Oracle® Banking Enterprise Limits and Collateral Management

Development of Maintenance Form

Release 14.8.0.0.0
G32532-01
April 2025

Oracle Banking Enterprise Limits and Collateral Management Development of Maintenance Form, Release 14.8.0.0.0

G32532-01

Contents

1
Preface

1.1 Purpose

This guide is designed to help acquaint you with the Oracle Banking Enterprise Limits and Collateral Management (ELCM) application. This guide provides answers to specific features and procedures that the user need to be aware of the module to function successfully.

User can further obtain information specific to a particular field by placing the cursor on the relevant field and pressing <F1> on the keyboard.

1.2 Audience

This guide is intended for the following User/User Roles:

Table 1-1 Audience

Role	Function
Back office data entry clerk	Input functions for funds
Back office managers/officers	Authorization functions
Product Managers	Product definition and authorization
End of day operators	Processing during end of day / beginning of day

1.3 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 customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

1.4 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.

1.5 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.

1.6 Basic Actions

Table 1-2 Basic Actions

Action	Description
Approve	Used to approve the initiated report. This button is displayed, once the user click **Authorize**.
Audit	Used to view the maker details, checker details, and report status.
Authorize	Used to authorize the report created. A maker of the screen is not allowed to authorize the report. Only a checker can authorize a report, created by a maker.
Close	Used to close a record. This action is available only when a record is created.
Confirm	Used to confirm the performed action.
Cancel	Used to cancel the performed action.
Compare	Used to view the comparison through the field values of old record and the current record. This button is displayed in the widget, once the user click **Authorize**.
Collapse All	Used to hide the details in the sections. This button is displayed, once the user click **Compare**.
Expand All	Used to expand and view all the details in the sections. This button is displayed, once the user click **Compare**.
New	Used to add a new record. When the user click **New**, the system displays a new record enabling to specify the required data.

Table 1-2 (Cont.) Basic Actions

Action	Description
OK	Used to confirm the details in the screen.
Save	Used to save the details entered or selected in the screen.
View	Used to view the report details in a particular modification stage. This button is displayed in the widget, once the user click **Authorize**.
View Difference only	Used to view a comparison through the field element values of old record and the current record, which has undergone changes. This button is displayed, once the user click **Compare**.
Unlock	Used to update the details of an existing record. System displays an existing record in editable mode.

1.7 Related Documents

For more information refer to the Oracle Banking manuals on:

- Development of Launch Forms and Others Screens
- Enterprise Collaterals User Guide
- Enterprise Limits and Collaterals Common User Guide

1.8 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.

1.9 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.

1.10 Acronyms and Abbreviations

The list of the acronyms and abbreviations used in this guide are as follows:

Table 1-3 Acronyms and Abbreviations

Acronyms	Abbreviations
CIF	Customer Information File

Table 1-3 (Cont.) Acronyms and Abbreviations

Acronyms	Abbreviations
CASA	Current Account and Savings Account
DDA	System that holds the CASA account and balances
ELCM	Enterprise Limits and Collateral Management
ECA	External Credit Approval
FCUBS	Oracle FLEXCUBE Universal Banking Solution
GW	Gateway
HTTP	Hyper Text Transfer Protocol
ID	Identification Number
Mark EOTI	Mark End of Transaction Input
Mark TI	Mark Transaction Input
OFSAA	Oracle Financial Services Analytical Applications
ORMD	Oracle Revenue and Billing Management
PK	Primary Key
RDBMS	Relational Data Base Management System
SMS	Security Services
UI	User Interface
VD	Value Date
XML	Extensible Mark-up Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

1.11 Symbols and Icons

The lists of symbols, buttons and shortcut key that are used in the application to perform various tasks are covered in this topic.

Table 1-4 Symbols and Icons

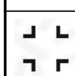
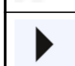
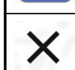
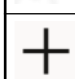
Icons	Function
	Perform search
	Minimize
	Navigate to the next record
	Navigate to the previous record
	Toggle OFF
	Toggle ON
	Delete
	Click this icon to add a new row.

Table 1-4 (Cont.) Symbols and Icons

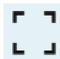
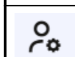
Icons	Function
	Click this icon to delete an existing row.
	List view
	Maximize
	Navigate to the first record
	Navigate to the last record
	Advance search
	Search record
	Save the record
	Reset the record
	Clear the record

Table 1-5 Symbols and Icons - Audit Details

Icons	Function
	A user
	Branch details
	Date and Time

1.12 Prerequisite

Specify the **User ID** and **Password**, and login to **Home** screen.

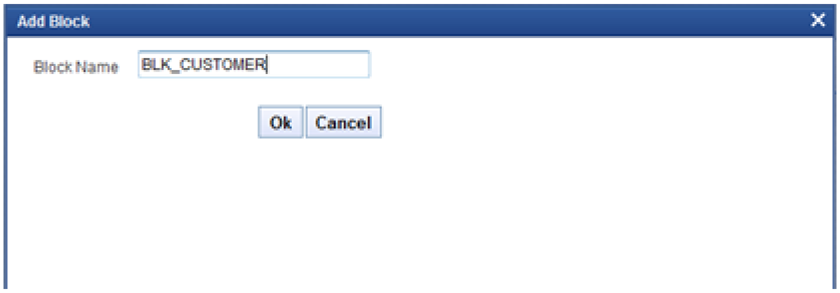
2 Create Data Block

This topic provides systematic instructions to create Data Block.

1. Under **Function Generation** screen, right-click on the **Data Block** and select **Add** option to create the new data block.

   The **Add Block** window displays.

   Figure 2-1 Add Block

2. Specify the **Block Name** field and click **OK**.

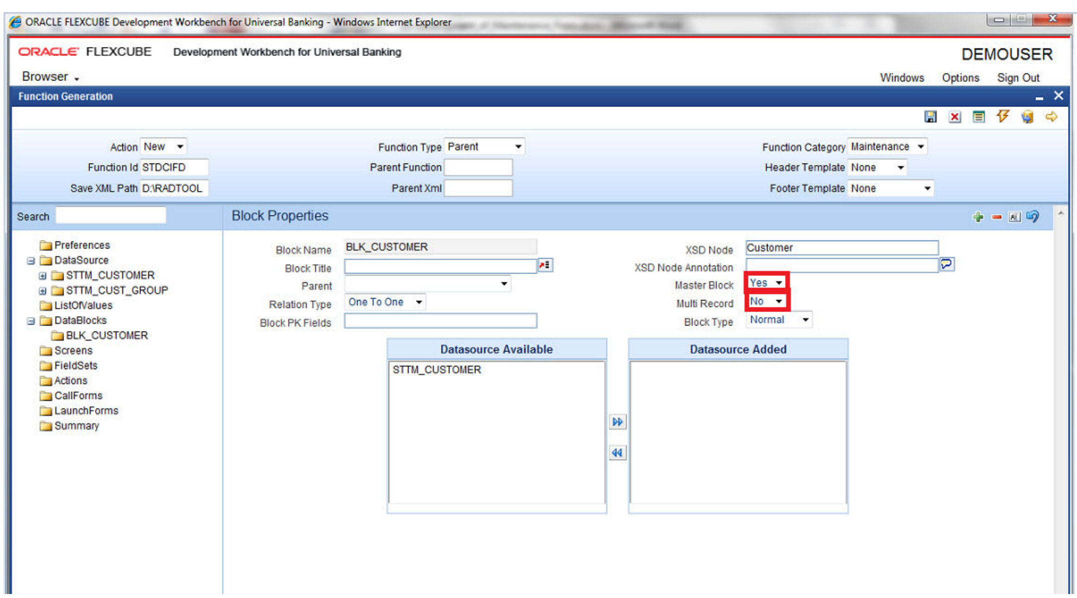
   Block Name should start with **BLK_**. The **Block Name** should be short and equivalent to a data source but not the same as **DataSource Name**.

   The new DataBlock gets created and **Block Properties** screen displays.

3. Select **Parent** block if added block is not **Master Block**.
4. Select **Multi Record** field as **Yes/No** based on **Master Block** value.

   Available data sources displays in the **DataSource Available** text area.

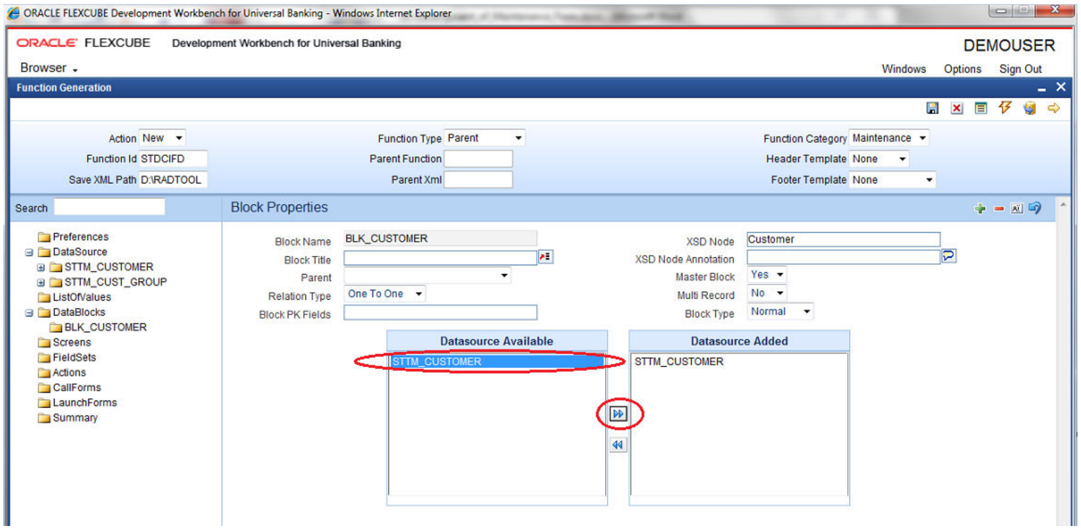
   Figure 2-2 Block Properties

5. Select the required data source and click **Move** to attach Data Source to the Block.

   The Selected Data sources gets attached to the Block.

Figure 2-3 Block Properties- Attaching Data Sources to Data Block



6. To add multi record data source to data block, select **Multi Record** as **Yes** in the data **Block Properties** screen.

   Multi Data Source once used to one block won't be available for reuse whereas a single record data source can be used in multiple blocks.

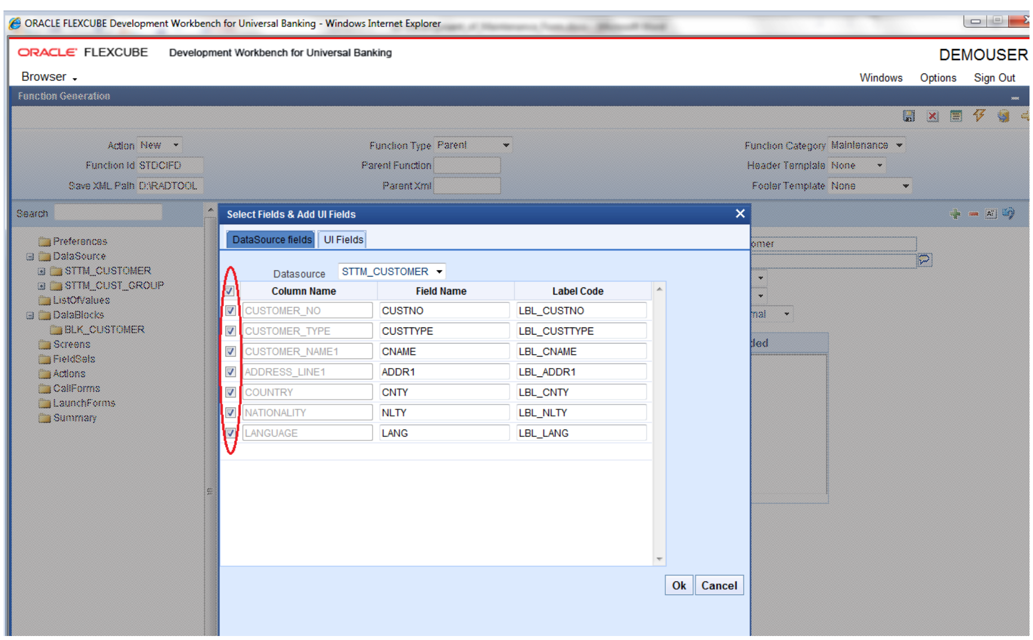
   All the data sources with **Multi Record** as **Yes** will be populated.

7. Select Block Fields as follows:

   a. Right-click on the newly added block.

      Select Fields & Add UI Fields window displays.

Figure 2-4 Select Fields & Add UI Fields



b. Check the right side check boxes to add the required fields.

c. Verify the **Field Name** and **Label Code** details and click **OK**.

   The **Field Name** should not be the same as the column name. Special characters are also not allowed in the **Field Name** (including underscore and space). **Label Code** will be automatically populated based on the **Field Name**. Management ODT Screen Development

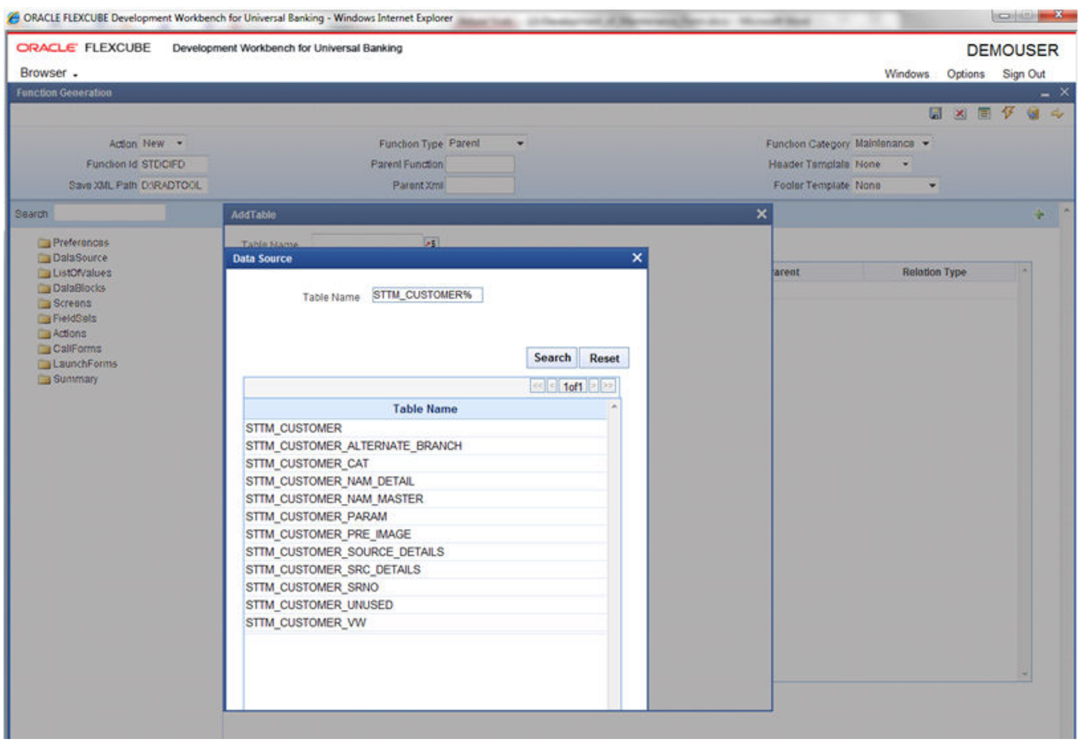
3
Create Data Sources

This topic provides systematic instructions to create Data Sources.

1. Under **Function Generation** screen, right-click on the **DataSource** node and select the **Add** option to create a new Data Source.
2. Go to the List Of values to get the list of tables available and select the required table from the list.

   If the user knows the exact **Table Name**, the user can specify the name directly.

   Figure 3-1 Data Source

3. Select **Master** as **Yes** if the added data source is Master Data Source for the screen.

   Every function ID should have one master data source.
4. Verify **Pk Cols** and **Pk Types** fields are populated and if its not entered then specify the **Pk Cols** and **Pk Types** fields.

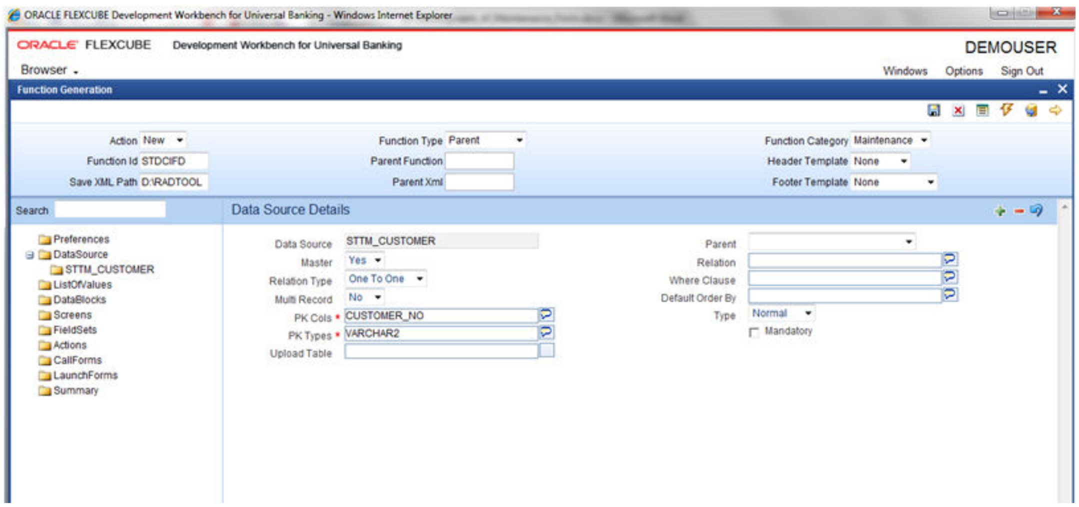
   Primary Key columns (i.e. **Pk Cols**) and Primary Types (i.e. **Pk Types**) are mandatory. If it is already maintained in user schema in **STTB_PK_COLS** it will populate automatically otherwise the user needs to enter values without fail. If the user misses **Pk Cols** and **Pk Types** package generation will fail.

> **Note:**
>
> Master Data Source cannot have any parent.

5. Click on the **Save** icon at the top right of the screen.

   Data Source Details screen displays with all the details.

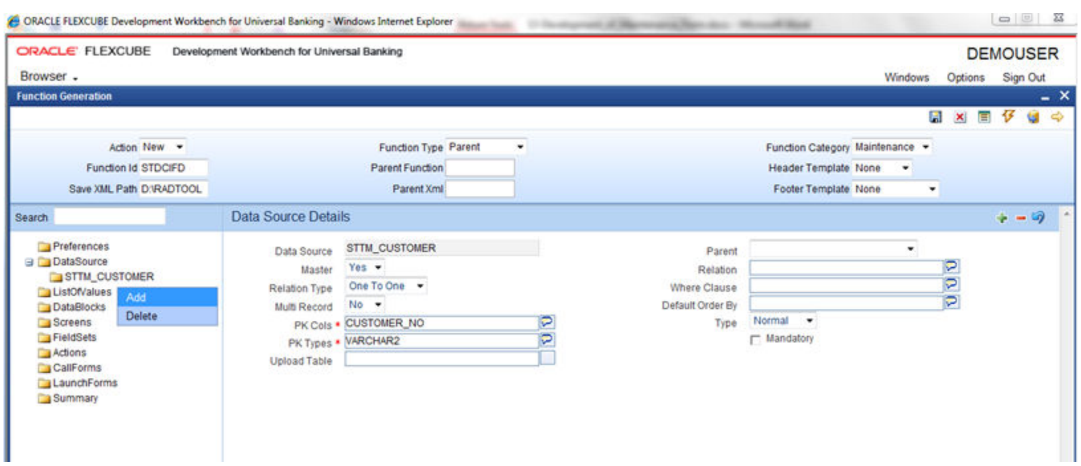
Figure 3-2 Data Source Details



6. Under the **DataSource** node, right-click on the added table (**STTM_CUSTOMER**).

   Data Source Details screen displays with added table.

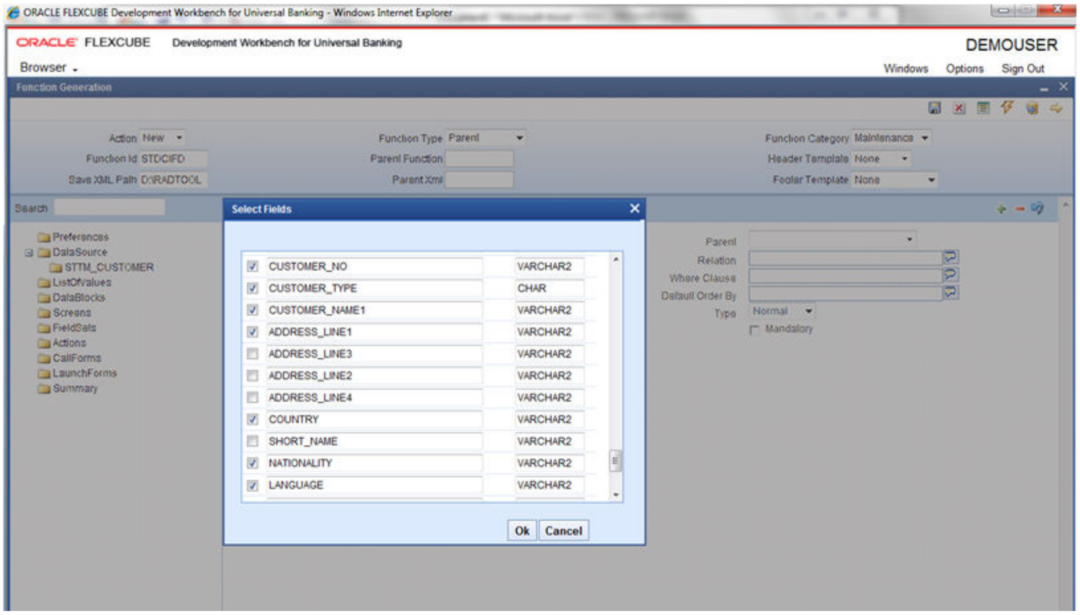
Figure 3-3 Data Source Details- Added Table



7. Click on the **Add** option to add fields to the table.

   Select Fields window displays.

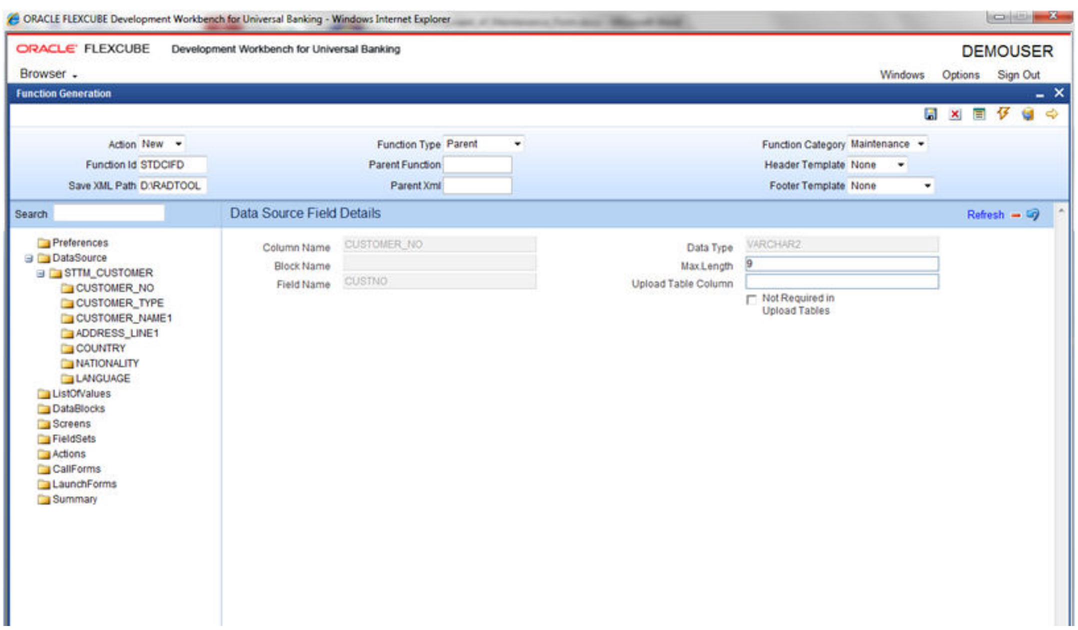
Figure 3-4 Select Fields



8. Select the required fields and click **OK**.

   Selected fields will get added to the Data Source Tree.
   Data Source Field Properties: Only maximum length can be modified by the developer in **Data Source Field Properties**. Rest will have defaulted from the table definition.

Figure 3-5 Data Source Field Details



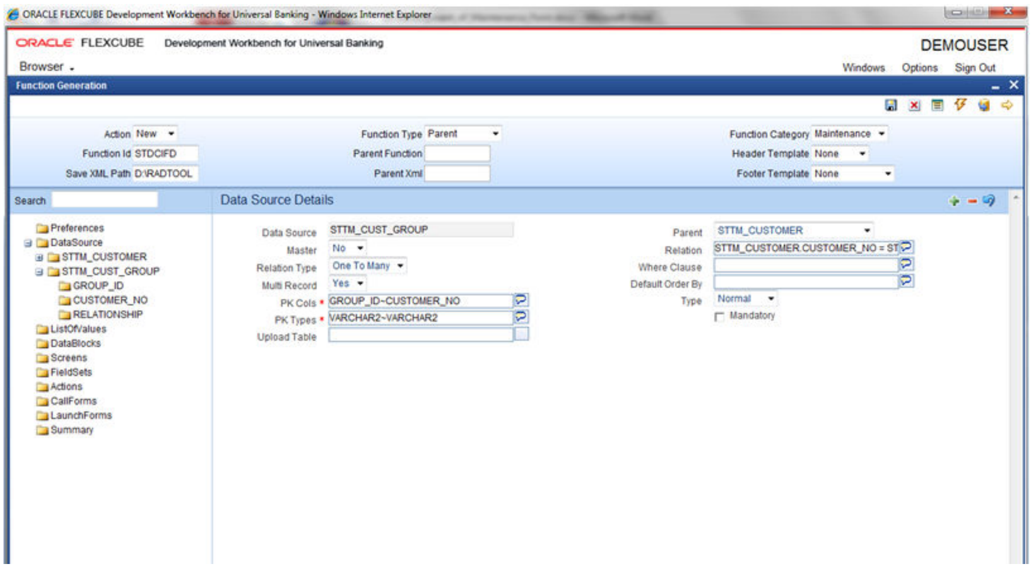
The data model of a single function ID would include multiple tables. All the tables need to add to the function ID. Note the following while adding child data sources.

9. To add Child Data Source, follow the below steps:

- In **Data Source Details** screen, select **Multi Record** value as **Yes** if the child data source is Multi record table.

  Child Data sources should always be associated with a parent. Relation is mandatory between parent and child. While giving relation, the parent data source should come on the left side of the relation.

Figure 3-6 Data Source Details- Child Data Source



> **Note:**
>
> A data source cannot be the parent to itself.

Note the following while adding data sources:

- If the data source is designed with relation type as 1: N with its parent, then it should have at least one more **Pk Col** than its parent (assuming the relationship is based on **Pk Cols**).
- Master data source needs to have the audit columns if footer template is Maint audit, but those should not be added to data source fields as the system will handle it.

Refer Management ODT Screen Development for detailed explanation on data sources.

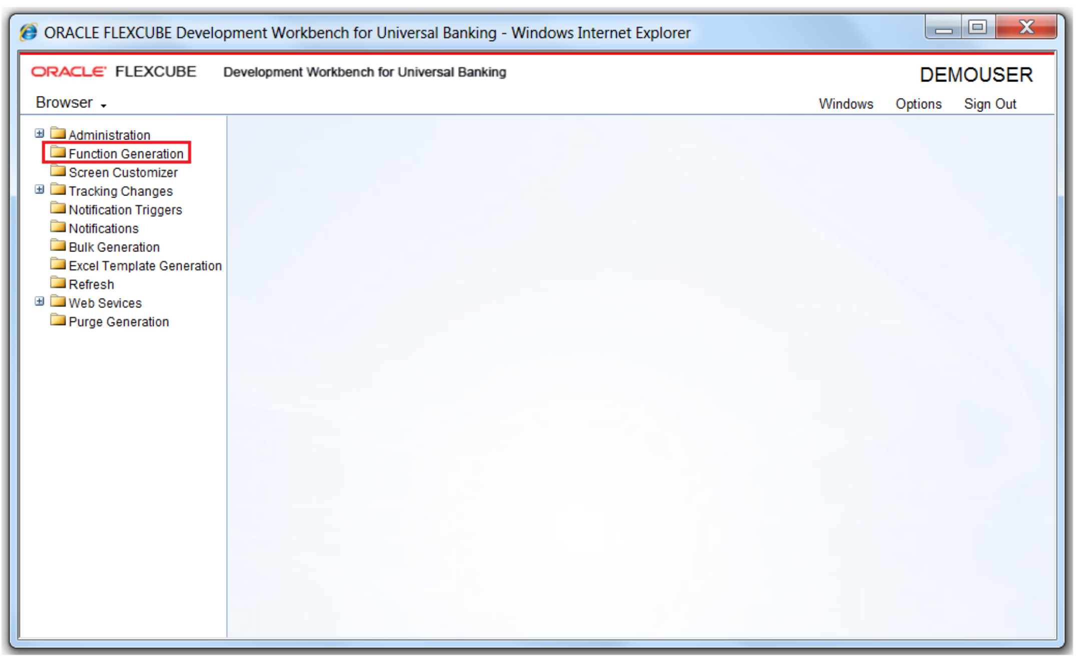
4
Create New Maintenance Function ID

This topic explains systematic instructions to create a new maintenance function ID.

1. Specify the **Username** and **Password** in the Development Workbench Universal Banking Login page and log in to the Home screen.

   Development Workbench Universal Banking Homepage displays.

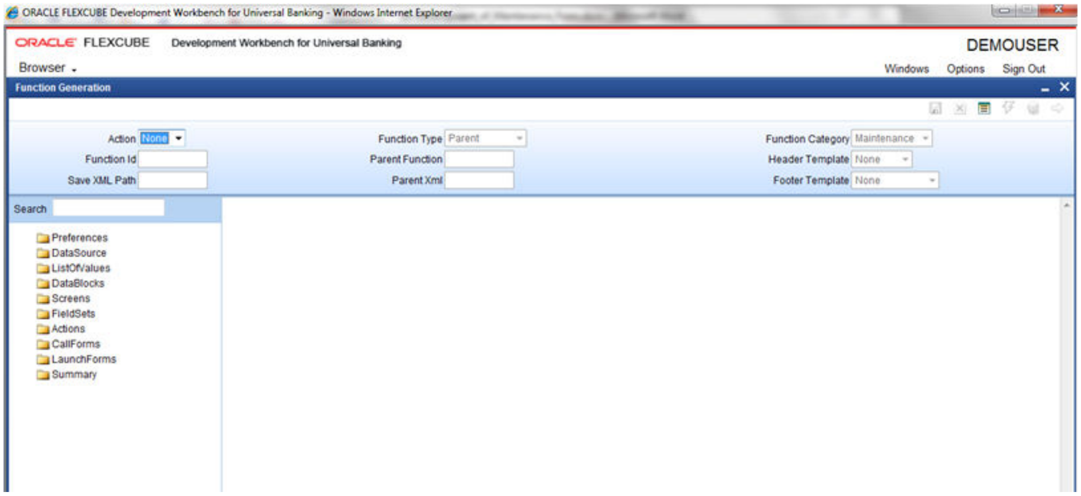
   Figure 4-1 Development Workbench Universal Banking

2. Click on the **Function Generation** node in the **Browser** menu.

   Function Generation screen displays.

Figure 4-2 Function Generation



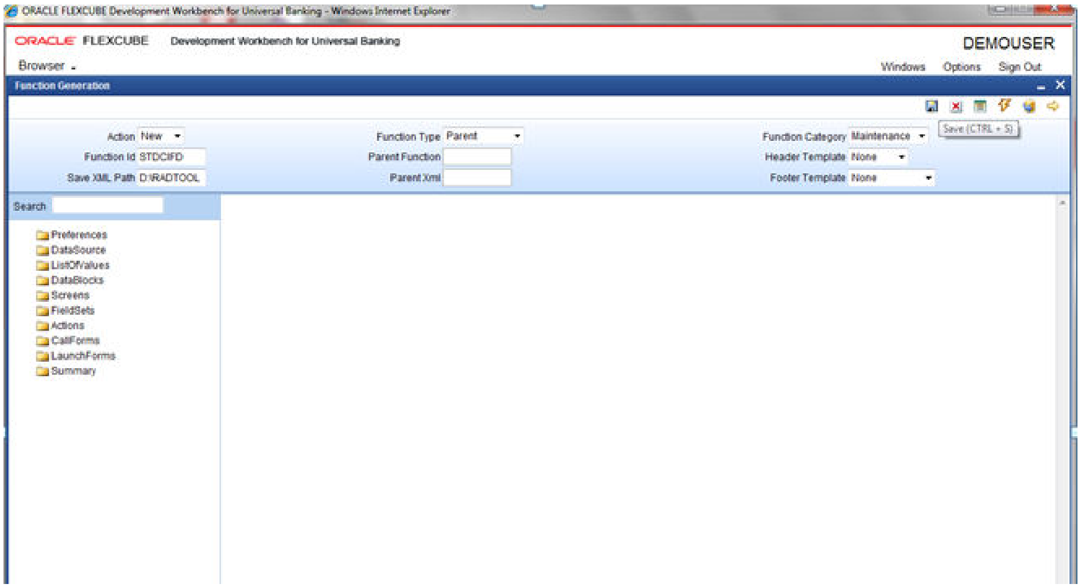
3. Specify the details in the Header section of the **Function Generation** screen.

Table 4-1 Function Generation- Field Description

Field	Description
Action	Select the **Action** field as **New**.
Function Type	Select **Function Type** field as **Parent**.
Parent XML	None
Parent Function	None
Function Category	Select the **Function Category** field as **Maintenance**.
Function ID	Enter the **Function ID** field as **STDCIFD**.
Header Template	None (Only for Process flow screens)
Footer Template	Maintenance Audit

4. Click on the **Save** icon on the top right of the screen.

Figure 4-3 Function Generation Screen- Save RADXML

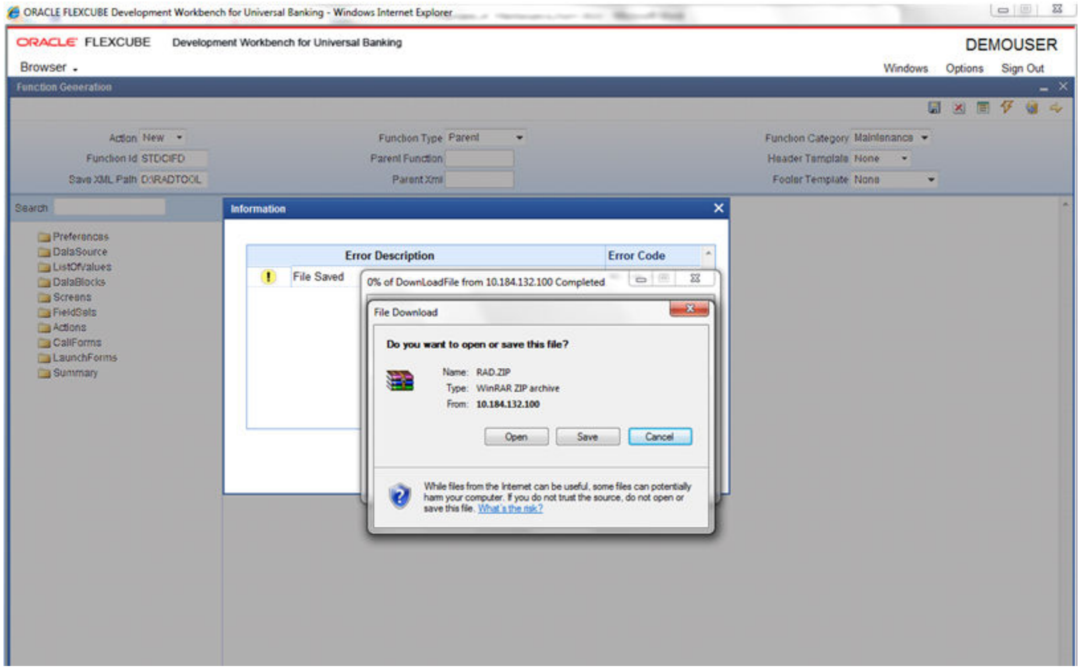


The user can save work at any point in time.

> **Note:**
>
> To work again with the saved RADXML file, select **Action** field as **Load** and load RADXML from the hard disk path.

Information window displays to confirm successful generation of Maintenance function ID and the save message displayed in **Information** window, if required.

Figure 4-4 Information



Note the following while providing header information for the Maintenance screen:

- **Naming Convention**: The third letter of the function ID has to be D. Ideally the function ID name should have 8 characters.
- **Footer Template**: Make sure that the master data source has the audit columns if the footer template is provided as Maintenance log.

Refer Management ODT Screen Development for the detailed explanation.

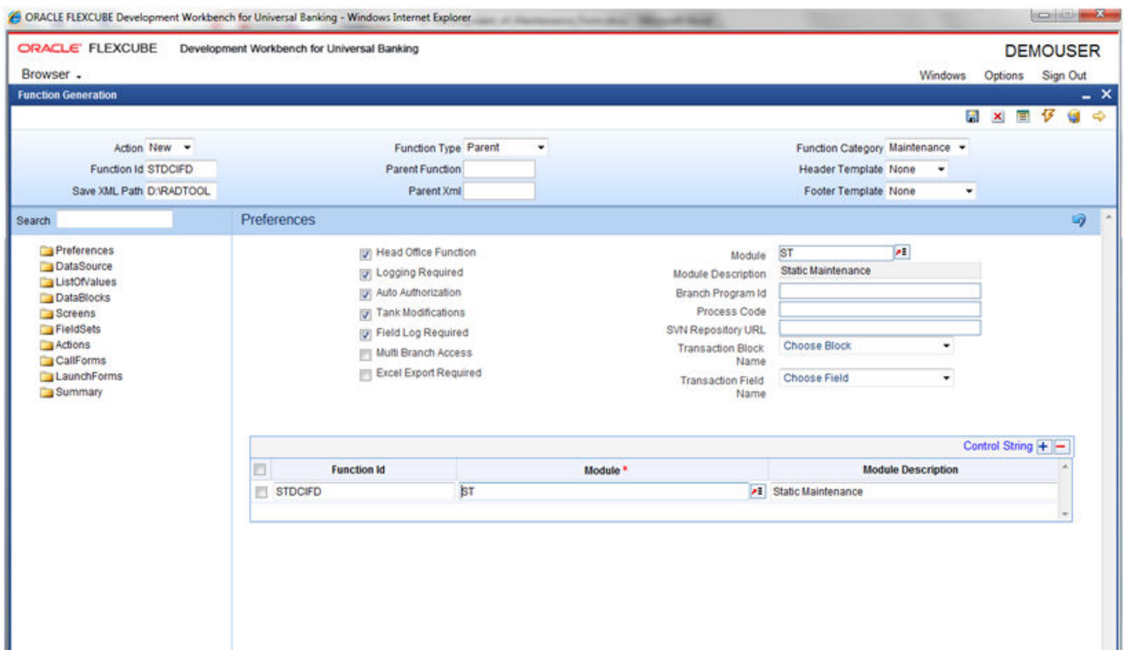
5
Preferences

This topic provides an overview of the Preferences screen.

Details entered in **Preferences** screen are used in generating INCS for **SMTB_MENU**, **SMTB_FUNCTION_DESCRIPTION** and **SMTB_ROLE_DETAILS**.

In the **Control String** field, the developer needs to select the actions which should be available for **Preferences** screen in the Oracle FLEXCUBE Universal Banking.

Figure 5-1 Preferences- Maintenance Screen



Note the following points while providing details in the Preferences screen

1. **Control String**: **REVERSE**, **ROLLOVER**, **CONFIRM**, **LIQUIDATE**, **HOLD** operations are not applicable for maintenance screens.
2. **Defining Browser Menu Tree**: **Browser** menu tree will be defined in the script generated for **smtb_function_description**.

The following labels has to be maintained for generation of proper script:

- Main Menu: **LBL_{function ID}_MAIN_MENU**
- Sub Menu 1: **LBL_{function ID}_SUB_MENU_1**
- Sub Menu 2: **LBL_{function ID}_SUB_MENU_2**
- Description: **LBL_{function ID}_DESC**

Refer Management ODT Screen Development for the detailed explanation on preferences.

Example 5-1 Labels to be maintained for STDCIFD

- **LBL_STDCIFD_MAIN_MENU**
- **LBL_STDCIFD_SUB_MENU_1**
- **LBL_STDCIFD_SUB_MENU_2**
- **LBL_STDCIFD_DESC**

6 Screen Development

This topic provides an overview of Maintenance screen development.

The design and development of a Maintenance function ID are similar to any other function IDs. This section briefs the steps in designing a Maintenance screen. STDCINF is the sample function ID used for demonstration in this document. For a detailed explanation, refer to the document: Management ODT Screen Development

7

Overview of Maintenance Screen

This topic provides an overview of the Maintenance Screen.

Maintenance function IDs are used for storing maintenance data which are required for processing any contracts, batches, or for any other maintenance which are dependent on this.

Business logic for a maintenance function ID would be provided by the Development Workbench generated files. In most cases, system-provided logic would be sufficient. Extra validations can be coded in the hook packages by the developer.

Example 7-1   Customer maintenance screen

If any customer wants to use the service of a bank, details about the customer will have to be maintained in the system. This will be maintenance data which will be required for other maintenances (creating an account for the customer) as well as for transaction processing (debiting of customer account).

8 Add Summary

This topic provides systematic instructions to add Summary.

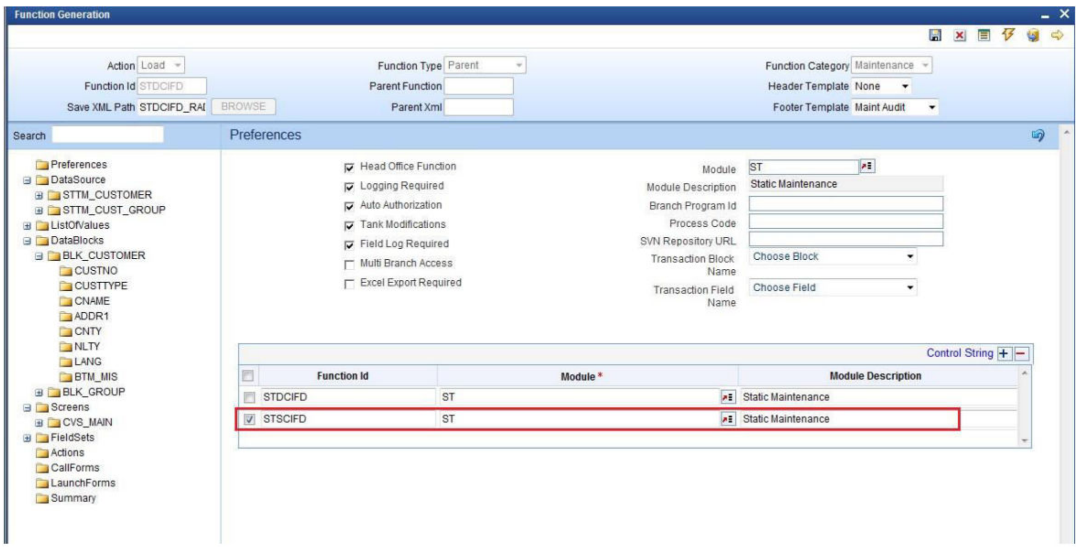
1. In **Function Generation** screen, click on the **Preferences** node.

   The **Preferences** screen displays.

2. In **Preferences** screen, add entry for the summary screen.

   Preferences screen displays.

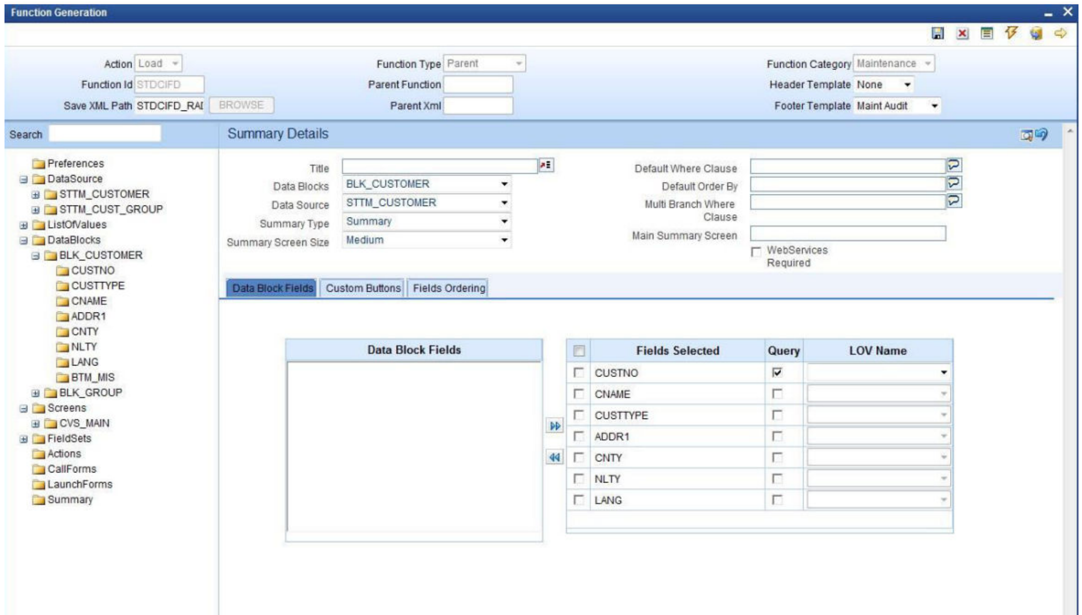
Figure 8-1 Preferences



3. Click on the **Summary** Node.

   The **Summary Details** screen displays.

Figure 8-2 Summary Details



4. Specify the details in the **Summary Details** screen.

Table 8-1 Summary Details

Field	Description
Title	Enter the Summary title. Select label code from LOV.
Data Blocks	Select Data Block master block and summary blocks will be displayed. Select a required block from the drop-down list.
Data Source	Select Data Source for summary.
Summary Type	Select Summary Type.
Summary Screen Size	Select Summary Screen size.
Default Where Clause	Enter if any where clause is required.
Default Order By	Enter Default order by if required.
Multi Branch Where Clause	Enter Multi Branch where clause if required.

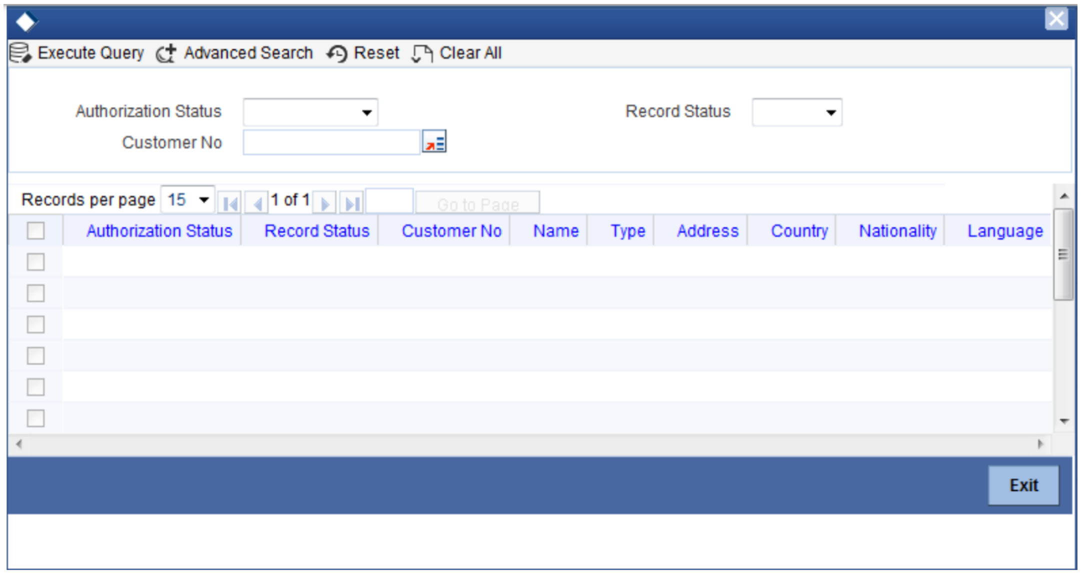
5. Attach the fields required in the summary result grid.

   If the field is required as part of filtering, the query has to be checked for the particular field.

6. Provide the position of fields in the **Result** grid and **Summary Query** set.
7. For summary preview, right-click on the **Summary** node and click on the **Preview**.

   The **Summary Preview** screen displays.

Figure 8-3 Summary Preview



9

Attach Call forms

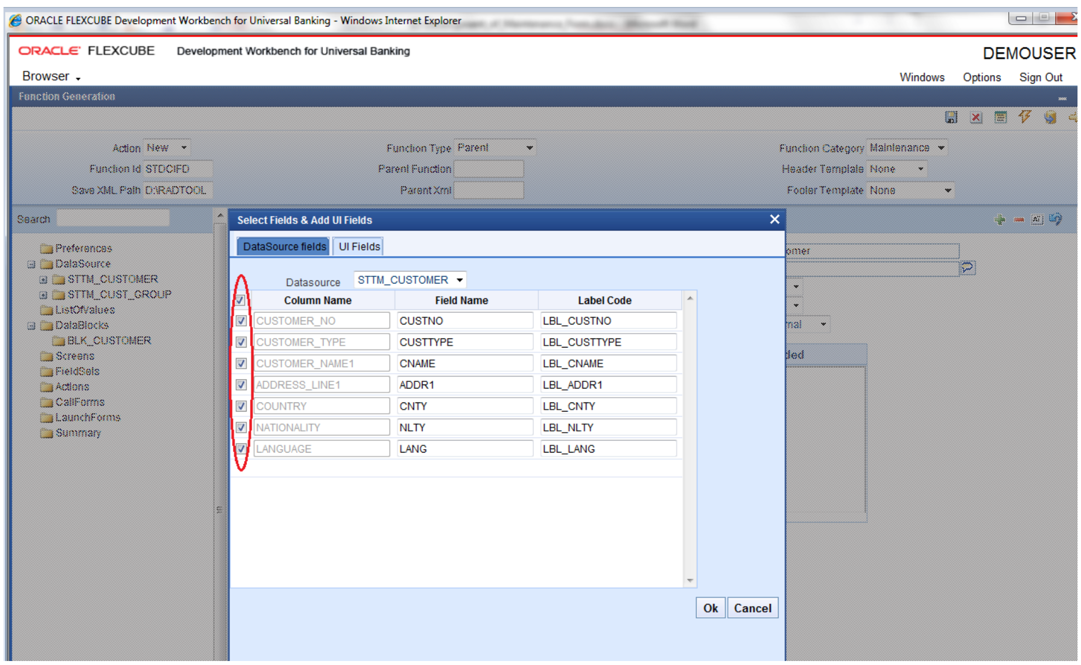
This topic explains systematic instructions to attach call forms.

Maintenance Call forms can be attached to a maintenance screen. Refer the document Call Form for developing call forms. For attaching Call forms follow the below given instructions:

1. Right-click on the **Block** and select **Add Fields** option.

   The **Select Fields and Add UI fields** window displays.

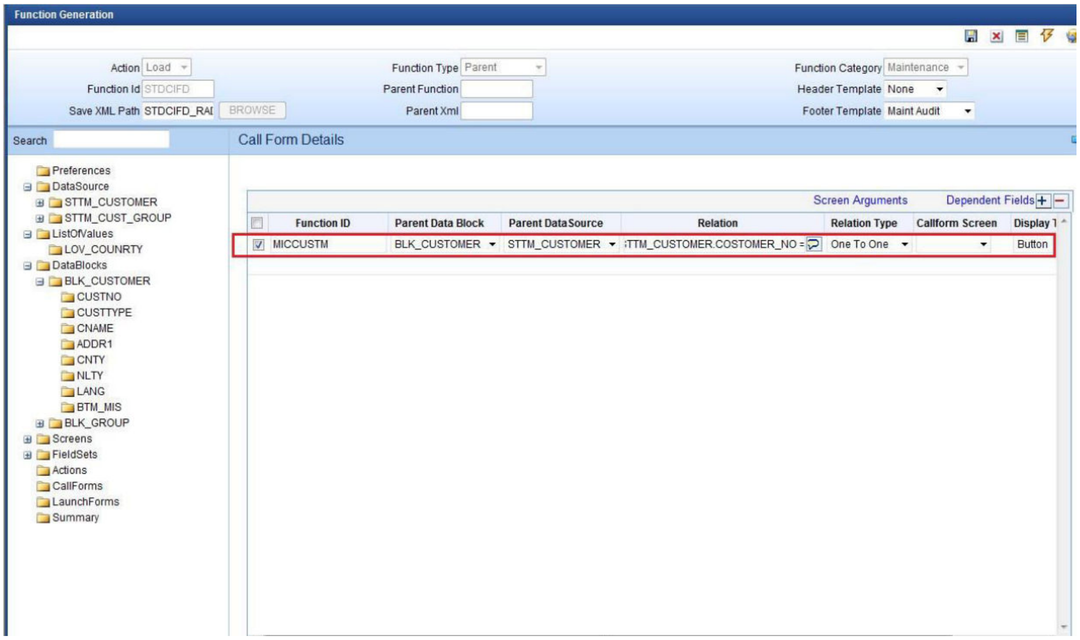
Figure 9-1 Select Fields and Add UI fields



2. Select **UI Fields** tab and click on **Add (+)** to add a row.
3. Enter button name and click **OK**.
4. Select **Display Type** as a button and enter **Field Label**.
5. Add Call form details to the **Call Form** node.

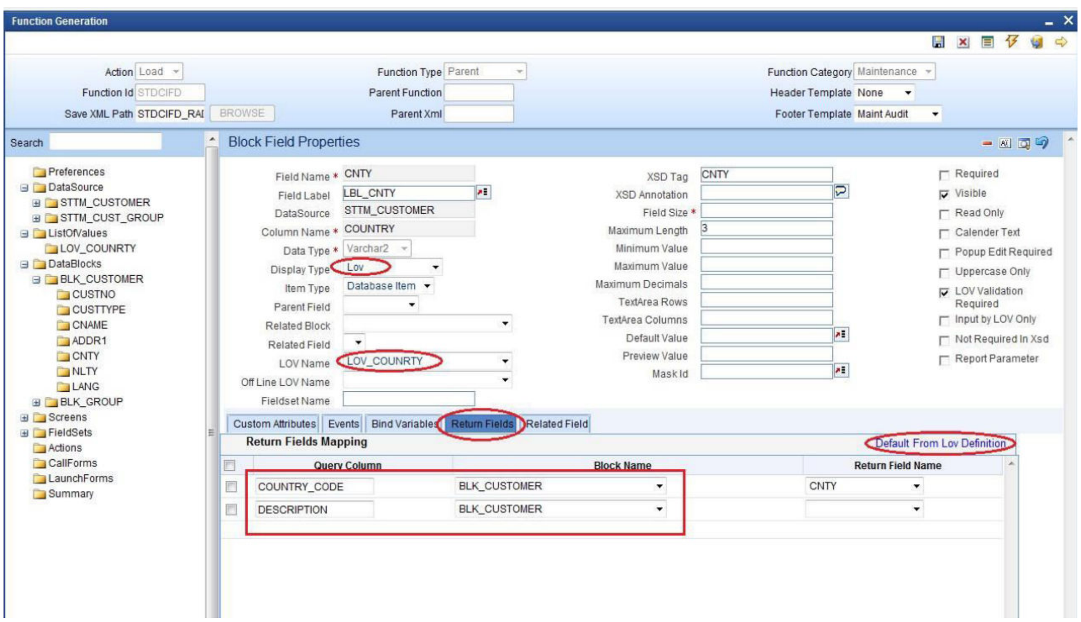
   The **Call Form Details** screen displays.

Figure 9-2   Call Form Details



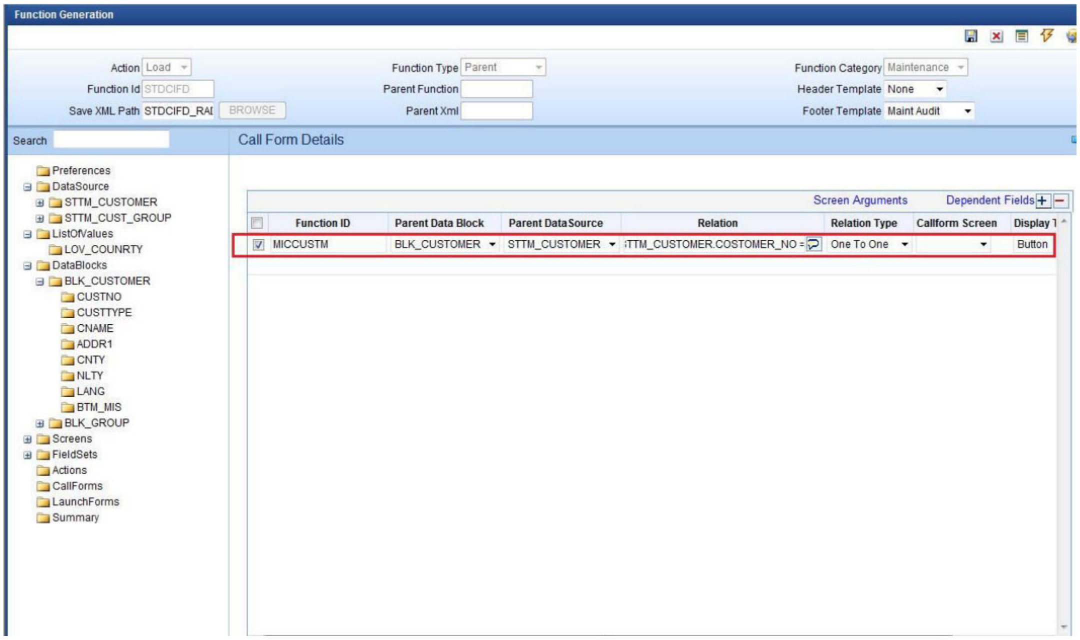
Add event to the button: On selecting event type as **Call Form** or **Launch Form** or **Sub Screen**, button will be displayed on the bottom of the screen. If the user needs to place the button position in the desired place on the screen, the **Event Type** should be **Normal**. The user has to write code in release specific JavaScript file to launch the screen.

Figure 9-3   Block Field Properties



6. Check the preview of the screen with call form buttons.

Figure 9-4 Call Form



10

Create New Field Sets

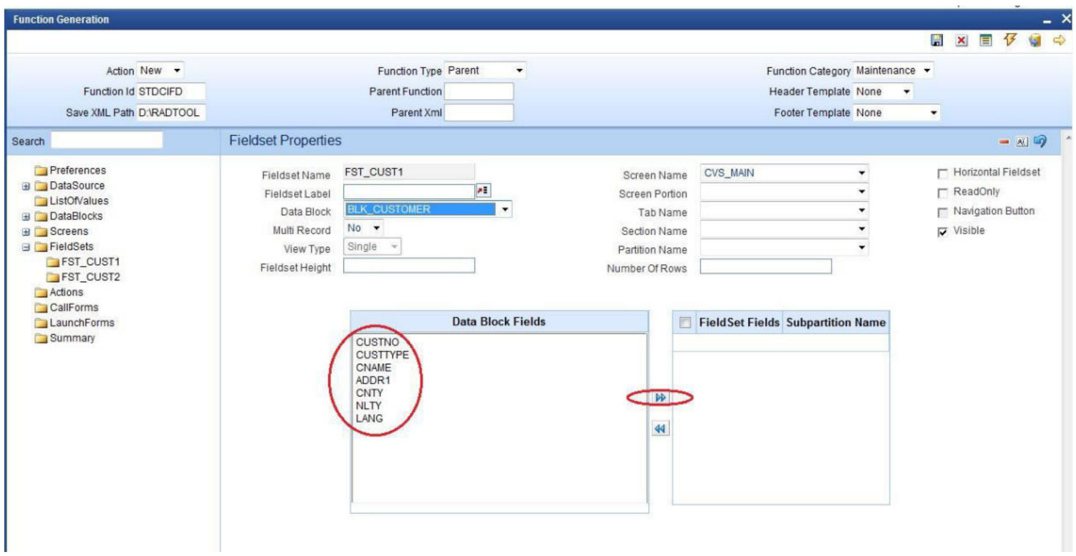
This topic provides systematic instructions to create a new fieldset.

Refer to the topic #unique_33 for the detailed information to create a new Field set. A group of fields can be grouped in a Fieldset which can be placed together on the screen. FieldSet Name should start with **FST_<>**.

1. In the **Fieldset Properties** screen, select the **Data Block** adding to the fieldset.

   All fields available to the block displays in the **Data Block Fields** text area.

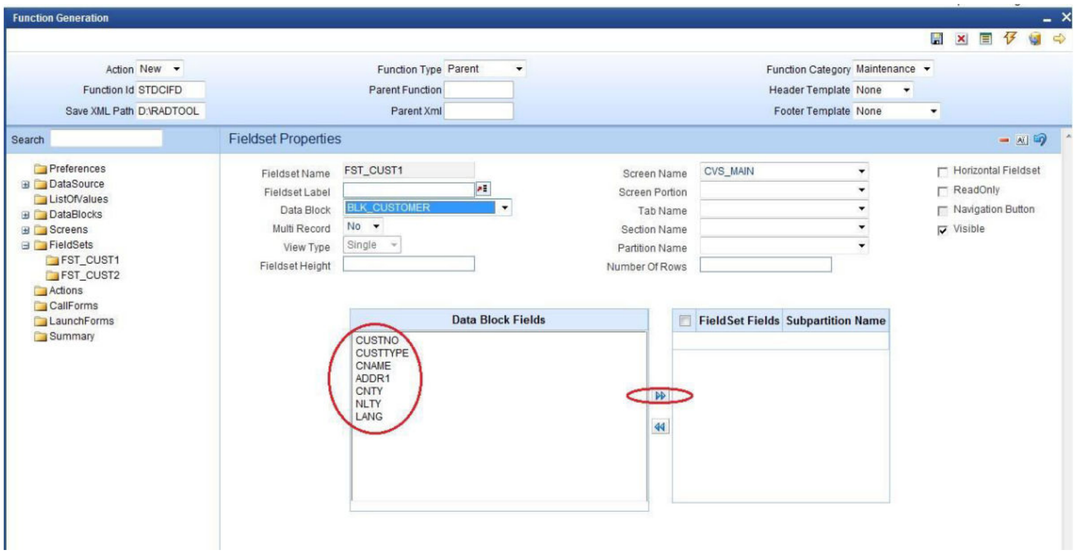
Figure 10-1 Fieldset Properties- Data Block Fields



2. Move fields from **Data Block Fields** to **Fieldset Fields**.

   All fields available to the block displays in the **Fieldset Fields** text area.

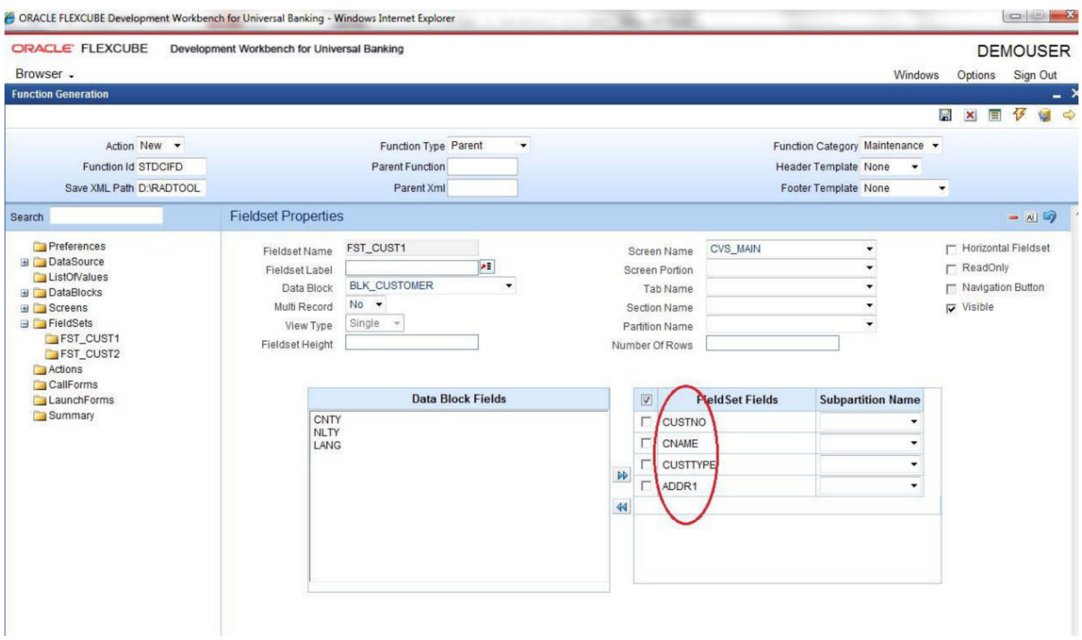
Figure 10-2   Fieldset Properties- Fieldset Fields



3. Select the screen portion (**Header**/**Body**/**Footer**) where this fieldset has to be placed.
4. Select remaining details like **Tab**, **Section**, and **Partition**.

   The **FieldSet Properties** screen displays.

Figure 10-3   Fieldset Properties- Fieldset Details

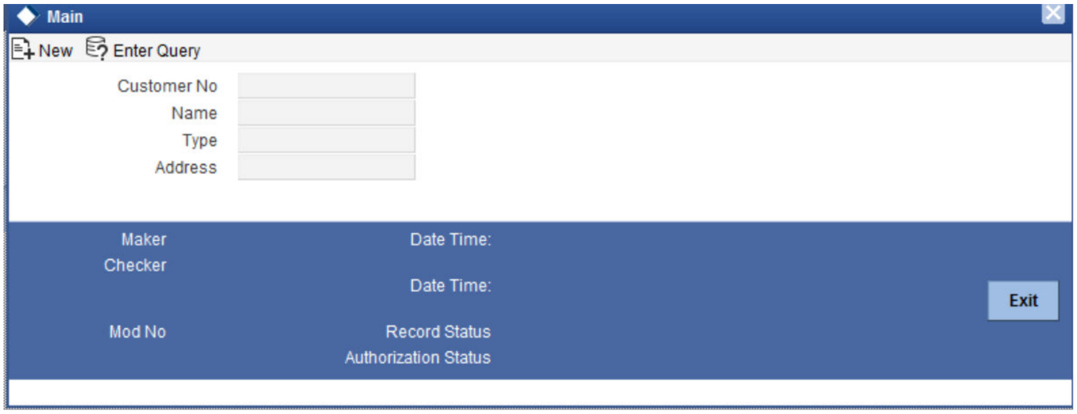


Once fields are added to the field set, the developer can check the preview of the designed screen.

5. Right-click on screen name and click on the **Preview** option.

   The **Preview** screen displays.

Figure 10-4 Preview

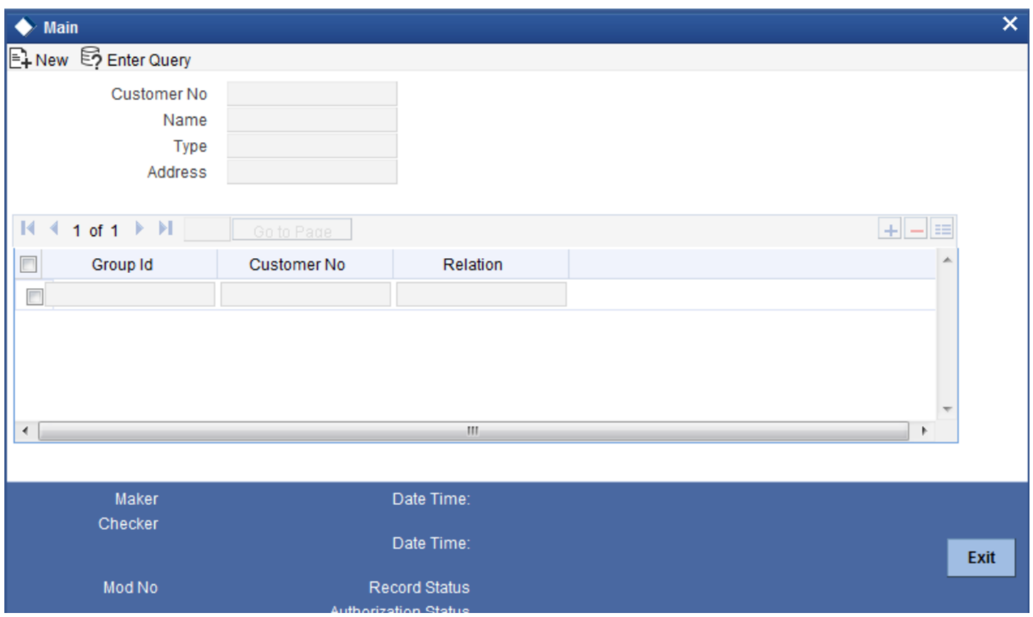


.

6. For adding Multi entry block to the fieldset, follow the below-given steps:
   a. In the **Fieldset Properties** screen, select **Multi Record** field as **Yes**.

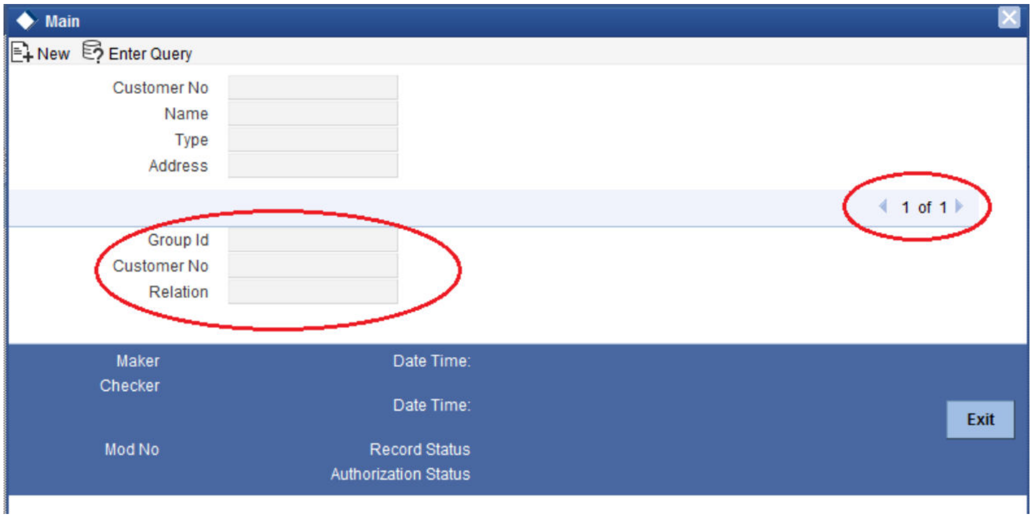
      In the case of Multi records, the **View Type** can be either **Single** or **Multiple** (By Default). The below image shows a multiple view multi-record fieldset:

      Figure 10-5 Preview- Multiple view multi-record Fieldset



   b. Check the **Navigation Button** field for a multi-record single view.

Figure 10-6   Fieldset Properties- Single view multi-record Fieldset



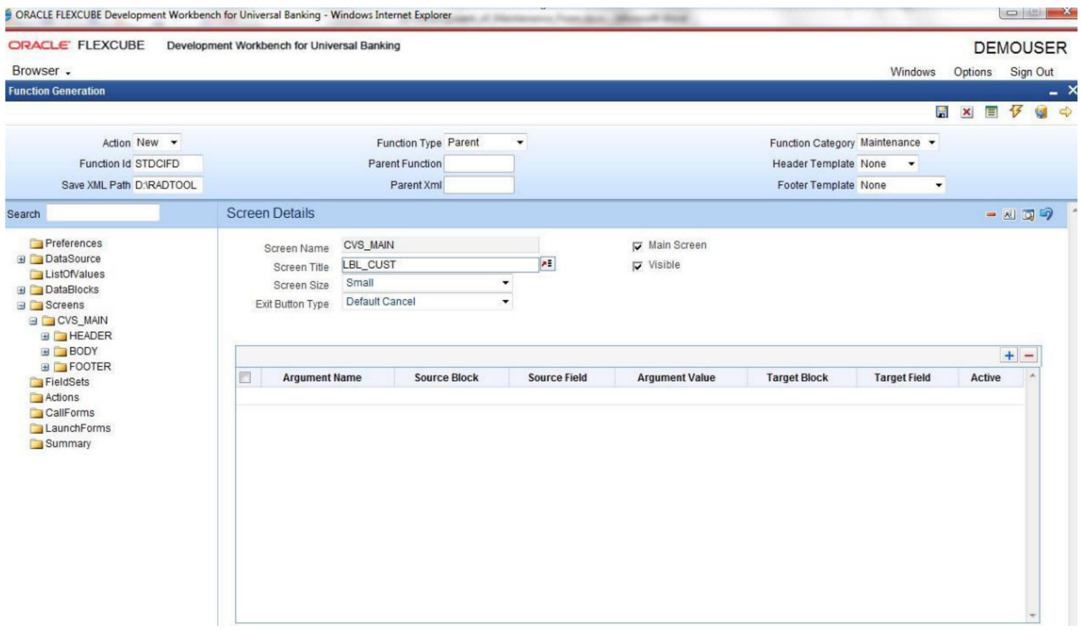
11
Create New Screens

This topic provides systematic instructions to create a new screen.

1. In the **Function Generation** screen, right-click on the **Screens** node and select **Add** option from the right-click menu.

   Add Screen window displays.

Figure 11-1 Add Screen



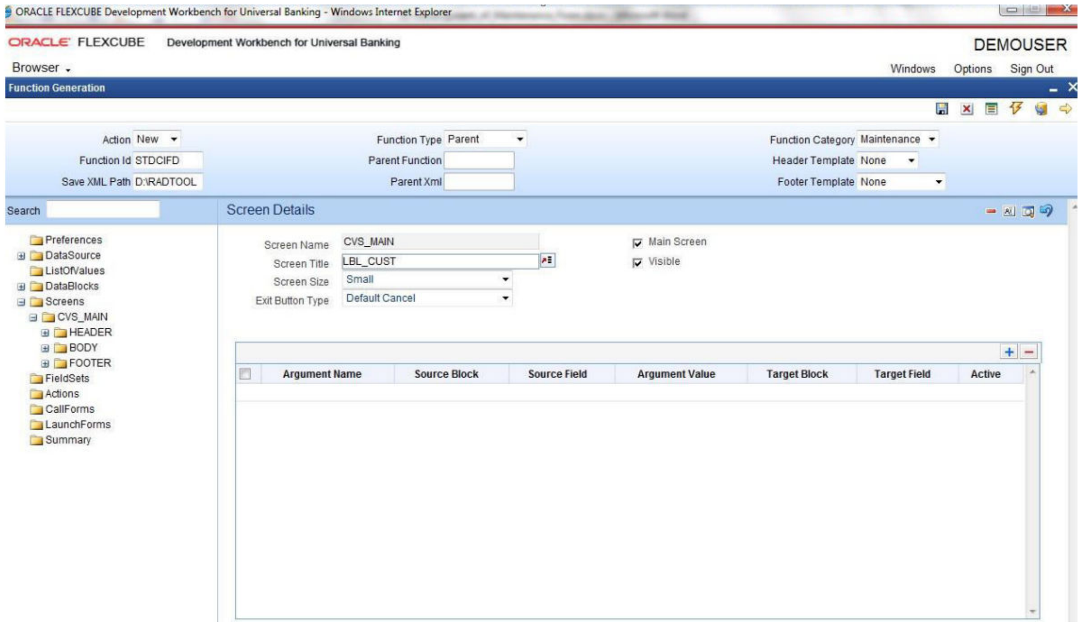
2. Specify the **Screen Name** field.

   Screen Name should start with **CVS_<Name>**.

3. Specify the **Screen Details** and click on the **Save** icon to create a new screen.

   A new screen gets created and displays under the **Screen** node.

Figure 11-2   Screen Details- New Screen



By default screens are divided into 3 parts:

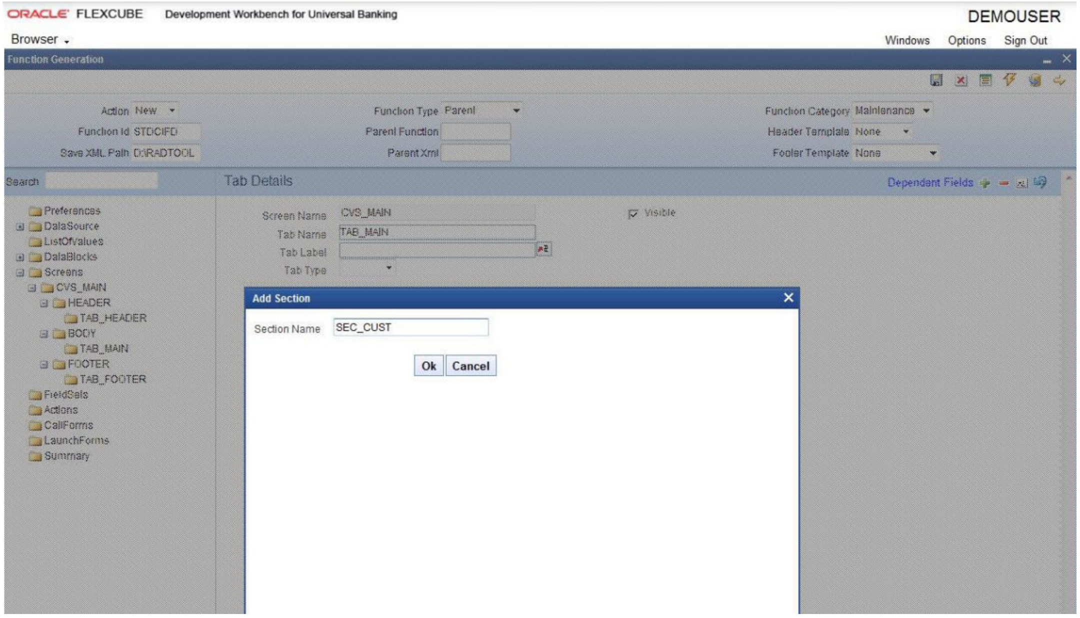
- **Header**
- **Body**
- **Footer**

One Main Screen is Mandatory and **Tabs** can be defined on any of the screen portions as required. User can add **Sections** to **Tabs**. Each section can be divided into **Partitions**.

4. To add a new **Section** to **Tab**, right-click on the **Tab** and then select **Add** option.

   The **Add Section** screen displays.

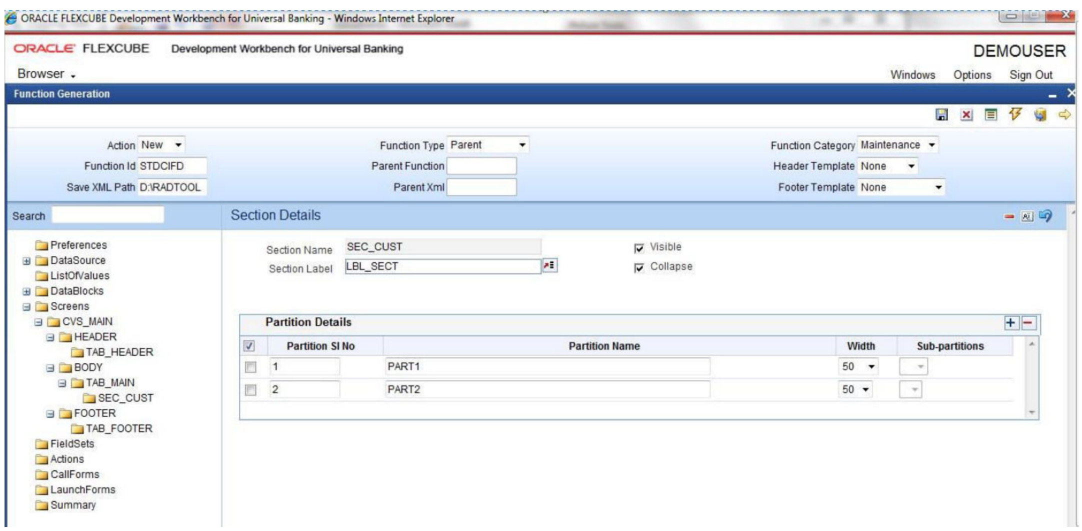
Figure 11-3 Add Section



5. Specify the **Section Name** field and click **OK**.

   A new **Section** gets created and displays under the respective **Tab**.

Figure 11-4 Screen Details



12
Define LOV

This topic provides systematic instructions to define LOVs.

1. To add **LOV**, right-click on the **ListOfValues** node and select **Add** option from the right-click menu.

   List Of values can be defined for the function ID using **LOV** node

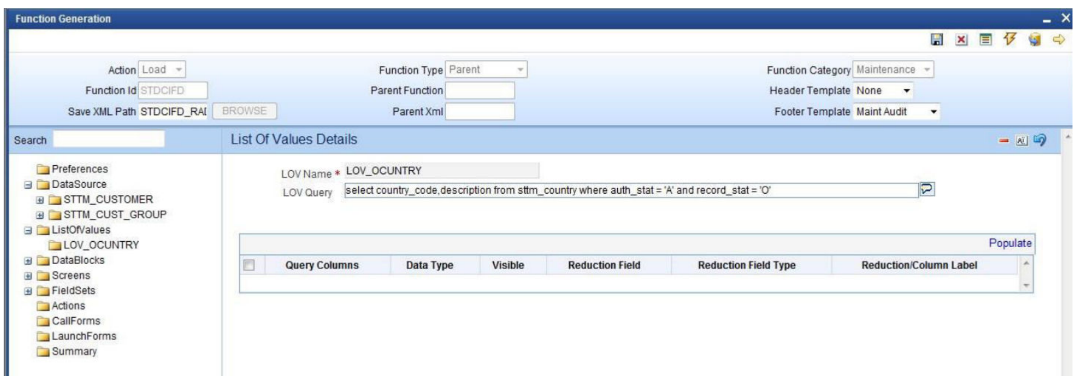
2. Specify the **LOV Name** field and then click **OK**.

   LOV name should start with **LOV_<name>**.

   For Example: **LOV_COUNTRY**

   The **List Of Values Details** screen displays.

Figure 12-1 List Of Values Details



3. Specify the fields in the **List Of Values Details** screen.
4. Enter a valid query and click **Populate**.

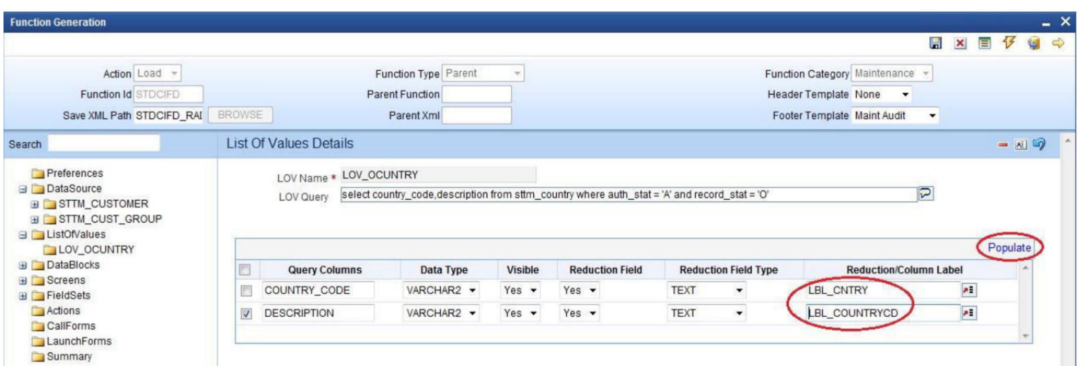
   LOV Query window displays.

Figure 12-2 LOV Query

5. Click **OK**.

   The **List Of Values Details** screen displays.

Figure 12-3 List Of Values Details- Query



> **Note:**
>
> **Reduction/Column Label** are mandatory. If the user won't provide **Reduction/ Column Label** field, an error displays on click **LOV** after deployment in Oracle FLEXCUBE Universal Banking.

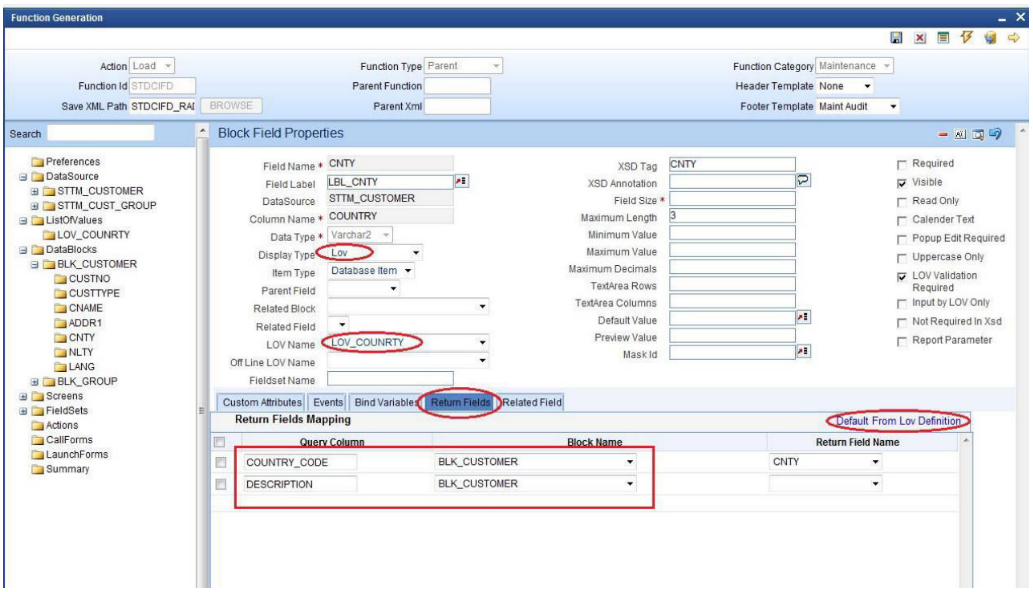
6. After defining LOV, go to block and the corresponding field where the LOV has to be attached.
7. For Block Field Properties to attach LOV to the field follow steps as follows:
   a. Select **Display Type** as **LOV**.
   b. Select the required **Lov Name** from the list of all defined LOV's.
   c. Click on the **Return Fields** tab.

      Return Fields Mapping tab opens.
   d. Click **Default from the Lov Definition**.

      The result fields maintained in the LOV query gets populated.
   e. Select the desired field (and its block) to which the result of the LOV query should have defaulted.

      If the return field is not required to have defaulted to any field in the screen, the return field value can be left blank.

Figure 12-4   Block Field Properties



Use of Bind Variable: If the list of values should be based on any other field value from the screen, bind variables can be used.

For Example: Define Lov as shown in below query, where clause should contain condition with **?**.

```
SELECT cust_ac_no, branch_code, ccy
from sttms_cust_account
where cust_no = ?
and record_stat = 'O'
and once_auth = 'Y'
and ac_stat_de_post = 'Y'
```

   f. In the block field, after selecting **Return Fields**, click on the **Bind Variables** tab.

      Bind Variables Mapping tab opens.

g. Click **Default From LOV Definition**.

   New rows will be created depending on the number of bind variables provided in the LOV query.

h. Select the bind filed in the screen (and its block) for the LOV.

i. Select the **Data Type** of the field.

   Block Field Properties screen displays.

13
Extensibility in Backend Coding

This topic provides an overview of extensibility in back-end coding for the Hook Packages.

Release-specific code has to be written in the Hook Packages generated.

Functions in Hook Packages

Different functions available in the Hook Package of a Maintenance Form are:

1. **Skip Handler**: **Pr_Skip_Handler**
   This can be used to skip the logic written in another release. For Example: logic written in KERNEL release can be skipped in CLUSTER release.
2. **Fn_post_bulid_type_structure**
   If any change has to be made in the field values obtained from the form before the start of processing, code can be written here.
3. **Fn_pre_check_mandatory**
4. **Fn_post_check_mandatory**
   Any extra mandatory checks on the field values from the screen can be written here.
5. **Fn_pre_query**
6. **Fn_post_query**
   Any specific logic while querying can be written in these functions. It is called from **fn_query** of the main package.
7. **Fn_pre_upload_db**
8. **Fn_post_upload_db**
   Any logic while uploading data to tables can be written here.
9. **Fn_pre_default_and_validate**
10. **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.

Flow of control through Hook packages

The flow of control through the Hook Packages for a particular stage is as explained in the figure below:

Figure 13-1 Flow of control through Hook Packages

Example: For **Fn_check_mandatory**, flow will be as:

STPKS_ STDCIFD_MAIN. Fn_Check_Mandatory, and then **STPKS_ STDCIFD_CUSTOM.Fn_Pre_Check_Mandatory**, and then **STPKS_ STDCIFD_CLUSTER.Fn_Pre_Check_Mandatory**, and then **STPKS_ STDCIFD_KERNEL.Fn_Pre_Check_Mandatory**, and then **STPKS_ STDCIFD_MAIN .Fn_Sys_Check_Mandatory**, and then **STPKS_ STDCIFD_KERNEL.Fn_Check_Mandatory**, and then **STPKS_**

STDCIFD_CLUSTER.Fn_Check_Mandatory, and then **STPKS_ STDCIFD_CUSTOM.Fn_Check_Mandatory**

By passing Base Release Functionality

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

The developer also has an option to bypass the base release hook if need be. For example, if the validations are written in **STPKS_STDCINF _KERNEL**, **FN_PRE_CHECK_MANDATORY** are not required or not suitable for the Cluster release, the system provides an option to bypass the code written by the 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: The flow for the Mandatory Stage for STDCIFD:

14 Extensible Development

This topic provides an overview of extensibility in JavaScript coding.

The developer can add code in hook packages and release specific JavaScript files.

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

This JavaScript file allows the 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 ()

Extensibility in Backend Coding

Refer to the topic #unique_38 for detailed information.

15

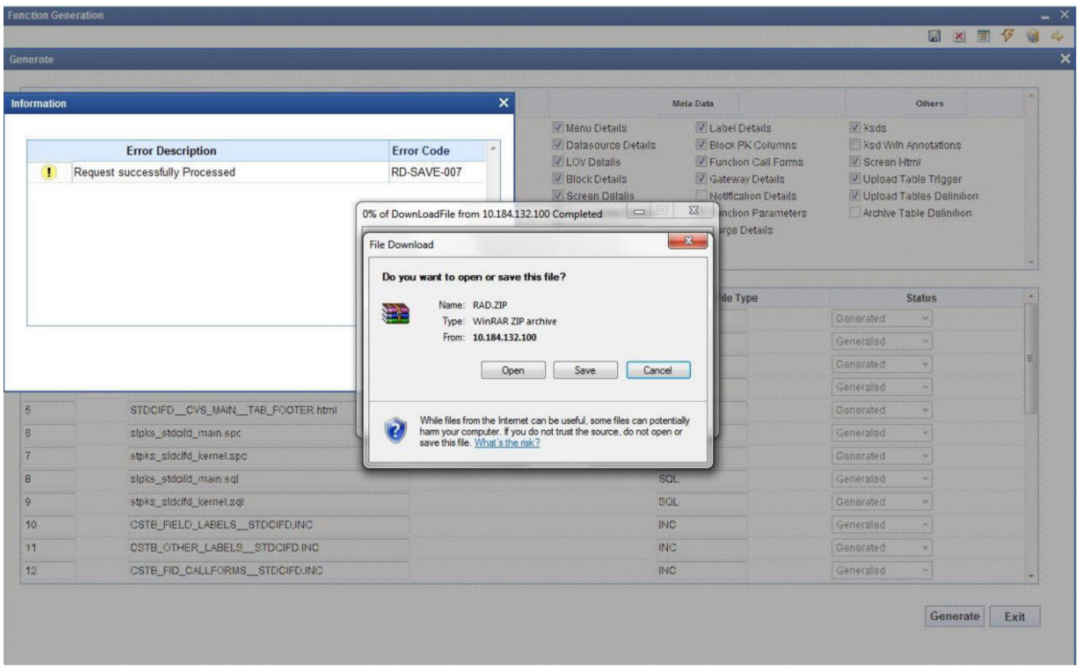
Generate and Deploy Files

This topic explains systematic instructions to generate and deploy files.

1. To generate files, click **Generate**.

   The **Generate** screen displays.

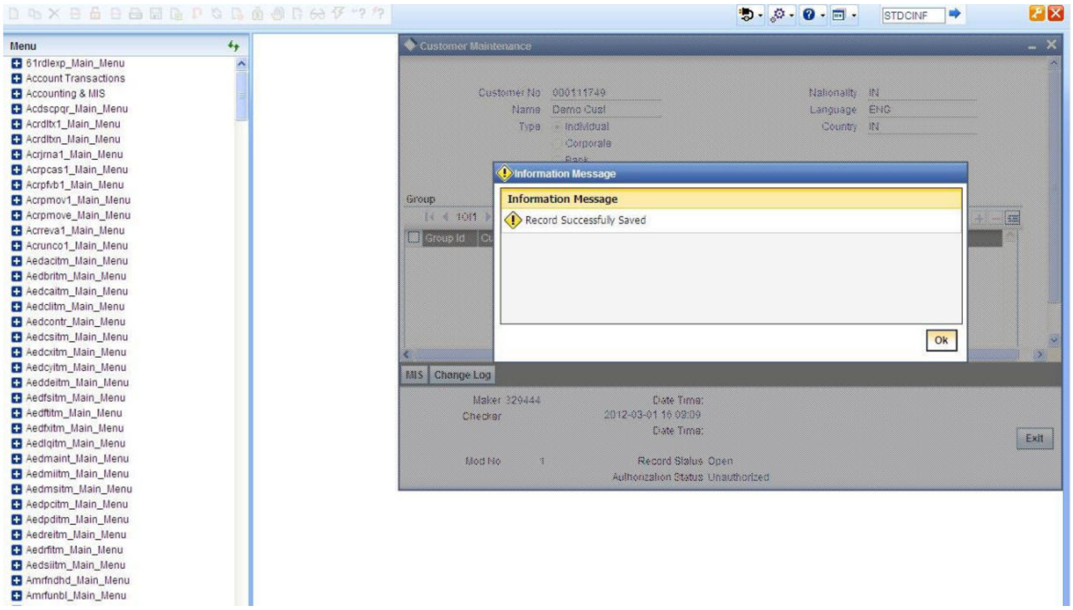
   Figure 15-1   Generate

2. Select the required files to generate, and click on **Generate** button.

   The **Information** window displays with the status.

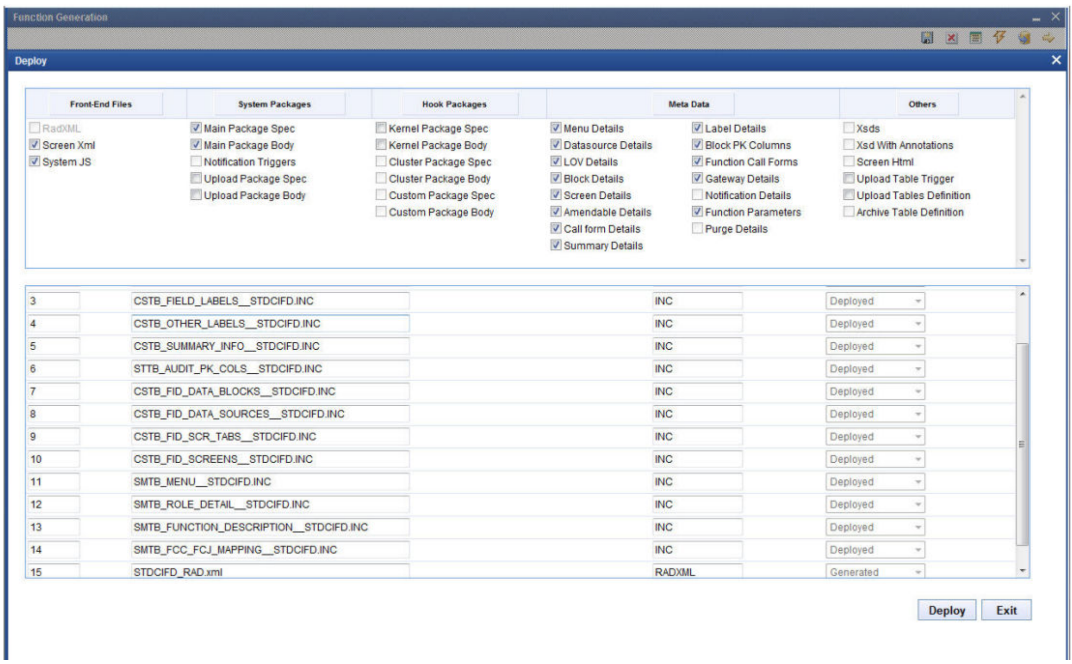
Figure 15-2 Information



3. To deploy files, click on the **Deploy** button.

   The **Deploy** screen displays.

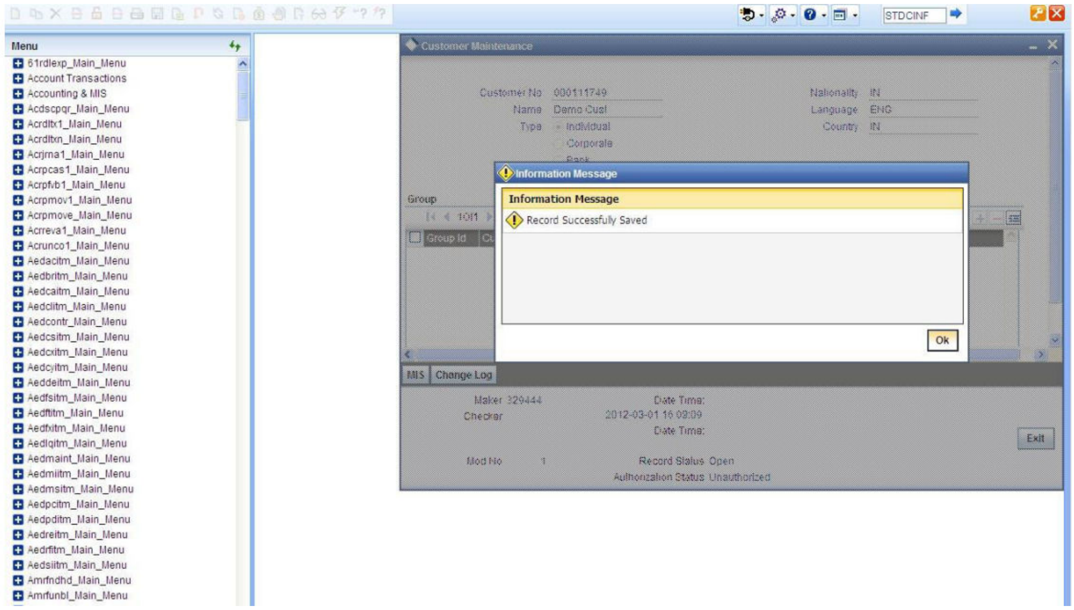
Figure 15-3 Deploy



4. Select the required files to be deployed to the server and then click on **Deploy** button.

   The **Information** window displays with the status.

Figure 15-4   Information



On successful deployment, status will be displayed as deployed.

5. For testing, launch the screen from FLEXCUBE and try sample operations on the screen (**NEW**, **MODIFY**, **QUERY**, etc.)

16
Generated Units

This topic provides an overview of generated units for a Maintenance screen.

The following units will be generated for a Maintenance screen. Refer to document Development Workbench- Screen Development II for a detailed explanation on the same.

Front End Units

Table 16-1 Front End Units

Front End Units	Description
Language XML	This file is an XML markup of presentation details, for the designed Call Form specific to a language.
SYS JavaScript File	This JavaScript file mainly contains a list of declared variables required for the functioning of the screen.
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 specifically in that release.

Data Base Units

Table 16-2 Data Base Units

Data Base Units	Description
Static Scripts	The following static scripts generated are required for the proper functioning of a **Call Form** screen. Refer document on generated units for a detailed explanation. 1. Menu Details Scripts for **SMTB_MENU** and **SMTB_FCC_FCJ_MAPPING**, **SMTB_ROLE_DETAIL**, **SMTB_FCC_GCJ_MAPPING** are required for the functioning of Maintenance screen. 2. LOV Details 3. Amendable Details 4. Label Details 5. Screen Details 6. Block Details 7. Data Source Details 8. Call form Details 9. Summary Details

Table 16-2 (Cont.) Data Base Units

Data Base Units	Description
System Packages	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 Workbench. The main package has the below stages for a maintenance form: • Converting Ts to PL/SQL Composite Type • Checking for mandatory fields • Defaulting and validating the data • Writing into Database • Querying the Data from the database • Converting the Modified Composite Type again to TS Each of these stages has **Pre** and **Post** hooks in the Kernel, Cluster, and Custom Packages. And 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.
Hook Packages	Release-specific packages will be generated based on the release type (KERNEL, CLUSTER or CUSTOM). The developer can add his code in the release-specific hook package. 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 a similar structure and are for the respective teams to work on.

Other Units

XSD: XSD's will be generated if gateway operations are required for the particular function ID. Maintenance for the same has to be done in the Actions node.

17

Maintain Amendable fields

This topic provides systematic instructions to maintain Amendable fields.

Amendable Fields: If the user needs to modify data of a particular field on unlock, in Workbench developer has to maintain fields as amendable.

1. Click on the **Action** node.
2. Click **Amendable** next to the action for which the field has to be made amendable.
3. Select the fields in each block which user can modify for the selected action.
4. Click **OK**.