

Oracle® Retail Workspace Retail Library

Reference Guide

Release 13.1

January 2011

Note: The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Contents

1	Introduction	
	Pre-Requisites	1-2
	Accessing the Workspace Retail Library.....	1-2
	Configuring an RMS Data Source.....	1-3
2	Create Regular Item	
	Functional Design	2-1
	Technical Design	2-2
	Overview	2-2
	Generating Java Objects for RIB types	2-2
	Creating Business Components	2-2
	User Interface Design	2-4
	Validations	2-5
	Calling the RMS API.....	2-5
	Using the Create Regular Item Micro Application.....	2-5
	Limitations.....	2-6
	Additional Customizations	2-6
3	Create Purchase Order	
	Functional Design	3-1
	Technical Design	3-2
	Overview	3-2
	Generating Java Objects for RIB Types.....	3-2
	Creating Business Components	3-3
	User Interface Design	3-5
	Passing Input Parameters to the Application	3-6
	Validations	3-6
	Calling the RMS API.....	3-6
	Using the Application	3-7
	Limitations.....	3-7
	Additional Customizations	3-8
4	Cancel Purchase Order	
	Functional Design	4-1
	Technical design	4-2

Overview	4-2
Generating Java Objects for RIB types	4-2
Creating Business Components	4-3
User Interface Design	4-4
Passing Input Parameters to the Application	4-4
Validations	4-5
Calling the RMS API.....	4-5
Using the Application	4-5
Limitations	4-6
Additional Customizations	4-6

5 Item Transfer

Functional Design	5-1
Technical design	5-2
Overview	5-2
Generating Java Objects for RIB types	5-2
Creating Business Components	5-3
User Interface Design	5-5
Passing Input Parameters to the Application	5-5
Validations	5-6
Calling the RMS API.....	5-6
Using the Application	5-7
Limitations	5-7
Additional Customizations	5-7

6 Update Vendor

Functional Design	6-1
Technical design	6-2
Overview	6-2
Generating Java Objects for RIB Types	6-2
Creating Business Components	6-4
User Interface Design	6-4
Passing Input Parameters to the Application	6-4
Validations	6-5
Calling the RMS API.....	6-5
Using the Application	6-6
Limitations	6-6
Additional Customizations	6-6

7 Bulk Cancellation/Approval of Purchase Orders

Functional Design	7-2
Technical Design	7-2
Overview	7-2
Generating Java Objects for RIB Types	7-3
Creating Business Components	7-4
User Interface Design	7-5

Validations	7-6
Calling the RMS API.....	7-6
Application Usage	7-6
Limitations	7-7
Additional Customizations	7-7

8 Maps

Functional Design	8-1
Technical Design	8-2
Components of the Sample Base Application.....	8-2
RMS Database Table for the Maps Micro Application	8-3
DDL for the Database Table	8-3
Sample Data for the Database Table	8-4
Run the Micro Application	8-4
Validations	8-6
Using the Application	8-6
Limitations	8-7
Additional Customizations	8-7
Creating a View Object.....	8-8
Exposing the View Object for the User Interface.....	8-8
Adding the Theme to the Base Map	8-9
Setting Up the Theme Using a Configuration File	8-9

Introduction

Oracle Retail Workspace Retail Library Release 13.1 provides a set of Micro Applications that can be used to perform some of the operations of Oracle Retail Merchandising System (RMS) from outside the application. These Micro Applications have evolved from the thought of providing simple, intelligent, and rich interfaces to the users of the Retail applications. The Micro Applications act like widgets and can be deployed on Oracle Retail Workspace (ORW).

Based on your business need, you can choose to make all or some available in the workspaces for users. This document describes the functional and technical design details of the Micro Applications available in this release. It includes the following:

- [Create Regular Item](#)
- [Create Purchase Order](#)
- [Cancel Purchase Order](#)
- [Item Transfer](#)
- [Update Vendor](#)
- [Bulk Cancellation/Approval of Purchase Orders](#)
- [Maps](#)

Each Micro Application is used to perform a specific retail business operation. They can be used as a stand-alone application or be launched from other Web-based applications. When launched from another application, you have the flexibility of passing input parameters to the Micro Application that was launched. Passing parameters between the application pages enhances the usability of the applications. There are many parameters on each application page. When an application page is launched from another page, the value for one or many of the parameters present on the called page can be passed.

For example, a user is on a page that displays all the items present in a store. If the user wants to place a purchase order for one of the items, user selects the specific item and launches a Create Purchase Order page. The location and item value parameters are then passed from the current application page to the Create Purchase Order page. When the Create Purchase Order page is rendered, the location and item values are assigned to the relevant form fields automatically. The user then just has to select the supplier value manually and create a purchase order.

Pre-Requisites

Before proceeding, you must take the following into consideration:

- Information included in this document is for reference-purposes only.
- This document is intended for application integrators and implementation personnel who are familiar with the Oracle Application Development Framework (ADF), specifically ADF Faces, ADF Business Components, and ADF Task Flows. Knowledge of Oracle Retail Merchandising System (RMS) and Retail Integration Bus (RIB), specifically X APIs and RIB objects, is also required.
- It includes information that will help you create your own version of the Micro Application based on your business need. It is your responsibility to ensure that the application meets all the business requirements.
- You may need to make additional customizations, for the application work with your business workflows.
- The Micro Applications available in this release do not include any in-built security features. It is recommended that you implement any relevant security features required and used by your business.

Accessing the Workspace Retail Library

The Workspace Retail Library package is available on My Oracle Support with the patch ID **10632792**.

To access the Workspace Retail Library package:

1. In a Web browser, open the following URL:
<https://support.oracle.com/>
The My Oracle Support Web page appears.
2. Select a language and sign on to the Web site by clicking **Sign In**.
Once signed in, the **My Oracle Support | Dashboard** screen appears.
3. Click the **Patches & Updates** tab.
4. On the **Patch & Updates** screen, under **Patch Search**, click **Patch ID or Number**.
5. In the **Patch ID or Number is** field, enter **10632792**.
6. Optionally, you can also choose a platform from the **Platform is** drop-down list.
7. Click **Search**. The **Patch Search Results** screen appears.
8. In the **Patch Search Results** screen, under **Patch ID**, click the relevant patch.
9. On the next screen, click **Download** (appears on the left side of the screen).

Note: On the Patch Search Results screen, you can also select the row that matches the patch description, and then click Download on the toolbar that appears under the selected row.

10. Unpack the ZIP file to your working directory.

Configuring an RMS Data Source

Micro Applications are packaged as ADF JAR libraries. They can be included in any ADF-based Web application. All Micro Applications are based on RMS data source that you must define before any of these JAR libraries are deployed.

You must create a data source pointing to an RMS schema with the following JNDI name in the application server where the final application will be deployed:

```
jdbc/RMSDataSource
```

Create Regular Item

Creating an item is one of the most widely performed actions in RMS and new items are added to the database frequently. The *Create Regular Item* Micro Application is a simple and efficient application that can be used to create items instantly, especially when there is a demand to create them quickly. It takes only a few mandatory attributes of the item, creates the item in RMS, and inserts them in the RMS database quickly. It interacts with the RMS database to fetch the required data and commit the changes made by the user. If the users want to add more details to the same item, they can later edit the item in the RMS application.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical Design](#)
- [Using the Create Regular Item Micro Application](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Create Regular Item Micro Application enables the users to create a regular item without accessing the RMS application. It communicates synchronously with RMS through the external APIs exposed by the RMS application. It sends the item details through the API. The API in RMS does validations and then creates the item with a new item number and inserts the item, in the Worksheet (W) status, in the RMS database.

Note: API does not perform all the validations or operations as they were performed when item is created using the RMS application. You may need to make the necessary changes to meet your business requirements.

This solution does not modify any code in RMS. It uses the existing APIs to perform the operations. The Create Regular Item Micro Application takes the item details entered by the user and passes them as arguments to the external API with the Worksheet (W) status.

Note: The external APIs provide a limited functionality.

The RMS application then processes the message sent by the application and inserts the item in the RMS database tables. The processing done on the RMS side is not in scope of this solution. You may need to modify the RMS processes to achieve the desired results.

Technical Design

This section highlights the technical design of the Create Regular Item Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

- [Overview](#)
- [Generating Java Objects for RIB types](#)
- [Creating Business Components](#)
- [User Interface Design](#)
- [Validations](#)
- [Calling the RMS API](#)

Overview

The Create Regular Item Micro Application provides an user interface where the minimum required details to create an item are captured. The application then calls the RMS APIs to create an item with the details specified by the user. However, the RMS APIs require the RIB objects as input parameters. Based on the RIB-Type Objects, Application builds the corresponding Java objects and populates the objects, with the information gathered from the user interface and RMS database. A JDBC call is then made to the RMS API, passing the Java object as a parameter. The RMS API is then executed on the RMS database schema, which updates the corresponding tables in the database.

Generating Java Objects for RIB types

To create an item, the *RIB_XItemDesc_REC* RIB object is passed to the RMS X API. The application uses this RIB object to create the Java objects. This automatically generates the relevant Java objects for all the inner objects.

For more information on the structure of the RIB object, refer to the RIB XSDs.

Creating Business Components

The following SQL-based business components have been used to create the Micro Application:

- **DepartmentVO**

To fetch the list of departments:

```
SELECT dept AS DEPARTMENT_ID,  
       dept_name AS DEPARTMENT_NAME,  
       concat( concat(dept, '-'), dept_name) AS DEPARTMENT_SEARCH  
FROM depts
```

- **ClassVO**

To populate the list of classes under the selected department:

```
SELECT class AS CLASS_ID,
       class_name AS CLASS_NAME,
       concat( concat( class, '-' ), class_name) AS CLASS_SEARCH
FROM class
WHERE dept=:BindVarDeptID
```

- **SubclassVO**

To populate the list of subclasses corresponding to the department and class selected:

```
SELECT subclass AS SUBCLASS_ID,
       sub_name AS SUBCLASS_NAME,
       concat( concat( subclass, '-' ), sub_name) AS SUBCLASS_SEARCH
FROM subclass
WHERE dept=:BindVarDeptID AND class=:BindVarClassID
```

- **SupplierVO**

To populate the list of suppliers:

```
SELECT supplier AS SUPPLIER_ID,
       sup_name AS SUPPLIER_NAME,
       concat( concat( supplier, '-' ), sup_name) AS SUPPLIER_SEARCH
FROM sups WHERE supplier_parent IS NULL
AND status = 'A'
```

- **SupplierSiteVO**

To populate the list of supplier sites:

```
SELECT supplier AS SUPPLIER_ID,
       sup_name AS SUPPLIER_NAME,
       concat( concat( supplier, '-' ), sup_name) AS SUPPLIER_SEARCH
FROM sups WHERE supplier_parent IS NOT NULL
AND status = 'A'
```

- **SystemOptionsVO**

This VO is used to fetch the value of supplier_sites_ind from system_options table:

```
SELECT SystemOptions.SUPPLIER_SITES_IND
FROM SYSTEM_OPTIONS SystemOptions
```

- **CountryVO**

To populate the list of countries:

```
SELECT country_id AS COUNTRY_ID,
       country_desc AS COUNTRY_DESC,
       concat( concat( country_id, '-' ), country_desc) AS COUNTRY_SEARCH
FROM country
```

- **ItemNumberTypeVO**

To populate the types of item numbers:

```
SELECT code AS ITEM_NO_TYPE_CODE,
       code_seq AS ITEM_NO_CODE_SEQ,
       code_desc AS ITEM_NO_CODE_DESC
FROM code_detail WHERE code_type='UPCT'
ORDER BY code_seq
```

- **RetailPriceZoneGroupVO**

To populate the list of retail price zone groups:

```
SELECT zone_group_id AS RETAIL_PRICE_ZONE_GRP_ID,
       name AS RETAIL_PRICE_ZONE_NAME,
       concat(concat(zone_group_id, '-'),name) AS RETAIL_PRICE_ZONE_SEARCH
FROM rpm_zone_group
```

- **CostZoneGroupVO**

To populate the list of cost zone groups:

```
SELECT zone_group_id AS ZONE_GROUP_ID,
       description AS ZONE_DESCRIPTION,
       concat(concat(zone_group_id, '-'),description) AS ZONE_SEARCH
FROM cost_zone_group
```

- **StandardUomVO**

To populate the list of unit of measures available:

```
SELECT uom AS UOM_ID,
       uom_desc AS UOM_DESC,
       concat(concat(uom, '-'), uom_desc) AS UOM_SEARCH
FROM uom_class
```

User Interface Design

The following figure displays the user interface for the Micro Application:

Figure 2–1 User Interface for the Create Regular Item Micro Application

The screenshot shows a web-based form for creating a regular item. It features the following elements:

- A list of required fields (marked with asterisks):
 - * Item Description
 - * Department
 - * Class
 - * Subclass
 - * Supplier Site
 - * Country
 - * Unit Cost
- A "Hide" section containing:
 - An "Edit Advanced Field" checkbox.
 - Item Number Type Code: ITEM
 - Item Number (Manual):
 - Retail Price Zone Group ID: 1
 - Cost Zone Group: 1000
 - Standard Unit Of Measure: EA
 - UOM Conversion Factor: 1
- "Create" and "Cancel" buttons at the bottom.

Validations

The following validations are implemented in the application:

- Depending on the supplier sites indicator, the page displays the supplier or supplier site.
- Users can only select valid class and subclass values.
- Only approved suppliers or supplier sites can be selected.
- All validations imposed by RMS will be executed on the RMS side.

Calling the RMS API

Once the item details entered by the user is captured, the application uses it to build the required type of the RIB object (RIB_XItemDesc_REC) and call the RMS API passing the RIB object as the parameter when the Create Item button is pressed.

Signature of the API used to create an item is as follows:

```
RMSSUB_XITEM.CONSUME (O_STATUS_CODE,
                      O_ERROR_MESSAGE,
                      L_RIB_XITEMDESC_REC,
                      I_MESSAGE_TYPE);
```

Where,

- O_STATUS_CODE is an IN OUT parameter which holds the status of the transaction.
 - O_STATUS_CODE = 'S' indicates that the transaction was successful.
 - O_STATUS_CODE = 'E' indicates that an error had occurred during the transaction.
- O_ERROR_MESSAGE is an IN OUT parameter that holds the error message.
- L_RIB_XITEMDESC_REC is an IN parameter that holds the item details.
- I_MESSAGE_TYPE is the message type. To create an item, message type to be passed is 'xitemcre'.

Using the Create Regular Item Micro Application

To use the Create Regular Item Micro Application:

1. In the **Item Description** field, enter the item name or description.
2. In the **Department** field, select the department it belongs to.
3. In the **Class** and **Subclass** fields, select the relevant class and subclass of the item.
4. In the **Supplier Site** field, select the primary supplier of the item.
5. In the **Country** field, select the supplier country.
6. In the **Unit Cost** field, enter unit cost of the item.
7. Fields in the advanced section are assigned default values. If those values need to be edited, check the **Edit Advanced Field** check box.
8. In the **Item Number Type Code** field, select the item number type.

9. The item number is automatically populated for all the item number types, except for the manual type. If item type is manual, enter the item number in the **Item Number (Manual)** field.
10. In the **Retail Price Zone Group ID** and **Cost Zone Group** fields, select the retail price zone and cost zone group values.
11. In the **Standard Unit of Measure** field, select the standard unit of measure (UOM) for the item.
12. If the selected UOM is not EACH, enter the conversion factor for the UOM in the **UOM Conversion Factor** field.
13. Click the **Create button** to create the item.

If the transaction needs to be cancelled or the page needs to be reset, click the **Cancel** button.

New item is created and inserted into the RMS database. A confirmation message is printed at the bottom of the page with the item number created.

Limitations

The following limitations apply to this Micro Application:

- Only simple and regular items can be created using the application.
- Only one supplier-country-loc combination can be attached to the item and this combination is taken as the primary detail by default.
- Indicators like Sellable, Orderable, Inventory, and Forecast-able default to the value 'Y'.
- Items with transaction level as 'Line' can only be created.
- Locations cannot be added to the item through this application. Use the RMS application to enter the location information.
- Only the parameters present on the page are passed to the RMS API. RMS application has to be used to add more parameter values to the item.
- Application supports limited functionality that can be achieved using the parameters on the user interface.
- Application does not check the privileges of the user to create items.
- Security is not implemented in the application.

Additional Customizations

You may need to implement any additional customization that is not handled through the RMS API, but required by the business process.

Create Purchase Order

Creating a purchase order requires the user to log on to the RMS application, navigate to the Order screen, select the Create Order option, fill all the required details, add items and location to the order, and then submit it. The Create Purchase Order (PO) Micro Application is a simple and efficient application that simplifies this process. It interacts with the RMS database to fetch the required data and commit the changes made by the user.

Users can use this Micro Application to create a purchase order by providing the minimum required details on a single screen. They just need to launch the Micro Application from Workspace, enter the location, supplier, and item details, and then click the Create Order button.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical Design](#)
- [Using the Application](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Create Purchase Order Micro Application enables the users to create a purchase order without accessing the RMS application. It communicates synchronously with RMS through the external APIs exposed by the RMS application. It sends the order details through the API. The API in RMS does validations and then creates the order with a new order number and inserts the order, in the Worksheet (W) status, in the RMS database.

Note: API does not perform all the validations or operations as they were performed when item is created using the RMS application. You may need to make the necessary changes to meet your business requirements.

This solution does not modify any code in RMS. It uses the existing APIs to perform the operations. The Create Purchase Order Micro Application takes the order details entered by the user and passes them as arguments to the external API with the Worksheet (W) status.

Note: The external APIs provide a limited functionality.

The RMS application then processes the message sent by the application and inserts the item in the RMS database tables. The processing done on the RMS side is not in scope of this solution. You may need to modify the RMS processes to achieve the desired results.

Technical Design

This section highlights the technical design of the Create Purchase Order Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

- [Overview](#)
- [Generating Java Objects for RIB Types](#)
- [Creating Business Components](#)
- [User Interface Design](#)
- [Passing Input Parameters to the Application](#)
- [Validations](#)
- [Calling the RMS API](#)

Overview

The Create Purchase Order Micro Application provides an user interface where the minimum required details to create an order are captured. The application then calls the RMS APIs to create an order with the details specified by the user. However, the RMS APIs require the RIB objects as input parameters. Based on the RIB-Type Objects, Application builds the corresponding Java objects and populates the objects, with the information gathered from the user interface and RMS database. A JDBC call is then made to the RMS API, passing the Java object as a parameter. The RMS API is then executed on the RMS database schema, which updates the corresponding tables in the database.

Generating Java Objects for RIB Types

To create a purchase order, the *RIB_XOrderDesc_REC* RIB object is passed to the RMS X API. This object contains a collection of *RIB_XOrderDtl_REC* objects. The application uses the *RIB_XOrderDesc_REC* RIB object to create the Java objects. This automatically generates the relevant Java objects for the *RIB_XOrderDtl_REC* object.

Structure of the *RIB_XOrderDesc_REC* object:

```
order_no varchar2(10),
supplier varchar2(10),
currency_code varchar2(3),
terms varchar2(15),
not_before_date date,
not_after_date date,
otb_eow_date date,
dept number(4),
status varchar2(1),
exchange_rate number(20,10),
include_on_ord_ind varchar2(1),
```

```
written_date date,
XOrderDtl_TBL "RIB_XOrderDtl_TBL",
orig_ind varchar2(1),
edi_po_ind varchar2(1),
pre_mark_ind varchar2(1),
user_id varchar2(30),
comment_desc varchar2(2000),
ExtOfXOrderDesc_TBL "RIB_ExtOfXOrderDesc_TBL"
```

Structure of RIB_XOrderDtl_REC:

```
item varchar2(25),
location number(10),
unit_cost number(20,4),
ref_item varchar2(25),
origin_country_id varchar2(3),
supp_pack_size number(12,4),
qty_ordered number(12,4),
location_type varchar2(1),
cancel_ind varchar2(1),
reinststate_ind varchar2(1),
ExtOfXOrderDtl_TBL "RIB_ExtOfXOrderDtl_TBL"
```

Creating Business Components

The following SQL-based business components have been used to create the Micro Application:

- **LocationVO**

To populate the LOV for location where order needs to be created:

```
SELECT distinct s.store Location,
s.store_name LocName,
'S' LocType,
concat(concat(s.store, '-'),s.store_name) AS Location_Search,
s.org_unit_id OrgUnitIdb
FROM v_STORE s,
PARTNER_ORG_UNIT pou,
SYSTEM_OPTIONS so
WHERE s.stockholding_ind = 'Y'
AND s.store_type = 'C'
AND (get_vdate < (s.store_close_date-s.stop_order_days)
OR s.store_close_date IS NULL)
AND ((so.supplier_sites_ind = 'N'
AND pou.partner_type = 'S'
AND pou.partner = NVL(:ParamSupp,pou.partner)
AND s.org_unit_id = pou.org_unit_id)
OR (so.supplier_sites_ind = 'Y'
AND pou.partner_type = 'U'
AND pou.partner = NVL(:ParamSupp,pou.partner)
AND s.org_unit_id = pou.org_unit_id))

UNION ALL

SELECT distinct wh.wh Location,
wh.wh_name LocName,
'W' LocType,
concat(concat(wh.wh, '-'),wh.wh_name) AS Location_Search,
wh.org_unit_id OrgUnitIdb
```

```

FROM v_WH wh,
     PARTNER_ORG_UNIT pou,
     SYSTEM_OPTIONS so
WHERE wh.stockholding_ind = 'Y'
AND ((so.supplier_sites_ind = 'N'
AND pou.partner_type = 'S'
AND pou.partner = NVL(:ParamSupp,pou.partner)
AND wh.org_unit_id = pou.org_unit_id)
OR (so.supplier_sites_ind = 'Y'
AND pou.partner_type = 'U'
and pou.partner = NVL(:ParamSupp,pou.partner)
AND wh.org_unit_id = pou.org_unit_id))
ORDER BY LocName

```

- **VSupMsobYVO**

To select the supplier for the order:

```

SELECT distinct sups.supplier Supplier,
     sups.SUP_NAME SupplierName,
     concat(concat(sups.supplier, '-'), sups.SUP_NAME) AS Supplier_Search
FROM SYSTEM_OPTIONS so,
     V_SUPS sups,
     PARTNER_ORG_UNIT pou,
     STORE,
     WH,
     ITEM_SUPPLIER
WHERE sups.sup_status = 'A'
AND pou.partner = sups.supplier
AND ITEM_SUPPLIER.item = NVL(:ParamItem,ITEM_SUPPLIER.item)
AND sups.supplier = DECODE(:ParamItem,NULL,sups.supplier,ITEM_
SUPPLIER.supplier)
AND NVL(:ParamLoc,1) = DECODE(:ParamLocType,'S',STORE.store,'W',WH.wh,1)
AND pou.org_unit_id = DECODE(:ParamLocType,'S',STORE.org_unit_id,'W',WH.org_
unit_id,NVL(:OrgUnitID,pou.org_unit_id))
AND ((so.supplier_sites_ind = 'Y'
AND supplier_parent IS NOT NULL
AND pou.PARTNER_TYPE = 'U')
OR (so.supplier_sites_ind = 'N'
AND supplier_parent IS NULL
AND pou.partner_type = 'S'))
ORDER BY 2

```

- **ItemsVO**

To select the items associated with the supplier and also to get the details of a particular item:

```

SELECT VItemMaster.ITEM,
     VItemMaster.ITEM_DESC,
     ItemSuppCountry.ORIGIN_COUNTRY_ID,
     ItemSuppCountry.ITEM AS ITEM1,
     ItemSuppCountry.SUPPLIER,
     ItemSuppCountry.UNIT_COST
FROM ITEM_SUPP_COUNTRY ItemSuppCountry,
     V_ITEM_MASTER VItemMaster
WHERE VItemMaster.ITEM = ItemSuppCountry.ITEM
AND ItemSuppCountry.supplier = NVL(:bindSupplier, ItemSuppCountry.supplier)
AND ItemSuppCountry.primary_country_ind = 'Y'
AND ItemSuppCountry.unit_cost > 0
AND VItemMaster.status = 'A'
AND VItemMaster.orderable_ind = 'Y'

```

```

AND NVL(VItemMaster.deposit_item_type, 'X') != 'A'
AND (VItemMaster.item_level = VItemMaster.tran_level
OR (VItemMaster.item_level + 1 = VItemMaster.tran_level
AND EXISTS
  (SELECT 'x'
   FROM diff_group_head
   WHERE VItemMaster.diff_1 = diff_group_id
   OR VItemMaster.diff_2 = diff_group_id
  )))
AND NOT EXISTS
  (SELECT 'x'
   FROM packitem,
   pack_tmpl_head
   WHERE pack_tmpl_head.fash_prepack_ind = 'Y'
   AND packitem.pack_tmpl_id = pack_tmpl_head.pack_tmpl_id
   AND packitem.pack_no = VItemMaster.item
  )
AND VItemMaster.item= NVL(:ParamItem,VItemMaster.item)

```

- **OrderedItemsVO**

This is a view object that is populated programmatically at run time with the details of the item the user has selected with an editable field to enter the quantity of the item required.

- **SystemOptionsVO**

This view object is used to fetch the values of the indicators, `supplier_sites_ind`, and `multiple_set_of_books_ind`, from `system_options` table.

```

SELECT SystemOptions.FINANCIAL_AP,
       SystemOptions.MULTIPLE_SET_OF_BOOKS_IND,
       SystemOptions.SUPPLIER_SITES_IND
FROM SYSTEM_OPTIONS SystemOptions

```

User Interface Design

The following figure displays the user interface for the Micro Application:

Figure 3–1 User Interface for the Create Purchase Order Micro Application

The screenshot shows a web-based user interface for creating a purchase order. At the top, there are two required input fields: '* Location ID' and '* Supplier Site', each with a search icon. Below these is a toolbar with 'View', '+', '-', and 'Detach' buttons. A table with columns 'Item', 'Item Desc.', and '* Ordered Qty.' is shown, currently displaying 'No data to display.'. At the bottom are 'Create Order' and 'Cancel' buttons.

Passing Input Parameters to the Application

The Create Purchase Order Micro Application is used to place an order to procure items at specified locations. The application takes Supplier, Location, and Item as parameters. The user can pass these parameters in any combination to the Create Purchase Order Micro Application.

Note: It is assumed that the values passed for the parameters are correct. There is no explicit check in the application to validate the values passed.

The application that calls the Micro Application can pass values for any of the following parameters:

- location and locationType – Both values need to be passed together. *locationType* takes S or W as input values and *location* takes the Location ID as the input value.
- supplier – takes Supplier ID as the input value.
- item – takes Item ID as the input value.

The application must use the exact names, as specified above, to pass the parameter values. None of the parameters are mandatory. If no values are passed, application works like a stand-alone application. Limitation of the functionality is that only one item can be passed as parameter to the application, even though an order can contain multiple items.

Validations

The following validations are implemented in the application:

- Entering values for location, supplier and item fields is mandatory.
- Both location and supplier have to belong to the same organization.
- Only stock holding and open stores/warehouses can be selected for location.
- If supplier sites are used by the retailer, only supplier sites can be selected on the page. Otherwise, only suppliers can be selected.
- Only valid items that belong to the selected supplier can be added to the order.
- Order quantity of an item has to be a positive integer value.
- All validations imposed by RMS will be executed on the RMS side.

Calling the RMS API

Once the location, supplier, and item details entered by the user is captured, the application uses it to build the required type of the RIB object (RIB_XOrderDesc_REC) and call the RMS API passing the RIB object as the parameter when the Create Order button is pressed.

Signature of the API used to create the order is as follows:

```
RMSSUB_XORDER.CONSUME(O_STATUS_CODE,  
                      O_ERROR_MESSAGE,  
                      L_RIB_XORDERDESC_REC,  
                      I_MESSAGE_TYPE);
```

Where,

- O_STATUS_CODE is an IN OUT parameter which holds the status of the transaction.
 - O_STATUS_CODE = 'S', indicates that the transaction was successful.
 - O_STATUS_CODE = 'E', indicates that an error had occurred during the transaction.
- O_ERROR_MESSAGE is an IN OUT parameter that holds the error message.
- L_RIB_XORDERDESC_REC is an IN parameter that holds that RIB Object for purchase order.
- I_MESSAGE_TYPE is the message type. To create a purchase order, message type to be passed is 'xordercre' (Message sets the status of the order to *worksheet*).
- Display the status of the transaction after the completion of the operation.

Using the Application

To use the Create Purchase Order Micro Application:

1. In the **Location ID** field, select the location number from the List of Values (LOV).
2. In the **Supplier Site** field, select the supplier from LOV.
3. Select the items to be ordered by clicking on the **Plus** sign, selecting the item needed, and then click **OK**.
4. Enter the required quantity of item selected.
5. Review the order details and then click the **Create Order** button.

Application calls the RMS API and executes the transaction. Status of the transaction is printed at the bottom of the page.

In case you want to delete the items you added before clicking the **Create Order** button, you can select the item and then click the **Cross Mark** button.

In case you want to cancel the operation before clicking the **Create Order** button, click the **Cancel** button. It resets the page without making any changes to the selected order.

You can select the supplier first, then select a location and add the items. You can also select the supplier, enter the items, and then select a location.

Limitations

The following limitations apply to this Micro Application:

- Order can be created for only one location.
- No check for minimum orderable quantity of a supplier.
- Order can only be created in worksheet status.
- Only the parameters present on the page are passed to the RMS API. RMS application has to be used to add more parameter values to the order.
- The application supports limited functionality that can be achieved using the parameters on the user interface.
- The application does not check the privileges of the user to create orders.
- Security is not implemented in the application.

Additional Customizations

You may need to implement any additional customization that is not handled through the RMS API, but required by the business process.

Cancel Purchase Order

Cancelling items in a purchase order requires the users to log on to the RMS application, navigate to the Edit Order screen, select the order number, navigate to the Options menu, click Cancel All Items option, confirm the operation, select the reason for cancellation, and then exit from the screen. The Cancel Purchase Order (PO) Micro Application is a simple and efficient application that simplifies this process. It interacts with the RMS database to fetch the required data and commit the changes made by the user.

This Micro Application enables users to cancel an order that already exists in the Approved status. Users just need to launch the Micro Application from the Workspace, enter the order number to be cancelled, and click the Cancel button.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical design](#)
- [Using the Application](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Cancel Purchase Order Micro Application enables the users to cancel a purchase order without accessing the RMS application. It communicates synchronously with RMS through the external APIs exposed by the RMS application. It sends the order cancellation information through the API. The API in RMS does validations and then updates the order with the Cancelled (C) status, in the RMS database.

Note: API does not perform all the validations or operations as they were performed when item is created using the RMS application. You may need to make the necessary changes to meet your business requirements.

This solution does not modify any code in RMS. It uses the existing APIs to perform the operations. The Cancel Purchase Order Micro Application takes the order number entered by the user and passes them as arguments to the external API with the Cancelled (C) status.

Note: The external APIs provide a limited functionality.

The RMS application then processes the message sent by the application and inserts the item in the RMS database tables. The processing done on the RMS side is not in scope of this solution. You may need to modify the RMS processes to achieve the desired results.

Technical design

This section highlights the technical design of the Cancel Purchase Order Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

- [Overview](#)
- [Generating Java Objects for RIB types](#)
- [Creating Business Components](#)
- [User Interface Design](#)
- [Passing Input Parameters to the Application](#)
- [Validations](#)
- [Calling the RMS API](#)

Overview

The Cancel Purchase Order Micro Application provides an user interface where the order number to be cancelled are captured. The application then calls the RMS APIs to cancel the selected order. However, the RMS APIs require the RIB objects as input parameters. Based on the RIB-Type Objects, Application builds the corresponding Java objects and populates the objects, with the information gathered from the user interface and RMS database. A JDBC call is then made to the RMS API, passing the Java object as a parameter. The RMS API is then executed on the RMS database schema, which updates the corresponding tables in the database.

Generating Java Objects for RIB types

To cancel a purchase order, the *RIB_XOrderDesc_REC* RIB object is passed to the RMS X API. This object contains a collection of *RIB_XOrderDtl_REC* objects. The application uses the *RIB_XOrderDesc_REC* RIB object to create the Java objects. This automatically generates the relevant Java objects for the *RIB_XOrderDtl_REC* object.

Structure of the *RIB_XOrderDesc_REC* object:

```
order_no varchar2(10),
supplier varchar2(10),
currency_code varchar2(3),
terms varchar2(15),
not_before_date date,
not_after_date date,
otb_eow_date date,
dept number(4),
status varchar2(1),
exchange_rate number(20,10),
include_on_ord_ind varchar2(1),
written_date date,
```

```
XOrderDtl_TBL "RIB_XOrderDtl_TBL",
orig_ind varchar2(1),
edi_po_ind varchar2(1),
pre_mark_ind varchar2(1),
user_id varchar2(30),
comment_desc varchar2(2000),
ExtOfXOrderDesc_TBL "RIB_ExtOfXOrderDesc_TBL"
```

Structure of RIB_XOrderDtl_REC:

```
item varchar2(25),
location number(10),
unit_cost number(20,4),
ref_item varchar2(25),
origin_country_id varchar2(3),
supp_pack_size number(12,4),
qty_ordered number(12,4),
location_type varchar2(1),
cancel_ind varchar2(1),
reinstate_ind varchar2(1),
ExtOfXOrderDtl_TBL "RIB_ExtOfXOrderDtl_TBL"
```

Creating Business Components

The following SQL-based business components have been used to create the Micro Application:

- **SelectOrderVO**

To populate the LOV for orders to be cancelled:

```
Select oh.order_no, oh.supplier,s.sup_name , concat(concat(oh.order_
no,'-'),concat( concat(oh.supplier, '-'), s.sup_name)) AS ORDER_SEARCH from
ordhead oh,sups s where status = 'A' and oh.supplier=s.supplier order by 1
```

- **OrdLocVO**

To populate the details of the selected order:

```
SELECT DISTINCT DECODE(ol.loc_type,'S','Store','W','Warehouse') AS location_
Type,
oh.supplier,
oh.supplier || ' - ' || s.sup_name AS supplier_name,
oh.dept ,
oh.terms ,
oh.terms || ' - ' ||th.terms_desc AS terms_desc,
oh.status ,
DECODE(oh.status,'A','Approved','W','Worksheet') AS status_desc,
ol.order_no ,
ol.item ,
im.item_desc ,
ol.qty_ordered ,
ol.location ,
DECODE(ol.loc_type,'S',store.store_name,'W',wh.wh_name) AS location_name ,
ol.loc_type
FROM store store,
wh wh ,
ordloc ol ,
item_master im,
ordhead oh,
sups s ,
```

```

terms_head th
WHERE ol.order_no=:order_no
and ol.order_no = oh.order_no
AND ol.item =im.item
AND s.supplier = oh.supplier
AND oh.terms =th.terms
AND (store.store=ol.location
OR wh.wh =ol.location)
ORDER BY ol.item
    
```

User Interface Design

The following figure displays the user interface for the Micro Application:

Figure 4–1 User Interface for the Cancel Purchase Order Micro Application

Item	Item Desc	Location Name	Ordered Qty.
100018128	Ba...	Store Su...	37
100018136	Ba...	Store Su...	19
100018144	Ba...	Store Su...	37
100018152	Ba...	Store Su...	56
100018161	Ba...	Store Su...	29
100019041	Ba...	Store Su...	73
100019067	Ba...	Store Su...	116

Passing Input Parameters to the Application

The Cancel Purchase Order Micro Application is used to cancel an order. The application takes the Order Number as the parameter.

Note: It is assumed that the values passed for the parameters are correct. There is no explicit check in the application to validate the values passed.

The application that calls the Micro Application can pass values for any of the following parameters:

- orderNo – takes Order ID as the input value.

The application must use the exact name, as specified above, to pass the parameter value. Order Number parameter is not mandatory. If no values are passed, application works like a stand-alone application.

If the order number is passed to the application, the application page appears with the details of the order number. Users can review the order details, and then click the Cancel Order button to cancel the order.

Validations

The following validations are implemented in the application:

- Only orders in Approved status can be cancelled.
- Selecting an order number is mandatory.
- It is not possible to enter an invalid order number and execute the operation.
- All validations imposed by RMS will be executed on the RMS side.

Calling the RMS API

Once the order number entered by the user is captured, the application uses it to build the required type of the RIB object (RIB_XOrderDesc_REC) and call the RMS API passing the RIB object as the parameter when the Cancel Order button is pressed.

Signature of the API used to create the order is as follows:

```
RMSSUB_XORDER.CONSUME (O_STATUS_CODE,
                       O_ERROR_MESSAGE,
                       L_RIB_XORDERDESC_REC,
                       I_MESSAGE_TYPE);
```

Where,

- O_STATUS_CODE is an IN OUT parameter which holds the status of the transaction.
 - O_STATUS_CODE = 'S', indicates that the transaction was successful.
 - O_STATUS_CODE = 'E', indicates that an error had occurred during the transaction.
- O_ERROR_MESSAGE is an IN OUT parameter that holds the error message.
- L_RIB_XORDERDESC_REC is an IN parameter that holds that RIB Object for purchase order.
- I_MESSAGE_TYPE is the message type. To cancel a purchase order, message type to be passed is '**xordermod**' (Message sets the status of the order to *cancelled/Closed*).
- Display the status of the transaction after the completion of the operation.

Using the Application

To use the Cancel Purchase Order Micro Application:

1. In the Order No field, select the order number from the List of Values (LOV).
All other details on the screen appear when the order is selected.
2. Review the order details, and then click the Cancel Order button.
3. Application calls the RMS API and executes the transaction. Status of the transaction is printed at the bottom of the page.

In case you want to cancel the operation before clicking the Cancel Order button, click the Cancel button. It resets the page without making any changes to the selected order.

Limitations

The following limitations apply to this Micro Application:

- The application does not check the privileges of the user to cancel orders.
- Security is not implemented in the application.
- The application only sends an order modify message to RMS with Cancelled status. Processing of the message in RMS is not in the scope of this application.

Additional Customizations

To replicate the Cancel Order functionality in RMS, some additional customizations are recommended for the Micro Application.

Allocations, shipments, and deals associated with the order being cancelled are not handled when the Cancel Purchase Order Micro Application is used. It is recommended that you call the relevant package that handles all these transactions in the RMS API.

You may need to implement any additional customization that is not handled through the RMS API, but required by the business process.

Item Transfer

Transferring items in the RMS application requires the users to fill many fields. The Item Transfer Micro Application is a simple and efficient application that simplifies this process. It interacts with the RMS database to fetch the required data and commit the changes made by the user.

Users just need to launch the Micro Application from the Workspace, select the source location, destination location, items to be transferred, transfer quantity, and then click the Transfer button.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical design](#)
- [Using the Application](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Item Transfer Micro Application enables the users to create an item transfer without accessing the RMS application. It communicates synchronously with RMS through the external APIs exposed by the RMS application. It sends the item transfer information through the API. The API in RMS does validations and then transfers the items.

Note: API does not perform all the validations or operations as they were performed when item is created using the RMS application. You may need to make the necessary changes to meet your business requirements.

This solution does not modify any code in RMS. It uses the existing APIs to perform the operations. The Item Transfer Micro Application takes the location and item details for the transfer entered by the user and passes them as arguments to the external API with the Approved (A) status. Items can be transferred from a store to warehouse, warehouse to store, and between two stores or two warehouses.

Note: The external APIs provide a limited functionality.

The RMS application then processes the message sent by the application and inserts the new transfer in the RMS database tables. The processing done on the RMS side is not in scope of this solution. You may need to modify the RMS processes to achieve the desired results.

Technical design

This section highlights the technical design of the Item Transfer Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

- [Overview](#)
- [Generating Java Objects for RIB types](#)
- [Creating Business Components](#)
- [User Interface Design](#)
- [Passing Input Parameters to the Application](#)
- [Validations](#)
- [Calling the RMS API](#)

Overview

The Item Transfer Micro Application provides an user interface where the location and item details are captured. The application then calls the RMS APIs to transfer the selected items with the relevant transfer quantities. However, the RMS APIs require the RIB objects as input parameters. Based on the RIB-Type Objects, Application builds the corresponding Java objects and populates the objects, with the information gathered from the user interface and RMS database. A JDBC call is then made to the RMS API, passing the Java object as a parameter. The RMS API is then executed on the RMS database schema, which updates the corresponding tables in the database.

Generating Java Objects for RIB types

To transfer an item, the *RIB_XTsFDesc_REC* RIB object is passed to the RMS X API. This object contains a collection of *RIB_XTsFDtl_REC* objects. The application uses the *RIB_XTsFDesc_REC* RIB object to create the Java objects. This automatically generates the relevant Java objects for the *RIB_XTsFDtl_REC* object.

Structure of the *RIB_XTsFDesc_REC* object:

```
tsf_no number(10),
from_loc_type varchar2(1),
from_loc varchar2(10),
to_loc_type varchar2(1),
to_loc varchar2(10),
delivery_date date,
dept number(4),
routing_code varchar2(1),
freight_code varchar2(1),
tsf_type varchar2(6),
XTsFDtl_TBL "RIB_XTsFDtl_TBL",
status varchar2(1),
user_id varchar2(30),
comment_desc varchar2(2000),
ExtOfXTsFDesc_TBL "RIB_ExtOfXTsFDesc_TBL"
```

Structure of the *RIB_XTsfdtl_REC* object:

```
item varchar2(25),
tsf_qty number(12,4),
supp_pack_size number(12,4),
inv_status number(2),
unit_cost number(20,4),
ExtOfXTsfdtl_TBL "RIB_ExtOfXTsfdtl_TBL"
```

Creating Business Components

The following SQL-based business components have been used to create the Micro Application:

■ SrcLocationVO

To populate the LOV for selecting the source location for the transfer.

```
SELECT distinct s1.store AS Location,
s1.store_name AS Loc_name,
'S' AS Loc_type,
concat(concat(s1.store, '-'), s1.store_name) AS LOCATION_SEARCH
FROM store s1, store s2, item_loc_soh ils
WHERE s1.stockholding_ind = 'Y'
AND DECODE(ils.loc_type, 'S', ils.loc, s1.store) = DECODE(ils.loc_
type, 'S', s1.store, ils.loc)
AND ils.item = NVL(:ParamItem, ils.item)
AND ils.stock_on_hand > 0
AND (( NVL(:to_loc_type, 'S') = 'S'
AND s2.store = NVL(:to_loc, s2.store)
AND s1.transfer_zone = NVL(s2.transfer_zone, s1.transfer_zone)
AND s1.store <> NVL(:to_loc, -1))
OR NVL(:to_loc_type, 'W') = 'W')

UNION

SELECT wh AS Location,
wh_name AS Loc_name,
'W' AS Loc_type,
concat(concat(wh, '-'), wh_name) AS LOCATION_SEARCH
FROM wh, item_loc_soh ils
WHERE stockholding_ind = 'Y'
AND DECODE(ils.loc_type, 'W', ils.loc, wh.wh) = DECODE(ils.loc_
type, 'W', wh.wh, ils.loc)
AND ils.item = NVL(:ParamItem, ils.item)
AND ils.stock_on_hand > 0
AND wh <> NVL(:to_loc, -1)
```

■ DestLocationVO

To populate the LOV for selecting the destination location for the transfer:

```
SELECT distinct s1.store AS Location,
s1.store_name AS Loc_name,
'S' AS Loc_type,
concat(concat(s1.store, '-'), s1.store_name) AS LOCATION_SEARCH
FROM store s1, store s2
WHERE s1.stockholding_ind = 'Y'
AND (( NVL(:from_loc_type, 'S') = 'S'
AND s2.store = NVL(:from_loc, s2.store)
AND s1.transfer_zone = NVL(s2.transfer_zone, s1.transfer_zone)
```

```

AND sl.store <> NVL(:from_loc,-1)
OR NVL(:from_loc_type,'W') = 'W'

UNION

SELECT wh AS Location,
       wh_name AS Loc_name,
       'W' AS Loc_type,
       concat(concat(wh, '- '),wh_name) AS LOCATION_SEARCH
FROM wh
WHERE stockholding_ind= 'Y'
AND wh <> NVL(:from_loc,-1)

```

- **DepartmentVO**

To populate the LOV for selecting the department for the transfer:

```
SELECT dept,dept_name FROM depts
```

- **ItemVO**

To populate the pop up for selecting the items for the transfer:

```

SELECT distinct im.item, im.item_desc, ils.stock_on_hand
FROM item_loc il, item_loc_soh ils, item_master im
WHERE im.dept = NVL(:dept,im.dept)
AND il.item = im.item
AND ils.item = im.item
AND il.loc = NVL(:from_loc,il.loc)
AND il.loc_type = NVL(:from_loc_type,il.loc_type)
AND ils.loc = NVL(:from_loc,ils.loc)
AND ils.loc_type = NVL(:from_loc_type,ils.loc_type)
AND im.status = 'A'
AND im.item_level = im.tran_level
AND im.inventory_ind = 'Y'
AND ils.stock_on_hand > 0
AND ((im.pack_ind = 'Y'
AND NVL(:from_loc_type,'W') = 'W'
AND NVL(il.receive_as_type,'P') = 'P')
OR im.pack_ind = 'N')
ORDER BY 1

```

- **SystemOptionsVO**

For checking the indicator of dept_level_transfers to enable the Department LOV.

```
SELECT SystemOptions.dept_level_transfers FROM SYSTEM_OPTIONS SystemOptions
```

- **TransferItemsVO**

This view object is populated with rows programmatically, not based on a query, from the ItemVO.

User Interface Design

The following figure displays the user interface for the Micro Application:

Figure 5–1 User Interface for the Item Transfer Micro Application

Item	Item Desc.	Available Qty.	Transfer Qty.
100005010	Gwen Taffeta Drt 55		10
100005108	Cashmere Turtler 77		15
100005052	Guinevere Silk Drt 44		40

Passing Input Parameters to the Application

The Item Transfer Micro Application is used to create a transfer of items between two specified locations. The application takes Source Location, Destination Location, and Item as parameters. The user can pass these parameters in any combination to the Item Transfer Micro Application.

Note: It is assumed that the values passed for the parameters are correct. There is no explicit check in the application to validate the values passed.

The application that calls the Micro Application can pass values for any of the following parameters:

- `sourceLocationType` and `sourceLocation` – Both values need to be passed together. `sourceLocationType` takes *S* or *W* as input values and `sourceLocation` takes the Location ID as the input value.
- `destinationLocationType` and `destinationLocation` – Both values need to be passed together. `destinationLocationType` takes *S* or *W* as input values and `destinationLocation` takes the Location ID as the input value.
- `Item` – takes Item ID as the input value.

The application must use the exact names, as specified above, to pass the parameter values. None of the parameters are mandatory. To pass a value for source location, the calling application has to assign values for both `sourceLocationType` and

sourceLocation variables. The same process must be followed for the destination location. If no values are passed, application works like a stand-alone application. Limitation of the functionality is that only one item can be passed as parameter to the application, even though a transfer can contain multiple items.

Validations

The following validations are implemented in the application:

- Values for source location, destination location, item, and transfer quantity are mandatory
- Transfer quantity is less than or equal to the available quantity.
- Only a positive integer value can be entered for transfer quantity.
- Source and destination locations should lie in the same transfer zone, if the locations are stores.
- Transferred item should have stock on hand greater than zero at a location for that location to be selected as source for the transfer.
- Only valid items that belong to the selected source location can be added to the transfer.
- All validations imposed by RMS will be executed on the RMS side.

Calling the RMS API

Once the transfer number generated by the NEXT_TRANSFER_NUMBER procedure, the application uses it to build the required type of the RIB object (RIB_XTsfDesc_REC) and call the RMS API passing the RIB object as the parameter when the Create Order button is pressed.

Signature of the API used to create the order is as follows:

```
RMSSUB_XTSF.CONSUME(O_STATUS_CODE,  
                   O_ERROR_MESSAGE,  
                   L_RIB_XTSFDESC_REC,  
                   I_MESSAGE_TYPE);
```

Where,

- O_STATUS_CODE is an IN OUT parameter which holds the status of the transaction.
 - O_STATUS_CODE = 'S', indicates that the transaction was successful.
 - O_STATUS_CODE = 'E', indicates that an error had occurred during the transaction.
- O_ERROR_MESSAGE is an IN OUT parameter that holds the error message.
- L_RIB_XTSFDESC_REC is an IN parameter that holds that RIB Object for item transfer.
- I_MESSAGE_TYPE is the message type. To create a transfer, message type to be passed is 'xtsfcre'.
- Display the status of the transaction after the completion of the operation.

Using the Application

To use the Item Transfer Micro Application:

1. In the **Source Location** field, select the source location from the List of Values (LOV).
2. In the **Destination Location** field, select the destination location from the LOV.
3. Click the **Plus** sign to select the items you want to transfer.
All other item details automatically appear on the screen when items are selected.
4. In the **Transfer Qty.** field, enter the transfer quantity for each of the items selected.
5. Review the transfer details, and then click the **Transfer** button.
6. Application calls the RMS API and executes the transaction. Status of the transaction is printed at the bottom of the page.

To delete the items from the table, select the items, and click the Cross Mark button.

In case you want to cancel the operation before clicking the Transfer button, click the Cancel button. It resets the page.

Limitations

The following limitations apply to this Micro Application:

- Only simple case of item transfer is handled.
- Only stores or warehouses can be entered for source and destination locations.
- Use of external finishers or internal finishers is not supported.
- The application supports limited functionality that can be achieved using the parameters on the user interface.
- The application does not check the privileges of the user to create transfer.
- Security is not implemented in the application.

Additional Customizations

You may need to implement any additional customization that is not handled through the RMS API, but required by the business process.

Update Vendor

The Update Vendor Micro Application is used to access the supplier contact information and modify them easily. The Micro Application takes the supplier number as input and displays the contact information of that supplier. The supplier information is fetched from the RMS database. Once the supplier contact information on the page are modified and submitted, values are committed to the corresponding tables in the RMS database.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical design](#)
- [Using the Application](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Update Vendor Micro Application enables the users to update the contact information of a supplier without accessing the RMS application. It communicates synchronously with RMS through the external APIs exposed by the RMS application. It sends the updated supplier information through the API. The API in RMS does validations and then updates the supplier database tables with the new values in the RMS database.

Note: API does not perform all the validations or operations as they were performed when item is created using the RMS application. You may need to make the necessary changes to meet your business requirements.

This solution does not modify any code in RMS. It uses the existing APIs to perform the operations. The Update Vendor Micro Application takes the order number entered by the user and passes them as arguments to the external API.

Note: The external APIs provide a limited functionality.

The RMS application then processes the message sent by the application and updates the supplier information in the RMS database tables. The processing done on the RMS

side is not in scope of this solution. You may need to modify the RMS processes to achieve the desired results.

Technical design

This section highlights the technical design of the Create Purchase Order Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

- [Overview](#)
- [Generating Java Objects for RIB Types](#)
- [Creating Business Components](#)
- [User Interface Design](#)
- [Passing Input Parameters to the Application](#)
- [Validations](#)
- [Calling the RMS API](#)

Overview

The Update Vendor Micro Application provides an user interface where the vendor number is captured to update the contact information. The application then calls the RMS APIs to update the contact information of the selected vendor. However, the RMS APIs require the RIB objects as input parameters. Based on the RIB-Type Objects, the application builds the corresponding Java objects and populates the objects, with the information gathered from user interface and RMS database. A JDBC call is then made to the RMS API, passing the Java object as a parameter. The RMS API is then executed on the RMS database schema, which updates the corresponding tables in the database.

Generating Java Objects for RIB Types

To update the vendor information, the *RIB_SupplierColDesc_REC* RIB object is passed to the RMS X API. The Java object for *RIB_SupplierColRef_REC* object contains a collection of *RIB_SupplierDesc_REC* object. The *RIB_SupplierDesc_REC* object further contains the collection of *RIB_SupAttr_REC* object.

The application uses the *RIB_SupplierColRef_REC* RIB object to create the Java objects for this RIB object and all child objects. This automatically generate the relevant Java objects for the *RIB_SupplierDesc_REC* RIB-Type object and *RIB_SupAttr_REC*.

Structure of *RIB_SupplierColDesc_REC* RIB Object:

```
SupplierDesc_TBL "RIB_SupplierDesc_TBL ",  
ExtOfSupplierColDesc_TBL " RIB_ExtOfSupplierColDesc_TBL "
```

Structure of *RIB_SupplierDesc_REC* RIB Object:

```
supplier_id number(10),  
SupAttr "RIB_SupAttr_REC",  
SupSite_TBL "RIB_SupSite_TBL",  
ExtOfSupplierDesc_TBL "RIB_ExtOfSupplierDesc_TBL"
```

Structure of *RIB_SupAttr_REC* RIB Object:

```
sup_name varchar2(240),  
sup_name_secondary varchar2(240),  
contact_name varchar2(120),
```

```
contact_phone varchar2(20),
contact_fax varchar2(20),
contact_pager varchar2(20),
sup_status varchar2(1),
qc_ind varchar2(1),
qc_pct number(12,4),
qc_freq varchar2(2),
vc_ind varchar2(1),
vc_pct number(12,4),
vc_freq number(2),
currency_code varchar2(3),
lang number(6),
terms varchar2(15),
freight_terms varchar2(30),
ret_allow_ind varchar2(1),
ret_auth_req varchar2(1),
ret_min_dol_amt number(20,4),
ret_courier varchar2(250),
handling_pct number(12,4),
edi_po_ind varchar2(1),
edi_po_chg varchar2(1),
edi_po_confirm varchar2(1),
edi_asn varchar2(1),
edi_sales_rpt_freq varchar2(1),
edi_supp_available_ind varchar2(1),
edi_contract_ind varchar2(1),
edi_invc_ind varchar2(1),
edi_channel_ind number(4),
cost_chg_pct_var number(12,4),
cost_chg_amt_var number(20,4),
replen_approval_ind varchar2(1),
ship_method varchar2(6),
payment_method varchar2(6),
contact_telex varchar2(20),
contact_email varchar2(100),
settlement_code varchar2(1),
pre_mark_ind varchar2(1),
auto_appr_invc_ind varchar2(1),
dbt_memo_code varchar2(1),
freight_charge_ind varchar2(1),
auto_appr_dbt_memo_ind varchar2(1),
prepay_invc_ind varchar2(1),
backorder_ind varchar2(1),
vat_region number(4),
inv_mgmt_lvl varchar2(6),
service_perf_req_ind varchar2(1),
invc_pay_loc varchar2(6),
invc_receive_loc varchar2(6),
addinvc_gross_net varchar2(6),
delivery_policy varchar2(6),
comment_desc varchar2(2000),
default_item_lead_time number(4),
duns_number varchar2(9),
duns_loc varchar2(4),
bracket_costing_ind varchar2(1),
vmi_order_status varchar2(6),
dsd_ind varchar2(1),
scale_aip_orders varchar2(1),
sup_qty_level varchar2(6),
ExtOfSupAttr_TBL "RIB_ExtOfSupAttr_TBL"
```

Creating Business Components

The following SQL-based business components have been used to create the Micro Application:

- **SupplierVO**

To populate the LOV (List of values) object for list of suppliers available:

```
select supplier, sup_name ,concat( concat(supplier, '-'), sup_name) AS
SUPPLIER_SEARCH from sups
```

- **SupplierDetailsVO**

To populate the contact details of the selected supplier:

```
Select SUPPLIER,
SUP_NAME,
CONTACT_NAME,
CONTACT_PHONE,
CONTACT_EMAIL,
COMMENT_DESC,
TERMS,
freight_terms,
currency_code
From SUPS
Where supplier=:supplierID
```

User Interface Design

The following figure displays the user interface for the Micro Application:

Figure 6–1 User Interface for the Update Vendor Micro Application



* Supplier : 2300

Supplier Name : Coca Cola

* Contact Name : John Mcnero

* Contact Phone : 001-2323-9999

Contact Email : john.mcnero@cocacola.com

Comments : Alternatively, pls use johnmc@gmail.com

Update Cancel

Supplier : 2300 has been updated successfully.

Passing Input Parameters to the Application

The Update Vendor Micro Application is used to modify the contact details of an vendor. The application takes the Vendor Number as the parameter.

Note: It is assumed that the values passed for the parameters are correct. There is no explicit check in the application to validate the values passed.

The application that calls the Micro Application can pass values for any of the following parameters:

- vendorId – takes Vendor ID as the input value.

The application must use the exact name, as specified above, to pass the parameter value. Vendor Number parameter is not mandatory. If no values are passed, application works like a stand-alone application.

If the vendor number is passed to the application, the application page appears with the details of the vendor. Users can then modify details and click the Update button to commit the changes.

Validations

The following validations are implemented in the application:

- Selecting Supplier is mandatory.
- It is not possible to enter an invalid Supplier number and execute the operation.
- Contact Name and Contact Phone are mandatory and these cannot be empty or contain only spaces.
- All validations imposed by RMS will be executed on the RMS side.

Calling the RMS API

Once the supplier number selected or entered by the user is captured, the application uses it to build the required type of the RIB object (RIB_SupplierColRef_REC) and call the RMS API passing the RIB object as the parameter when the Update button is pressed.

Signature of the API used to create the order is as follows:

```
RMSAIASUB_SUPPLIER.CONSUME(O_STATUS_CODE,
                             O_ERROR_MESSAGE,
                             O_OUTPUTOBJECT,
                             I_INPUTOBJECT,
                             I_INPUTOBJECT_TYPE);
```

Where,

- O_STATUS_CODE is an IN OUT parameter which holds the status of the transaction.
 - O_STATUS_CODE = 'S', indicates that the transaction was successful.
 - O_STATUS_CODE = 'E', indicates that an error had occurred during the transaction.
- O_ERROR_MESSAGE is an IN OUT parameter that holds the error message.
- O_OUTPUTOBJECT is an IN OUT parameter that holds the RIB Object of type RIB_SupplierColRef_REC for additional information to be passed.
- I_INPUTOBJECT is an IN parameter that holds that RIB Object RIB_SupplierColDesc_REC containing selected vendor details.
- I_INPUTOBJECT_TYPE is an IN parameter. It is meant to carry a message type to be passed to the RIB object to indicate the operation. To Update the Vendor information, message type has to be passed as 'suppmo' (Message modifies the vendor information).

The application uses the `O_STATUS_CODE` returned value to display the status of the transaction after the completion of the operation.

Using the Application

To use the Update Vendor Micro Application:

1. In the **Supplier** field, select the supplier from the List of Values (LOV). Alternatively, you can enter the supplier number in the **Supplier** field.
All other details of the supplier appear on the screen automatically.
2. Review the supplier contact details and then click the **Update** button.
3. The application calls the RMS API and executes the transaction. Status of the transaction is printed at the bottom of the page.

In case you want to cancel the operation before clicking Update button, click the Cancel button. It resets the page without making any changes to the selected Supplier.

Limitations

The following limitations apply to this Micro Application:

- The modified contact details will only be committed to the RMS database and they do not flow to any other applications that hold supplier information.
- The application does not check the privileges of the user to update the supplier details.
- Security is not implemented in the application.
- The application only sends the supplier modify message to RMS with the relevant changed contact details. Processing of the message in RMS is not in the scope of this application.

Additional Customizations

You may need to implement any additional customization that is not handled through the RMS API, but required by the business process.

Bulk Cancellation/Approval of Purchase Orders

Cancelling items in a purchase order requires the users to log on to the RMS application, navigate to the Edit Order screen, select the order number, navigate to the Options menu, click Cancel All Items option, confirm the operation, select the reason for cancellation, and then exit from the screen. Similarly, approving items in a purchase order requires the users to log on to the RMS application, navigate to the Edit Order screen, select the order number, navigate to the Options menu, and click Submit. Once submitted, click the Approve option from the Options menu, confirm the operation, and then exit from the screen. This is a time consuming process and the RMS application does not provide an option to cancel or approve multiple orders at the same time.

The Bulk Approval/Cancellation of Orders Micro Application is a simple and efficient application that simplifies the process of cancelling or approving an order and provides the users with an option to approve or cancel multiple orders at the same time. It interacts with the RMS database to fetch the required data and commit the changes made by the user. This Micro Application is designed to cancel or approve multiple orders that are eligible for cancellation or approval. Users can launch the Micro Application from the relevant workspace, select the relevant action (approval or cancellation), select the orders that need to be cancelled or approved, and then click the Cancel or Approve button on the page.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical Design](#)
- [Application Usage](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Bulk Approval/Cancellation of Orders Micro Application enables users to approve or cancel multiple purchase orders at the same time without logging on to the RMS application. It communicates synchronously with RMS through the external APIs exposed by the RMS application.

It sends the order approval or cancellation information through the API. The RMS API performs the validations and then updates the order with the relevant status based on the action taken (approval or cancellation) in the RMS database.

Note: The API does not perform all the validations/operations as they were performed when the approval or cancellation is done using the RMS application. Ensure that you make the necessary changes to the solution to meet your business requirements.

This solution does not modify any code in RMS. It uses the existing APIs to perform the operations. The Micro Application takes the action (approval/cancellation) and the order numbers selected by the user, and passes it as argument to the external API with Cancelled (C) or Approved (A) status based on the action selected.

Note: The external APIs provide a limited functionality.

The RMS application then processes the message sent by the application and updates the order status in the RMS database tables. The processing done on the RMS side is not in scope of this solution. You may need to modify the RMS processes to achieve the desired results.

Technical Design

This section highlights the technical design of the Bulk Approval/Cancellation Purchase Order Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

- [Overview](#)
- [Generating Java Objects for RIB Types](#)
- [Creating Business Components](#)
- [User Interface Design](#)
- [Validations](#)
- [Calling the RMS API](#)

Overview

The Bulk Approval/Cancellation of Purchase Order Micro Application provides a user interface where the user's choice of action (Approve or Cancel) is captured first. Based on the action, the table gets populated with the relevant orders. The application then calls the RMS API to approve or cancel the selected orders. However, the RMS APIs require the RIB objects as input parameters. Based on the RIB-Type Objects, Application builds the corresponding Java objects and populates the objects, with the information gathered from the user interface and RMS database. A JDBC call is then made to the RMS API, passing the Java object as a parameter. The RMS API is then

executed on the RMS database schema, which updates the corresponding tables in the database.

Generating Java Objects for RIB Types

To approve or cancel a purchase order, the RIB_XOrderDesc_REC RIB object is passed to the RMS X API. This object contains a collection of RIB_XOrderDtl_REC objects.

The application uses the RIB_XOrderDesc_REC RIB object to create the Java objects. This automatically generates the relevant Java objects for the RIB_XOrderDtl_REC object.

Structure of RIB_XOrderDesc_REC:

```
order_no varchar2(10),
supplier varchar2(10),
  currency_code varchar2(3),
  terms varchar2(15),
  not_before_date date,
  not_after_date date,
  oth_eow_date date,
  dept number(4),
  status varchar2(1),
  exchange_rate number(20,10),
  include_on_ord_ind varchar2(1),
  written_date date,
  XOrderDtl_TBL "RIB_XOrderDtl_TBL",
  orig_ind varchar2(1),
  edi_po_ind varchar2(1),
  pre_mark_ind varchar2(1),
  user_id varchar2(30),
  comment_desc varchar2(2000),
  ExtOfXOrderDesc_TBL "RIB_ExtOfXOrderDesc_TBL"
```

Structure of RIB_XOrderDtl_REC:

```
item varchar2(25),
location number(10),
  unit_cost number(20,4),
  ref_item varchar2(25),
  origin_country_id varchar2(3),
  supp_pack_size number(12,4),
  qty_ordered number(12,4),
  location_type varchar2(1),
  cancel_ind varchar2(1),
  reinstate_ind varchar2(1),
  ExtOfXOrderDtl_TBL "RIB_ExtOfXOrderDtl_TBL"
```

Creating Business Components

The following SQL-based business components have been used to create the Micro Application:

- **OrdersVO**

To populate the table with the orders eligible for cancellation or approval:

```
SELECT DISTINCT oh.order_no, oh.supplier,s.sup_name,
concat(concat(oh.order_no,'-'),concat( concat(oh.supplier, '-'), s.sup_name))
AS ORDER_SEARCH
FROM ordhead oh,sups s
WHERE status = :statusID and oh.supplier=s.supplier
ORDER BY 1
```

- **OrderDetailsVO**

To populate the details of the selected order:

```
SELECT DISTINCT DECODE(ol.loc_type,'S','Store','W','Warehouse')
AS location_Type,
oh.supplier || '-' || s.sup_name AS supplier_name,
oh.dept ,
oh.terms ,
oh.terms || '-' || th.terms_desc AS terms_desc,
oh.status ,
DECODE(oh.status,'A','Approved','W','Worksheet') AS status_desc,
ol.order_no ,
ol.item ,
im.item_desc ,
ol.qty_ordered ,
ol.location ,
DECODE(ol.loc_type,'S',store.store_name,'W',wh.wh_name) AS location_name ,
ol.loc_type
FROM store store,
wh wh ,
ordloc ol ,
item_master im,
ordhead oh,
sups s ,
terms_head th
WHERE ol.order_no=:order_no
and ol.order_no = oh.order_no
AND ol.item =im.item
AND s.supplier = oh.supplier
AND oh.terms =th.terms
AND (store.store=ol.location
OR wh.wh =ol.location)
ORDER BY ol.item
```

User Interface Design

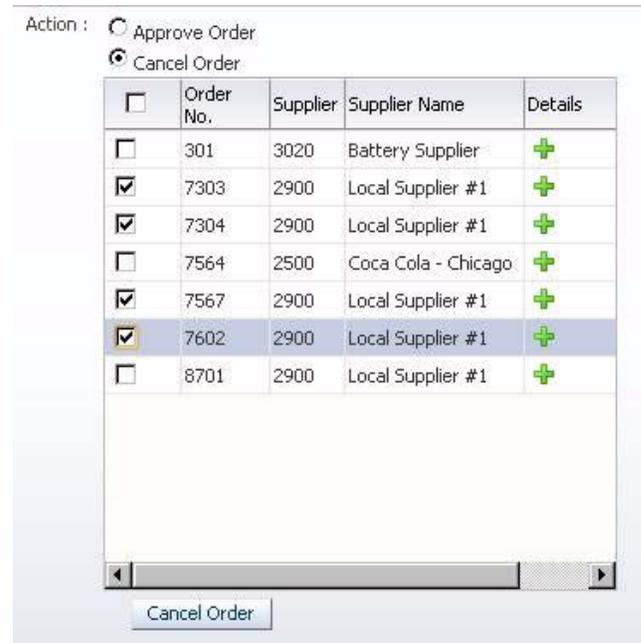
The following figure displays the user interface that appears when users select the Approve Order option:

Figure 7-1 User Interface for the Approve Order Option



The following figure displays the user interface that appears when users select the Cancel Order option:

Figure 7-2 User Interface for the Cancel Order Option



Validations

The following validations are implemented in the application:

- Only the orders in the Approved status can be cancelled.
- Only the orders in the Worksheet status are eligible for the approval operation.
- All validations imposed by RMS will be executed on the RMS side.

Calling the RMS API

Once the order numbers selected by the user is captured, the application uses it to build the required type of the RIB object (RIB_XOrderDesc_REC) and call the RMS API passing the RIB object as the parameter when the Cancel Order or Approve Order button is pressed.

Signature of the API used to create the order is as follows:

```
RMSSUB_XORDER.CONSUME (O_STATUS_CODE,  
                        O_ERROR_MESSAGE,  
                        L_RIB_XORDERDESC_REC,  
                        I_MESSAGE_TYPE) ;
```

Where,

- O_STATUS_CODE is an IN OUT parameter which holds the status of the transaction.
 - O_STATUS_CODE = 'S', indicates that the transaction was successful.
 - O_STATUS_CODE = 'E', indicates that an error had occurred during the transaction.
- O_ERROR_MESSAGE is an IN OUT parameter that holds the error message.
- L_RIB_XORDERDESC_REC is an IN parameter that holds that RIB Object for purchase order.
- I_MESSAGE_TYPE is the message type. To approve or cancel a purchase order, message type to be passed is '**xordermod**' (Message sets the status of the order to *cancelled/approved*).
- Display the status of the transaction after the completion of the operation.

Application Usage

To use the Bulk Approval/Cancellation of Purchase Orders Micro Application:

1. Under Action, select one of the following options:
 - Approve Order
 - Cancel Order

Based on the option you select, the eligible orders appear in the table below.

Figure 7-3 Bulk Approval/Cancellation of Purchase Orders User Interface

2. Review the order details by clicking the Plus sign (+).
3. Select the orders you want to approve or cancel.
4. Based on the option you selected in Step 1, the Approve Order or Cancel Order button appears. Select the Approve Order or Cancel Order button.
5. Application calls the RMS API and executes the transaction. Status of the transaction is printed in a pop up for the selected orders.

Limitations

The following limitations apply to this Micro Application:

- The application does not check the privileges of the user to cancel orders.
- Security is not implemented in the application.
- The application only sends an order modify message to RMS with Cancelled/Approved status. Processing of the message in RMS is not in the scope of this application.

Additional Customizations

To replicate the Modify Order functionality in RMS, some additional customizations are recommended for the Micro Application.

Allocations, shipments, and deals associated with the order being approved or cancelled are not handled when the Micro Application is used. It is recommended that you call the relevant package that handles all these transactions in the RMS API.

You may need to implement any additional customization that is not handled through the RMS API, but required by the business process.

In the Oracle Retail Merchandising System (RMS) application, data is displayed in a plain Oracle Forms-based format. It is represented in the same format without considering the type of data, location the data corresponds to, or how critical the data is. The Maps Micro Application provides users with a rich interface that will help them in managing stores located around the world by a simple click of a button. The Maps Micro Application is designed to provide a sense of business intelligence to the user by highlighting the data based on the location it corresponds to, how crucial it is, and what it represents. This in turn will help users in managing the stores better.

This chapter provides information you need to start using or implementing this Micro Application. It includes the following sections:

- [Functional Design](#)
- [Technical Design](#)
- [Using the Application](#)
- [Limitations](#)
- [Additional Customizations](#)

Functional Design

The Maps Micro Application provides a rich interface to the user and provides ways to highlight the data based on its type, value, and so on. It also provides means to take the right action on the right data at the right location required for the business with a simple click of a button.

The features of this Micro application are based on how you customize the application. You can customize it to include the features that best suit your business need. For example, a sales company may customize the application to manage the sales of items around the world, whereas another company that is in charge of monitoring the inventory may be interested in managing the stock of items located around the world.

The Maps Micro Application uses the concept of themes to display business data over a geographical map in a form that is most suitable to the business. This gives users a quick insight of the data relevant to the location. Users can customize the theme for the map to suit their business needs. Customizations include setting the data to be displayed as per the business requirements, and eventually facilitate the business process. As part of this Micro application, a base application framework is provided that is capable of accepting user-defined themes and provide the features mentioned above out of the box. It also includes several sample themes.

Technical Design

This section highlights the technical design of the Maps Micro Application. It also provides an overview of the components that were set up for the Micro Application. It includes the following topics:

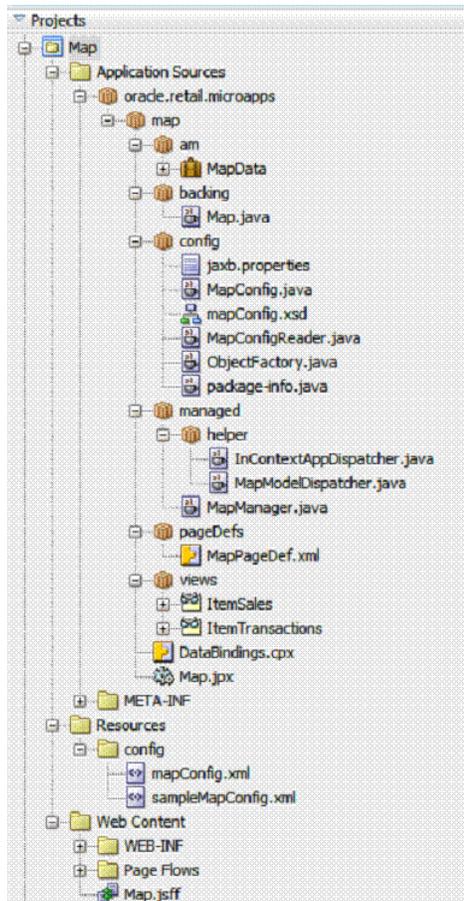
Components of the Sample Base Application

The sample base application already includes the functionality of displaying a geographical map using a pre-defined configuration file. The sample base application enables you to focus on creating new themes without having to set up a new application in Oracle JDeveloper, new project, or processing the configuration file for new themes. To access the sample base application:

- In Oracle JDeveloper, open the Workspace Retail Library package (MicroApplications.jws) you downloaded before. For more information, see [Accessing the Workspace Retail Library](#). The sample base application opens in Oracle JDeveloper.

The following figure illustrates the basic application folder structure (all folders expanded) of the sample application in the application explorer:

Figure 8–1 Basic Application Folder Structure of the Sample Application



The following table describes the important files and folders in the folder structure illustrated above. Any file or folder not included in the following table are automatically generated by Oracle JDeveloper:

Folder Name	File Name	Usage of the File/Folder
Am	MapData.xml	Application Module exposing the view objects as business components.
Config	Jaxb.properties MapConfig.java mapConfig.xsd MapConfigReader.java ObjectFactory.java package-info.java	Files generated by JAXB API to read and process main configuration file (sampleMapConfig.xml).
Managed	MapManager.java InContextAppDispatcher.java MapModelDispatcher.java	Managed bean and associated helper classes: Having the logic/algorithm to prepare themes dynamically and to implements the change listeners.
Views	ItemSales.xml	Sample theme provided in the base application for reference. This theme shows the total sales of particular item at a store location on a map.
Views	ItemTransactions.xml	Sample theme provided in the base application for reference. This theme shows the item transactions of a particular item at a store location on a map.
Resources/config	mapConfig.xml	Main configuration file for the themes related data.
Web Content	Map.jsff	JSF Page Fragment having Map control.
Web Content	testMap.JSF	Test JSF page incorporating Map.jsff JSF page fragment.

RMS Database Table for the Maps Micro Application

The Point Theme in the Maps Micro Application requires the longitude and latitude of every retail store and warehouse. The Micro Application does not support the Geo-coder service/point theme. To store this information, you will need to create a new database table in the RMS database. This section provides the DDL used for the table and the relevant sample data. Appropriate longitude and latitude data must be stored for every store and warehouses.

Note: The RMS database contains an ADDR database table that allows to capture the store and warehouse addresses. The map can then be set to use a Geo-Coder service to convert an address to the corresponding latitude-longitude pair. However, this approach requires a Geo-coder point theme to be supported in the map. At present Geo-Coder point theme is not supported.

DDL for the Database Table

This section provides the following SQL statement used to create the database table that stores the latitude and longitude information of every store and warehouse:

Note: The following SQL statement includes placeholders for the database user and table space name (highlighted in between <>). Ensure that you replace the placeholders with relevant values before running this statement on your database instance.

```
CREATE TABLE "<Database User>". "STORE_GEOMETRY"
(
  "STORE"          NUMBER(10,0) NOT NULL ENABLE,
  "STORE_TYPE"    VARCHAR2(2 BYTE) NOT NULL ENABLE,
  "STORE_NAME"    VARCHAR2(150 BYTE),
  "LATITUDE"      NUMBER(20,10),
  "LOGITUDE"      NUMBER(20,10),
  CONSTRAINT "STORE_GEOMETRY_PK" PRIMARY KEY ("STORE", "STORE_TYPE") USING INDEX
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS STORAGE(INITIAL 65536
  NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0 FREELISTS 1
  FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "<Table Space Name>" ENABLE
)
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING STORAGE
(
  INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT
)
TABLESPACE "<Table Space Name>" ;
```

Sample Data for the Database Table

The following table provides the sample data for the database table:

STORE	STORE_TYPE	STORE_NAME	LATITUDE	LOGITUDE
3212	S	Ottawa	45.4235	-75.6979
1131	S	Jacksonville	30.3322	-81.6557
5111	S	Sydney*	27.9634	-82.2073
5141	S	Melbourne	28.0836	-80.6081

The table includes the following columns:

- STORE – This column contains the identification number of the stores as present in the RMS schema in the Stores table.
- STORE_TYPE – This column specifies the store type, store (S) or warehouse (W).
- STORE_NAME – This column specifies the name or location of the store.
- LATITUDE – This column specifies latitude where the store is located.
- LOGITUDE – This column specifies the longitude where the store is located.

Run the Micro Application

To run the Micro Application in a Web browser:

1. Create a JSPX page.
2. Drag and drop the task flow (map-flow) created for the Maps Micro Application.
3. Run the page you created by selecting the Run option from the right-click menu. This action compiles the application and the relevant dependent projects, starts the default WebLogic Server domain, and deploys the application. It will also launch the application in the default Web browser for viewing and testing purposes.

The Web browser displays the geographical map with the two themes already present. Stores will be represented on the map by the colored pegs/pins/markings. Users can

click on the pins to view the item related information. Users can right-click on the pins to view the list of applications to be opened.

The following figures illustrate the application screens:

Figure 8–2 Application Screen When You Run the Application

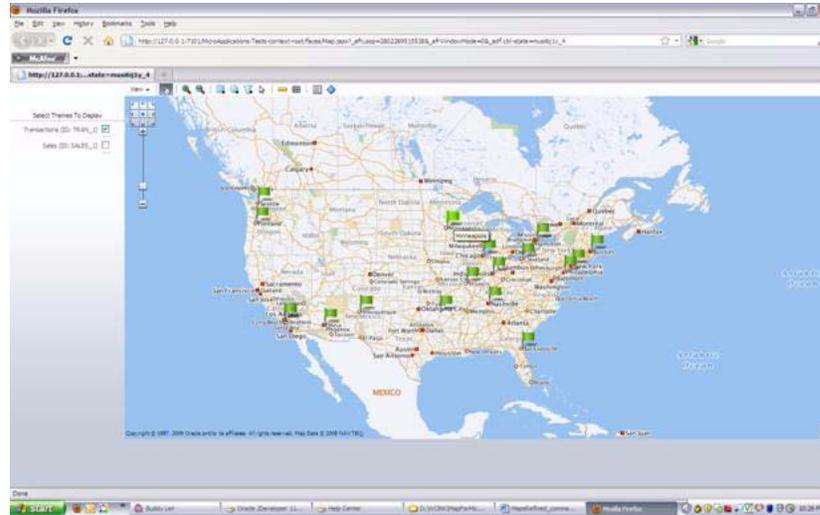


Figure 8–3 Item Related Information that Appears When You Click a Pin

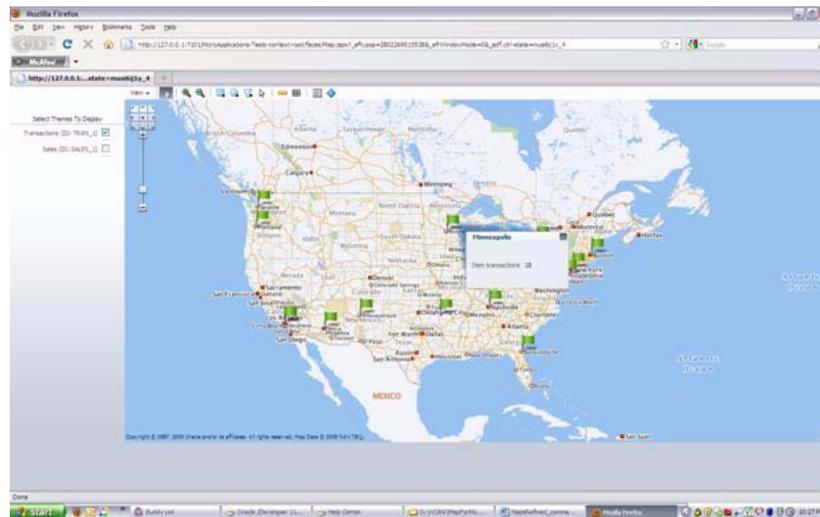
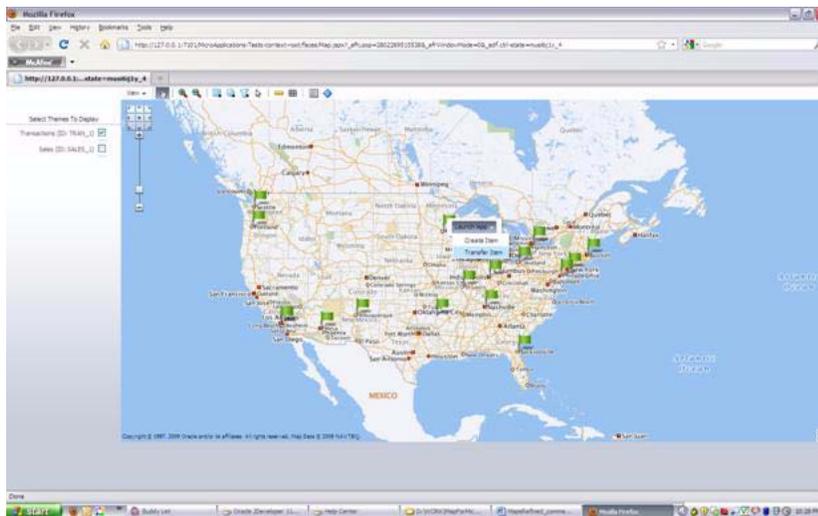
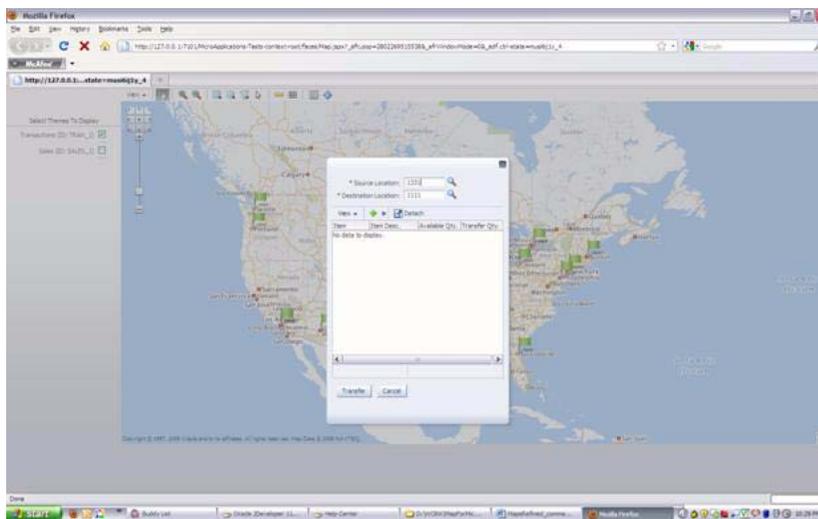


Figure 8–4 List of Applications that Appear When You Right-click on a Pin**Figure 8–5 Launching the Transfer Item Micro Application**

Validations

No Validations are done as part of this Micro Application. This application is used to launch other Micro Applications which may perform their own validation.

Using the Application

To use the Maps Micro Application:

1. In the map, click on any of the location to view the stock of items at that location.
2. Right-click on the location to view the list of Micro Applications that can be launched.
3. Select the relevant Micro Application. The Micro Application window appears.
4. Enter relevant details in the Micro Application.

The application then calls the RMS API of the corresponding application and executes the transaction. Status of the transaction appears in a pop-up window for the application.

Note: This is not the only usage of the application, you may customize the application to match your specific business needs.

Limitations

The following limitations apply to this Micro Application:

- The application does not check the privileges of the user to cancel orders.
- Security is not implemented in the application.
- The application can only perform the functionality that is available via the external API of the RMS application, which may not necessarily be the same when compared to the actual functionality in the RMS application.

Additional Customizations

You can customize the Maps Micro Application by adding new themes. This section describes how you can create and add a new theme. Creating a new theme involves the following steps:

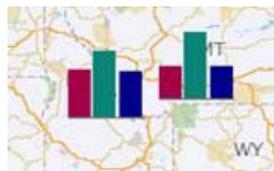
1. [Creating a View Object](#)
2. [Exposing the View Object for the User Interface](#)
3. [Adding the Theme to the Base Map](#)
4. [Setting Up the Theme Using a Configuration File](#)

Overview

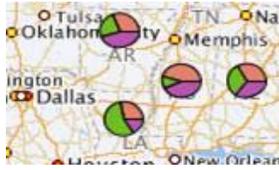
Currently the Maps Micro Application supports the following types of themes:

- **Bar Graph Theme** – In this type of theme, the map includes a bar graph at points that represent data values associated with those locations. For example, this tag may be used to display a bar graph at warehouse locations to show inventory levels at each warehouse. Or, you can build a map that includes a bar chart indicating the sales figure for each state.

Figure 8–6 Bar Graph Theme



- **Pie Graph Theme** – In this type of theme, the map includes a pie graph for each of the region in the underlying data as shown in figure below. The Pie Graph Theme tag is used to provide the ability to show statistics related to given locations on a map. This theme displays a pie graph at points that represent data values associated with those locations. For example, this theme may be used to display a pie graph at store locations indicating the sales values for a number of products at each store.

Figure 8–7 Pie Graph Theme

- Point Theme** – In this type of theme, the map includes an image or an HTML tag for each point in the underlying data as shown in figure below. The Point Theme provides the ability to specify different behaviors when the users click or hover the mouse pointer over a point on the map.

Figure 8–8 Point Theme

Creating a View Object

In the package *oracle.retail.microapps.map.view*, create a read-only view object using the following SQL statement:

```
SELECT ils.item, ils.stock_on_hand, ils.loc, ils.loc_type, s.store_name,
sg.latitude, sg.longitude
FROM item_loc_soh ils, store_geometry sg, store s
WHERE s.store = ils.loc and s.store = sg.store
and ils.item = :itemParam
```

Bind Variables

Name: ItemParam

Value: adf.context.viewScope.itemID

The variable **itemID** is present as an input parameter to the Map task flow. If the users of the task flow set this parameter, it will be available to the model layer in the view scope. It can then be used in user queries.

The view object must have a unique attribute defined as Key. This attribute will be used in the theme to uniquely identify the data point user clicks on.

Add the view object to the application module **MapData** in the package *oracle.retail.microapps.map.am*. Now this view object will be available for use in the user interface through DataControl.

Exposing the View Object for the User Interface

To expose the view object for the user interface:

- From the **oracle.retail.microapps.map.pageDefs** package, open the **MapPageDef** page definition file.

2. In the **Executable** section, create a new iterator. Select **data collection** as the view object just created. Take note of the iterator ID that is created. This iterator ID will be used while configuring map configuration file.

Adding the Theme to the Base Map

To add a theme to the base map:

1. In Oracle JDeveloper, open the **Map.jsff** JSF page fragment from the **web content** folder.
2. Click the **Source** tab.
3. Locate the base map tag `<dvt:map id="m1" ...>`.
4. Paste the following code inside the **dvt:map** tag as its child:

```
<dvt:mapPointTheme id="%ID%"
    value="#{pageFlowScope.MapManager.themesMap['%ID%']}"
    rightClickBehavior="Popup"
    builtInImage="PushPin_Red"
    rendered="#{pageFlowScope.MapManager.renderTheme['%ID%']}"
    clickListener="#{pageFlowScope.MapManager.clickListener}"/>
```

Note: Ensure that you replace the placeholder `%ID%` by a unique identification code. This will be used to configure the theme properties in the Map configuration file. You may choose a different **builtInImage** attribute for each theme using the Oracle JDeveloper's Property Inspector.

Setting Up the Theme Using a Configuration File

This framework enables you to add your own theme that represents your business data. While details for creating a new theme is covered in the sections above, this section provides details for the configuration file that you must use to set up the details of your theme. For the theme configured above, this section also provides a sample XML snippet.

The following table describes the parameters to be set in the configuration file:

SI No	Element	Parent Element	Attributes	Element Description
1	MapConfig	–	xmlns - used to avoid element name conflicts	–
2	StartingXCoordinate	MapConfig	–	X co-ordinate of the map that must be displayed when the map is rendered initially.
3	StartingYCoordinate	MapConfig	–	Y co-ordinate of the map that must be displayed when the map is rendered initially.
4	ZoomLevel	MapConfig	–	Initial zoom level of the map.
5	Themes	MapConfig	–	–

SI No	Element	Parent Element	Attributes	Element Description
6	Theme	Themes	<p>Type - Valid values are POINT_THEME, BAR_THEME, and PIE_THEME. This refers to the type of theme configured on the base map.</p> <p>Id - This is used to identify a specific theme. No two themes can have the same ID.</p> <p>Name - This specifies a name to the theme. Used for display in the user interface.</p> <p>Rendered - This can be set to true or false, based on which the theme will be displayed or hidden when the map is initially rendered.</p>	A theme is used to provide the ability to show statistics related to given locations on a map.
9	IteratorName	Theme	-	The iterator on which the theme is based. This must be of the same name as specified in page definition file for Map.jsff. It is used to iterate over the data in the database that is retrieved via a view object.
10	SpatialDataMapping	Theme	-	This specifies the geometrical position where the data points must be displayed in any theme.
11	Latitude	SpatialDataMapping	-	This specifies the attribute in the view object providing latitude for the data point.
12	Longitude	SpatialDataMapping	-	This specifies the attribute in the view object providing longitude for the data point.
13	Label	SpatialDataMapping	-	This specifies the attribute in the view object indicating a name or label of the data point.
-	Key	SpatialDataMapping	-	This specifies the attribute in the view object marked as key attribute.
14	BusinessDataMapping	Theme	-	This specifies the data to be displayed at the data point.
15	Data	BusinessDataMapping	<p>IterAttr - This specifies the attribute in the view object providing the numeric data for the data point.</p> <p>Label - Label for the data point.</p>	-
16	LaunchableApps	Theme	-	It is used to specify the applications that can be launched from this theme. These applications are packaged along with the Maps application, along with the corresponding task flow ids mentioned in the InContextAppDispatcher.java file.
17	App	LaunchableApps	<p>Name - Name of the application that can be launched.</p> <p>Id - Identification number of the application that can be launched.</p>	This specifies the details of application that can be launched from this theme.

SI No	Element	Parent Element	Attributes	Element Description
18	Parameter	App	<p>IteAttr - Name of the iterator column which will provide the value for the parameter that the application accepts.</p> <p>dest - This specifies the name with which this parameter value will be set in the ADF view scope, from where the Micro Application will retrieve its value.</p>	This is used when the application can be called with some of its attributes initially set. The values for these attributes are specified via parameters.

For the Point theme (configured in the sections above) to use the services of the framework, enter the following code as a child element to the *themes* XML element in the **mapConfig.xml** file (available in the resource folder):

```
<Theme type="POINT_THEME" id="IVT_1" name="Inventory Theme" rendered="true">
  <IteratorName>InventoryLevel1Iterator</IteratorName>
  <SpatialDataMapping>
    <Latitude>Latitude</Latitude>
    <Longitude>Longitude</Longitude>
    <Label>StoreName</Label>
    <Key>Loc</Key>
  </SpatialDataMapping>
  <BusinessDataMapping>
    <Data iterAttr="StockOnHand" label="Stock in hand"/>
  </BusinessDataMapping>
  <LaunchableApps>
    <App name="Create new Item" id="CREATE_ITEM"/>
    <App name="Transfer Item" id="TRANSFER_ITEM">
      <Parameter iterAttr="Loc" defaultValue="1231" dest="LOCATION"/>
    </App>
  </LaunchableApps>
</Theme>
```

Passing Parameters to the Micro Applications

When you launch a Micro Application from the Maps Micro Application, you can choose to pass variables for location selected in the map to the relevant Micro Application. Location selected in the map is available as the `#{viewScope.LOCATION}` EL Expression. In this case, the name of the variable *LOCATION* is the same as the value set for the *dest* attribute in the *Parameter* element.

From the code example above,

```
<LaunchableApps>
  <App name="Create new Item" id="CREATE_ITEM"/>
  <App name="Transfer Item" id="TRANSFER_ITEM">
    <Parameter iterAttr="Loc" defaultValue="1231" dest="LOCATION"/>
  </App>
</LaunchableApps>
```




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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2011, Oracle. All rights reserved.
This document is provided for information purposes only and the contents hereof are subject to change without notice.
This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.