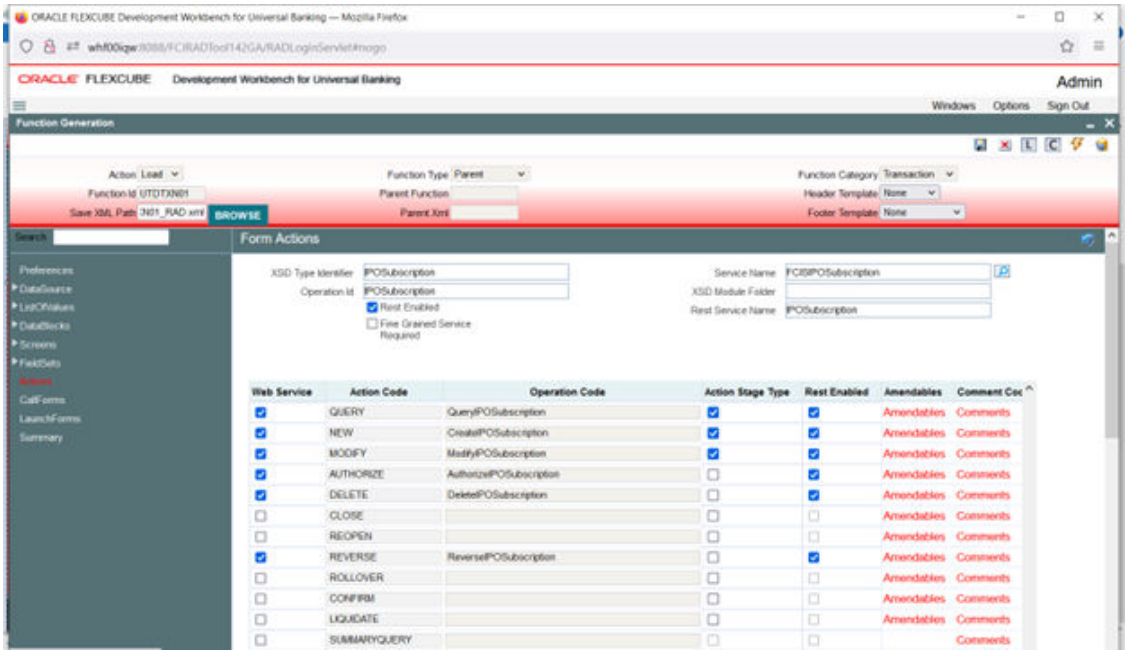
If a field is not required in the screen, but kept as hidden and value defaulted; then **The field has to be made invisible and attached to a field set**. If it is not attached to any fields set, the screen html won't contain the field and may result in script error while accessing the field.

2.7 Actions

This topic describes about the actions screen for Online Forms.

Mention the web service and amendable information in the **Form Actions** screen.

Figure 2-14 Actions Screen



Create Fieldsets and attach the fields to the field sets as required.

Mention the web service and amendable information in Actions Screen.

While maintaining web services and amendable information, note the following:

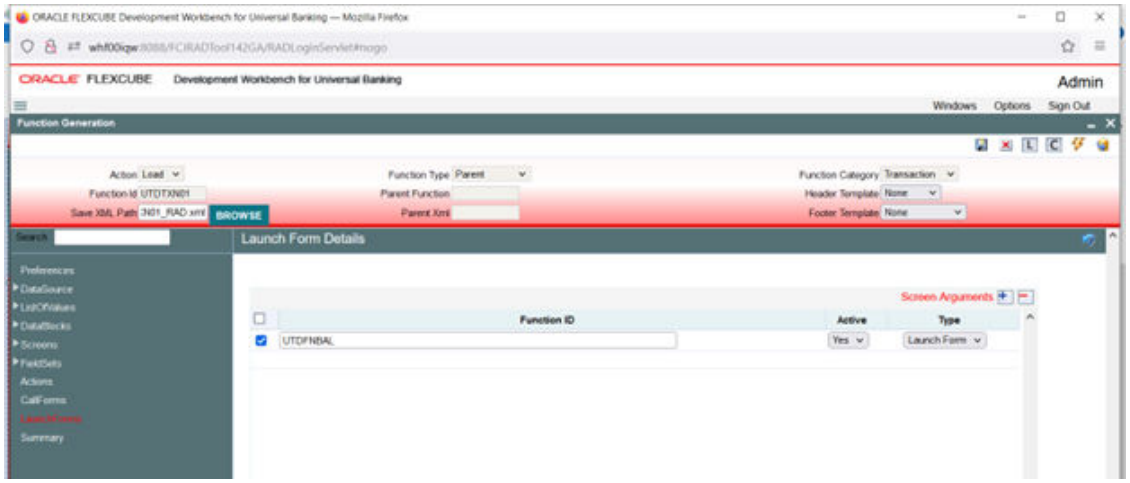
- Online forms will generate Type XSD and Message XSD. Operation-specific message xsd's will be generated. For example, name of the xsd generated will be:
  - LC-Contract-Types.xsd (Type XSD for LC Contract)
  - LC-CreateContract-Req-Full-MSG.xsd (Create Message XSD for LC Contract)
  - LC-CreateContract-Req-IO-MSG.xsd (Create Message XSD for LC Contract)
  - LC-CreateContract-Res-Full-MSG.xsd (Create Message XSD for LC Contract)
  - LC-CreateContract-Res-PK-MSG.xsd (Create Message XSD for LC Contract)
- Operation Id and Operation Code need to be maintained for the above reasons.
- Amendable information has to be maintained similar to any other function ids.

2.8 Launch Forms

This topic describes about the launch forms.

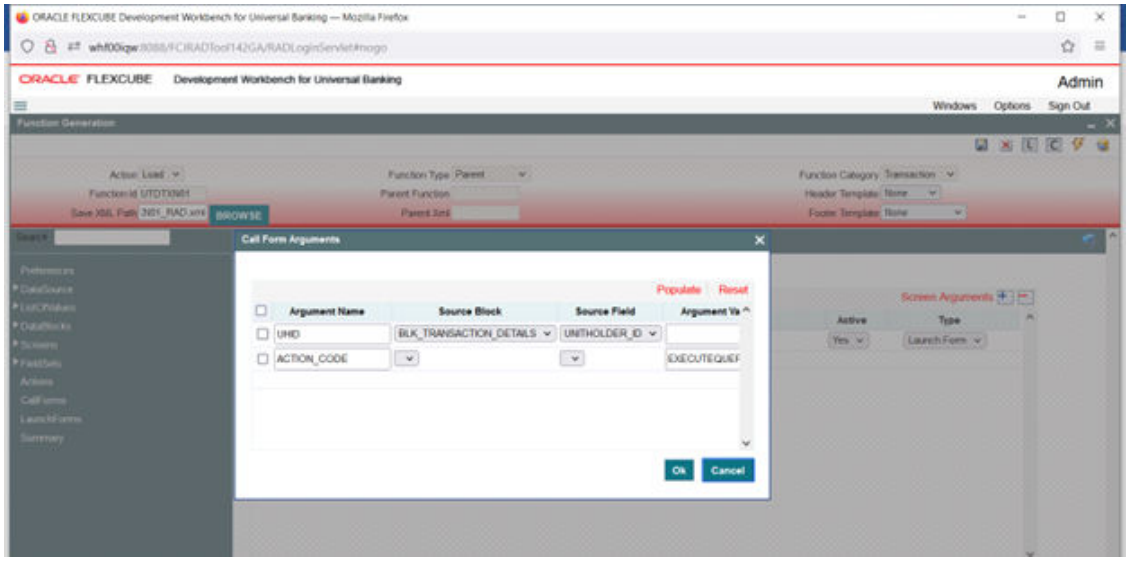
Launch Forms can be attached to Online form screen.

Figure 2-15 Launch Forms can be attached to Online form screen



Screen Arguments should be maintained for the launch form to query the proper contract record from the main online functions.

Figure 2-16 Process to attach launch forms is similar to any other function Id's

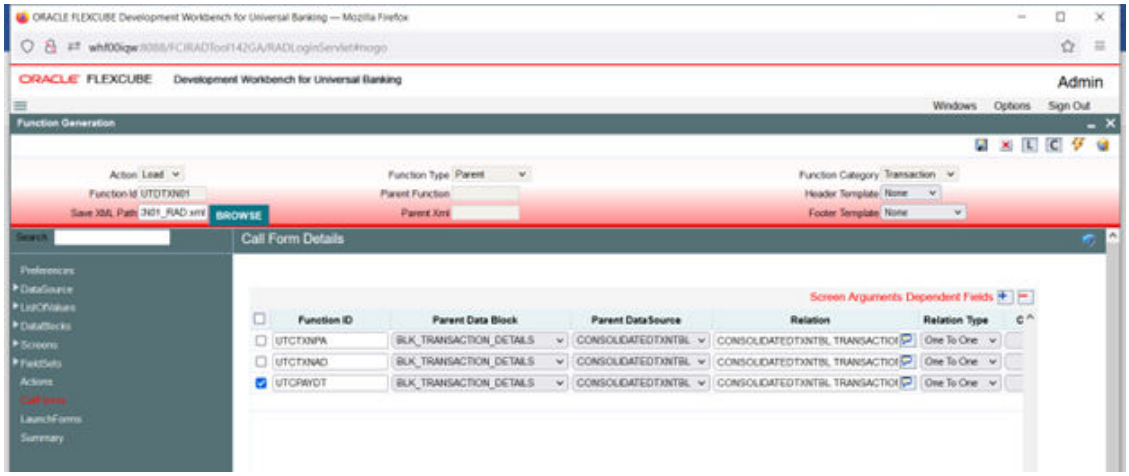


2.9 Call Forms

This topic describes about the call forms.

Call forms can be attached to Online form. Each call form should be mapped to Parent Data Block, Parent Data Source and proper relations should be maintained with parent data source of main online form.

Figure 2-17 Call Forms



Sreen Arguments should be given to each callform. So that the call form will display the respective data of calling main fuinction. Dependent Fields are required to re default the call form values when the user changes input data in the main form. Each of the subsytem pickup logic will have to be coded by the developer in release specific packages. Processing logic (sub system pickup) for the attached call forms has to be called from the main form package.

- Sub System Pickup/Processing
  This topic describes about the sub system pickup/processing.

2.9.1 Sub System Pickup/Processing

This topic describes about the sub system pickup/processing.

Subsystem pickup refers to the process of picking up the values in sub systems. Normally values in sub systems will be defaulted based on the data given in the main screen of the online form.

1. Defaulting of sub system: After providing values in the main screen, user may click on any sub system to view or change the value. On clicking the sub system for the first time, sub system values will be defaulted based on the values provided in the main screen. Action code passed will be SUBSYSPKP. The code for defaulting will have to written by the developer in corresponding hook packages in function.

   Fn_Post_Subsys_Pickup

   In this case SUBSYSSTAT for all subsystems will go as 'D' and processing done based on this flag for each sub system (call form). Note that SUBSYSPKP action will default values for all subsystems and not only the sub system being launched

   Example:MICTRMIS:D;ISCTRSTL:D;TACTRTAX:D;CSCTRUDF:D;CFCTROCH:D;CSCTRADV:D;FTCCGCLM:D;

   If user saves the contract without visiting any call forms, then all the subs systems will be defaulted before saving

2. Uploading of sub system: If after launching the subsystem with defaulted values; User changes the value in subsystem; the new user input values has to be uploaded to the system. Hence while saving, the subsystems which has been modified by user will be uploaded while others will be defaulted.In this case SUBSYSSTAT for the subsystem which

has been modified will go as ‘U’ .Developer has to write code for processing based on the flag.

- *Example: if user changes MIS details (MICTRMIS) from what was defaulted; then SUBSYSSTAT will go as*
- *MICTRMIS:U;ISCTRSTL:D;TACTRTAX:D;CSCTRUDF:D;CFCTROCH:D;CSCTRADV:D;FTCCGCLM:D*;

3. Re-defaulting of sub system: After launching and changing subsystem values; if user changes any values in main screen which are dependent field for the subsystem: subsystem values will have to be defaulted again based on the new main screen values. Hence the sub system will be re defaulted. In this case value entered by the user in subs system will be lost.In this case SUBSYSSTAT for the subsystem whose dependent fields has been modified will go as ‘R’. Developer has to write code for processing based on the flag.
   - *Example: In a Funds Transfer Contract Input Screen, assume that charge subsystem (CFCTROCH) is dependent on the values entered for debit and credit account. After launching the sub system and changing the charges manually; if user changes the account again the charges will have to re defaulted. The manually entered charges will not be considered. SUBSYSSTAT will go as*
   - *MICTRMIS:U;ISCTRSTL:D;TACTRTAX:D;CSCTRUDF:D;CFCTROCH:R;CSCTRADV:D;FTCCGCLM:D*;

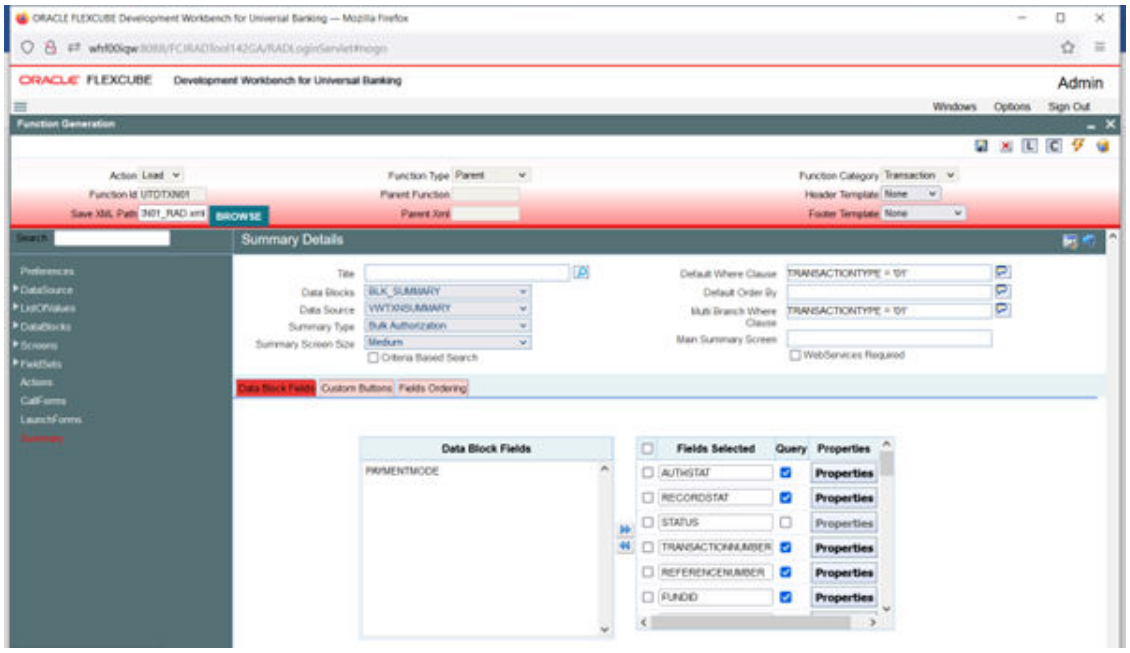
   Values for other subsystems will depend on each of their dependencies

2.10 Summary

This topic describes about the summary screen.

Summary screens can be designed for Online Form if required.

Figure 2-18 Summary



2.11 Preview

This topic describes about the preview.

The figure shows the preview of the Online form Input screen developed.

Figure 2-19 Input screen developed

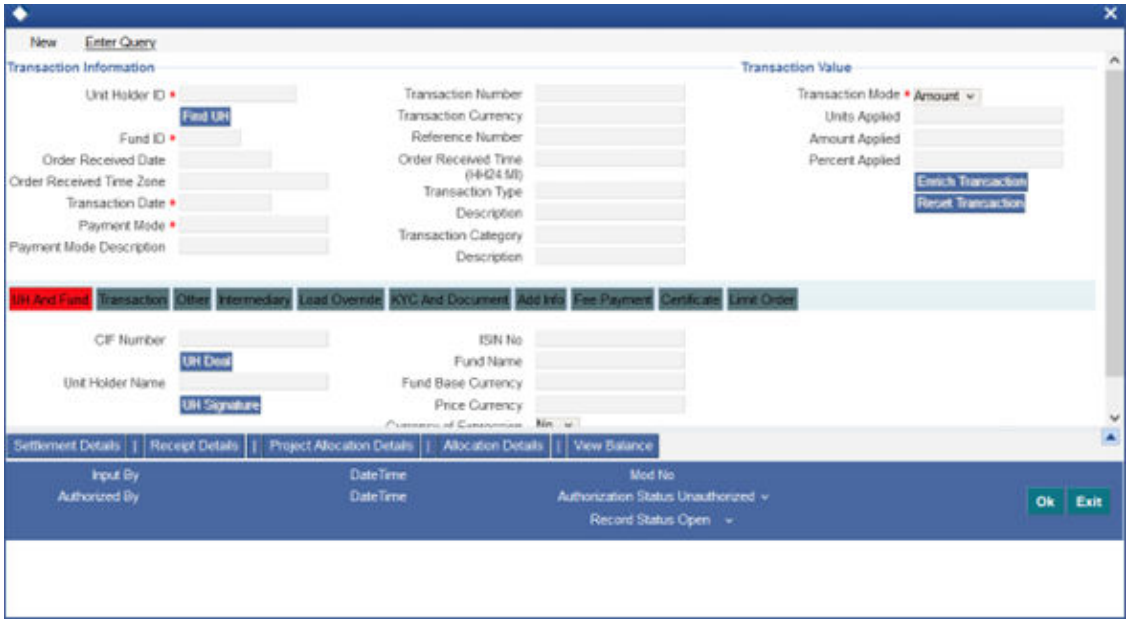
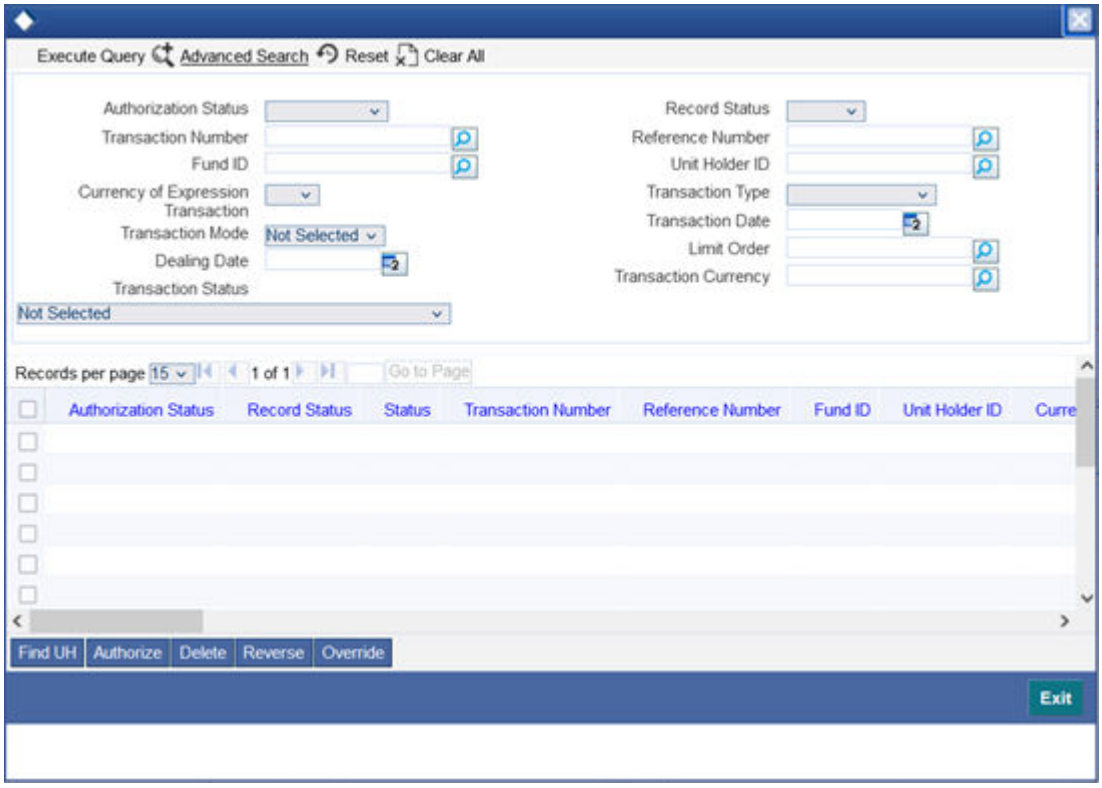


Figure 2-20 Summary screen developed



Generate the units for Online form and deploy them in the FLEXCUBE server for unit testing.

3

Attach Call Form to Main Function Id

This topic describes about the attach call form to main Function Id.

Call Forms cannot be launched independently. It has to be called from a main function id. Refer Call Forms section in Oracle FLEXCUBE Enterprise Limits and Collateral Management ODT Screen Development for detailed explanation

> **Note**
>
> Scripts for CSTB_CALL_FORM_NODES and SMTB_MENU tables generated by Call Form screen has to be deployed in FLEXCUBE schema before attaching Call form to the main function Id.

4
Generated Units

This topics describes about the generated units.

This topic contains the following sub-topics:

- Front End Units
  This topic consists of front end units.
- Data Base Units
  This topic describes about the data base units.
- Other Units
  This topic describes about the other units used in the module.

4.1 Front End Units

This topic consists of front end units.

This topic consists of sub-topics:

- Language xml
  This topic describes about the language xml.
- SYS JavaScript File
  This topic describes about the SYS javaScript file.
- Release Type Specific JavaScript File
  This topic describes about the release type specific javascript file.

4.1.1 Language xml

This topic describes about the language xml.

This file is an XML markup of presentation details, for the designed Call Form specific to a language.

Example – **UTDTXN01.xml** (UIXML for UT Screen).

4.1.2 SYS JavaScript File

This topic describes about the SYS javaScript file.

This JavaScript file mainly contains a list of declared variables required for the functioning of the screen.

Example – **UTDTXN01_SYS.js** (JS for UT Screen).

4.1.3 Release Type Specific JavaScript File

This topic describes about the release type specific javascript file.

This file won't be generated by the Tool. It has to be manually written by the developer if he has to write any code specific in that release.

Example – UTDTXN01_KERNEL.js (JS for KERNEL Release).

Example – UTDTXN01_CLUSTER.js (JS for CLUSTER Release).

Example – UTDTXN01_CUSTOM.js (JS for CUSTOM Release).

4.2 Data Base Units

This topic describes about the data base units.

This topic consists of sub-topics:

- Static Scripts
  This topic describes about the static scripts.
- System Packages
  This topic describes about the system packages.
- Hook Packages
  This topic describes about the hook packages.

4.2.1 Static Scripts

This topic describes about the static scripts.

The following static scripts generated are required for the proper functioning of an Online Form screen. Refer document on generated units for detailed explanation.

4.2.2 System Packages

This topic describes about the system packages.

Main package would be generated by the Tool and should not be modified by the developer.

There is small change in the structure of the package depending on the type of the call form (Maintenance or Transaction).

Unlike normal maintenance function ids, call form packages does not have any call to the business logic within itself (similar to transaction function id). If developer wishes to uses any functions within the main package , call has to be made from the release specific package.

Main package contains functions for :

- Converting Ts to PL/SQL Composite Type
- Calling fn_main.
- Mandatory checks (fn_check_mandatory).
- Default and validation(fn_default_and_validate)
- Querying(fn_query)
- Converting the Modified Composite Type again to TS

Except the functions for type conversions, others functions calls the respective hook functions in hook packages of the call forms. Thus no processing logic within the main package is used It is to be noted that each of these functions are called from the main package of the main function id (where this call form is used) during respective stages.

But the package contains many other system generated functions for operations like

- Mandatory checks(fn_sys_check_mandatory)
- Default and validation(fn_sys_default_and_validate)
- Uploading to DB(fn_sys_upload_db)
- Query operation (fn_sys_query) etc

These functions are not called anywhere in the package. These functions if required can be called by the developer from the release specific package. Otherwise developer can write his own logic for the same in the Hook Packages

4.2.3 Hook Packages

This topic describes about the hook packages.

Release specific packages will be generated based on the release type (KERNEL.CLUSTER or CUSTOM). The structure of the package depends on the type of call form (Maintenance or Transaction). Developer can add his code in the release specific hook package.

Example –utpks_utdtxn01_kernel.spc, utpks_utdtxn01_kernel.sql (Kernel Package)utpks_utdtxn01_cluster.spc, utpks_utdtxn01_cluster.sql (Cluster Package) utpks_utdtxn01_custom.spc, utpks_utdtxn01_custom.sql (Custom Package).

4.3 Other Units

This topic describes about the other units used in the module.

This topic consists of sub-topics:

- Xsd
  This topic describes about the Xsd.

4.3.1 Xsd

This topic describes about the Xsd.

Only Type XSD will be generated for a Call Form function Id. Subscript **Subys** will be added before XSD Type identifier in the name of the generated xsd.

This type xsd will be used in the type xsd of any function which uses the particular call form.

Example – UT-FCISTransaction-Types.xsd (Type XSD for UT)UT-CreateTransaction -Req-Full-MSG.xsd (Create Message XSD for UT)UT-CreateTransaction -Req-PK-MSG.xsd (Create Message XSD for UT)UT-CreateTransaction -Res-Full-MSG.xsd (Create Message XSD for UT) UT-CreateTransaction -Res-PK-MSG.xsd (Create Message XSD for UT).

5

Extensible Development

This topic describes about the extensible development.

Developer can add his code in hook packages and release specific JavaScript file.

This topic contains the following sub-topics:

- Extensibility in JavaScript Coding
  This topic describes about the extensibility in javascript coding.
- Extensibility in Backend Coding
  This topic describes about the extensibility in backend coding.

5.1 Extensibility in JavaScript Coding

This topic describes about the extensibility in javascript coding.

For release specific JavaScript coding, code has to be written in release specific JavaScript file. It follows the naming convention as : (Function Id)_(Release Type).js

Example: Code in **UTDTXN01_CLUSTER.js** is exclusive to cluster release

This JavaScript file allows developer to add functional code and is specific to 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
  Example: **fnPreLoad_CLUSTER(), fnPostLoad_CLUSTER()**
- Functions triggered by screen action events
  Example: **fnPreNew_ CLUSTER (), fnPostNew_ CLUSTER ()**

5.2 Extensibility in Backend Coding

This topic describes about the extensibility in backend coding.

For online forms, generated code does not provide any business logic. Insert statements won't be present as part of generated code in online packages. Developer has to write the business logic in release specific packages (or make call l to server functions from release specific packages).

Hooks will be provided in the following stages Resolving reference numbers:

- Checking mandatory fields
- Defaulting and validating
- Uploading to db
- Process
- Subsystem pickup
- Enrich

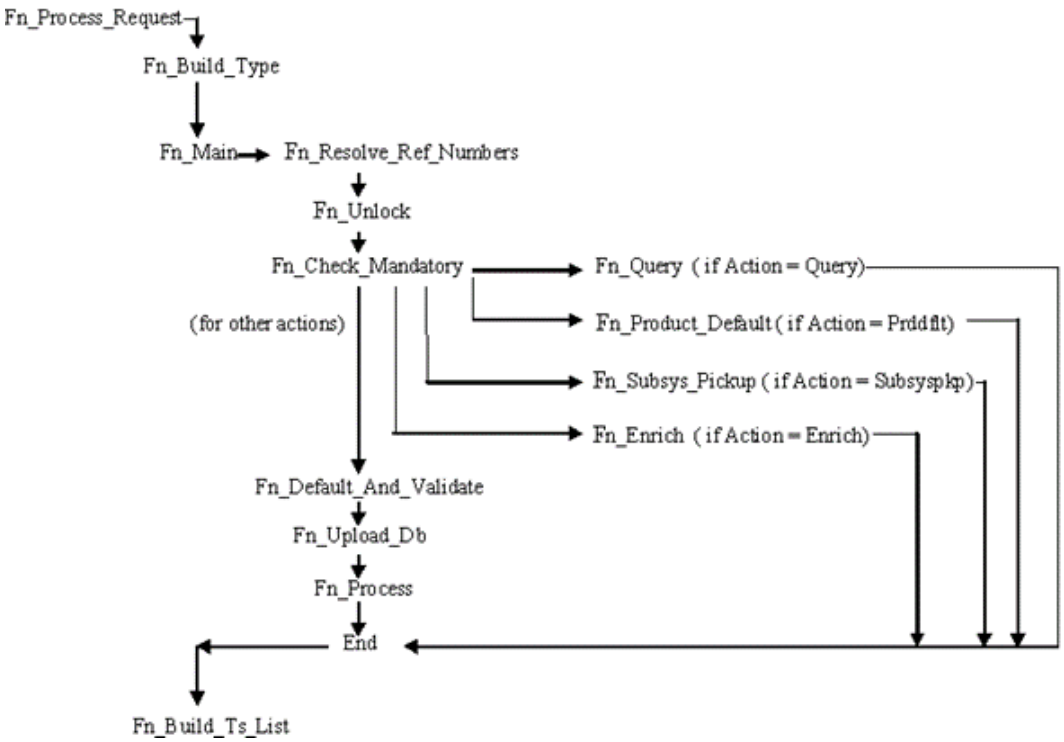
- Product Default
- Query

> **Note**
>
> The system generated code for uploading; defaulting etc. (*fn_sys_default_and_validate, fn_sys_upload_db etc.*) won't be called by the main package in online flow. If it is required, developer has to call it explicitly from release specific packages.

In online flow, upload to base tables happens first and processing is done on the inserted data after uploading. After processing, the response type will be build.

Note that in online flow, upload to base tables happens first and processing is done on the inserted data after uploading. After processing, the response type will be build.

Figure 5-1 Flow of control in an Online main package



Release specific code has to be written in the Hook Packages generated. The different functions available in the Hook Package of an Online Form are:

1. **Skip Handler** : **Pr_Skip_Handler**
   This can be used to skip the logic written in another release.

   Example: logic written in **KERNEL** release can be skipped in **CLUSTER** release

2. **Fn Main**
   This is called form the fn_main in main package.

3. **Fn_pre_resolve_ref_numbers**
4. **Fn_post_resolve_ref_numbers**
   This function validates the reference number. It is called from fn_ resolve_ref_numbers of the main package.
5. **Fn_pre_unlock**
6. **Fn_post_unlock**
   This function holds the contract level validations and modification logic for existing contract. It is called from fn_unlock of main package.
7. **Fn_pre_check_mandatory**
8. **Fn_post_check_mandatory**
   Any mandatory checks can be validated here. It is called from fn_chchk_mandatory of main package.
9. **Fn_pre_query**
10. **Fn_post_query**
    Any specific logic while querying can be written in these functions. It is called from fn_query of the main package.
11. **Fn_pre_product_default**
12. **Fn_post_product_default**
    This function has the logic to default the values for the contract based on the product maintenance. It is called from fn_product_default of main package.
13. **Fn_pre_subsys_pickup**
14. **Fn_post_subsys_pickup**
    This function does the subsystem pickup for the subsystem's (call form's) as per product maintenance for the contract. It is called from fn_subsys_pickup of main package.
15. **Fn_pre_enrich**
16. **Fn_post_enrich**
    After product default, user can default others values. That logic can be put here. it is called from fn_enrich of main package.
17. **Fn_pre_default_and_validate**
18. **Fn_post_default_and_validate**
    Any release specific logic for defaulting and validation can be written here. It is called from the fn_default_and_validate in the main package.
19. **Fn_pre_upload_db**
20. **Fn_post_upload_db**
    Any logic while uploading data to tables can be written here**.** It is called from fn_upload_db of main package.
21. **Fn_pre_process**
22. **Fn_post_process**
    These hook functions are specific to transaction online form screens. This function should have the call to all the server functions which process the input data for the contract as per the functionality. These are called from **fn_process** of the main package.