Oracle® Revenue Management and Billing

Version 2.4.0.0.0

Self Service User Guide

Revision 2.0

E62186-01

March, 2015

ORACLE

Oracle Revenue Management and Billing Self Service User Guide

E62186-01

Third Party Content, Products, and Services Disclaimer

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third party content, products, or services.

Preface

About This Document

This document provides an overview of the Oracle Revenue Management and Billing Self Service application. It explains the sample pages available in the Self Service application and how to customize these pages as per the customer's requirements.

Intended Audience

This document is intended for the following audience:

- End-Users
- System Administrators
- Consulting Team
- Implementation Team

Organization of the Document

The information in this document is organized into the following sections:

Section No.	Section Name	Description
Section 1	Oracle Revenue Management and Billing Self Service	Provides an overview of the Oracle Revenue Management and Billing Self Service application.
Section 2	Self Service Sample Pages	Lists and describes the various sample pages available in the Self Service application.
Section 3	Customizing the Self Service Application	Explains how to customize the Self Service application.

Related Documents

You can refer to the following documents for more information:

Document	Description
Oracle Revenue Management and Billing Self Service Installation and Configuration Guide	Provides detailed information on how to install and configure the Oracle Revenue Management and Billing Self Service application.

Contents

1. Oracle Revenue Management and Billing Self Service

Oracle Revenue Management and Billing Self Service is a Web application which is used by an end-customer for searching and updating their information. The Self Service application provides a framework and templates for implementing a typical Web Self Service application. In addition, some sample web pages are provided.

2. Self Service Sample Pages

There are limited number of pages in the Self Service application. These pages are provided as an example which can be used by the implementation team to further build the Self Service application as per the customer requirements. All the sample pages provided in the application can be used by the implementation team for creating similar pages in the Web application.

This section describes the following:

- Login Related Pages
- Account and Person Related Pages

2.1 Login Related Pages

The following pages are related to registering and logging into the Self Service application:

- Login Page
- Registration Page(s)
- Password Reminder Page

2.1.1 Login Page

The **Login** page allows a registered user to login to the Self Service application with a user name and password. The page also provides links to enable the user to register as a new user and to request a reminder for a forgotten password.

Note:

Customer Contact Added - A customer contact is added when a customer successfully logs in or if a customer's attempt to login failed if you have set up the appropriate customer contact classes and types in the configuration properties file. Contact your system administrator for more information.

2.1.2 Registration Page(s)

The **Registration** pages allow a customer to sign up for the Self Service application. The initial Customer Registration page asks the customer to provide the account number and name in order for the system to identify the customer's person record.

Once the customer's person record has been identified, the customer is asked to provide their email address and to define a user ID and password. The customer must also choose a password reminder question and indicate an answer. This information is stored in the person record.

Upon successful registration, an email confirmation is sent to the customer.

Note:

Enabling Email Correspondence - Sending an email is only possible if the Self Service application has been configured with the appropriate settings for communicating with an SMTP server. Contact your system administrator to verify that these settings have been defined.

Preventing Web Access - A customer may only view information about related accounts if the account and/or person record has been marked to allow web access.

2.1.3 Password Reminder Page

This page is displayed if the customer has clicked the **Forgot Password** hyperlink from the login page. The password reminder question is displayed and the customer is asked to provide the answer. If the correct answer is provided, an email is sent to the customer's email address with the user name and password.

Note:

Enabling Email Correspondence - Sending an email is only possible if the Self Service application has been configured with the appropriate settings for communicating with an SMTP server. Contact your system administrator to verify that these settings have been defined.

2.2 Account and Person Related Pages

The following pages are available once the customer has successfully logged into the Self Service application:

- My Accounts
- Account Information
- Account Financial History
- Billing History
- Make Payment
- Stop Service
- Update Personal Information
- Change Password

2.2.1 My Accounts

The **My Accounts** page displays after logging in if the customer's person record is linked to more than one account. For each account, information about the main customer is displayed, along with the account's balance. Once you select one of the accounts, the **Account Information** page appears.

2.2.2 Account Information

The **Account Information** page displays high-level information about the person and account. This page also displays a list of contracts linked to the account.

2.2.3 Account Financial History

The **Account Financial History** page displays the financial history for the account. The page is similar to the account financial history page in Oracle Revenue Management and Billing, with some minor differences. For example, only the current amount and current balance information is displayed. The payoff amount and payoff balance are not displayed. In addition, the customer does not see any adjustment whose adjustment type's print by default flag is not checked.

2.2.4 Billing History

The **Billing History** page displays the information about the customer's bills including the date, amount and the running balance. In addition, if the Self Service application has been configured to integrate with Oracle Documaker to display bills, the **View** hyperlink appears for each bill to allow the customer to see an image of each bill.

Note:

Integrate with Oracle Documaker - Contact your system administrator for more information about whether the Self Service application has been configured to integrate with Oracle Documaker.

2.2.5 Make Payment

The **Make Payment** page allows the customer to enter a credit card payment to pay all or part of the outstanding balance for the account.

Note:

Configuration Required - When setting up the Self Service application, an appropriate tender type and appropriate auto pay source codes should have been defined in the system and referenced in the configuration properties file. Contact your system administrator for more information.

Customer Contact Added - A customer contact is added when a customer enters a credit card payment if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

2.2.6 Stop Service

The **Stop Service** page allows the customer to request that service for one or more of their active contracts stop on a date entered by the customer.

Note:

Customer Contact Added - A customer contact is added when a customer requests contract to be stopped if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

To Do Entry Added - A To Do entry is added when a customer requests contract to be stopped if you have set up the appropriate To Do type in the configuration properties file. Contact your system administrator for more information.

2.2.7 Update Personal Information

The **Update Personal Information** page allows a customer to change personal information, such as phone numbers, email address, and mailing address.

Note:

Customer Contact Added - A customer contact is added when a customer has changed personal information if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

To Do Entry Added - A To Do entry is added when a customer has changed personal information if you have set up the appropriate To Do type in the configuration properties file. Contact your system administrator for more information.

2.2.8 Change Password

The **Change Password** page allows the customer to change their password. After the password is changed, an email is sent to the customer's email address with the user name and password.

Note:

Enabling Email Correspondence - Sending an email is only possible if the Self Service application has been configured with the appropriate settings for communicating with an SMTP server. Contact your system administrator to verify that these settings have been defined.

3. Customizing the Self Service Application

The Self Service application provides the Oracle Revenue Management and Billing customers with a framework for implementing the transactions.

This framework enables the expansion of the functions available to your customer to include other business functions supported by the system. The following sections identify the different possibilities for customizing and expanding the Self Service application.

Note:

Upgrading after Customization - If you would like to modify any of the web application functionality provided by the system, you must make copies of the files provided and only make your changes in the copied files. If you have followed this procedure, every effort will be made to provide a seamless upgrade of the web application for bug fixes and future releases.

This section covers the following topics:

3.1 Technical Overview

The Self Service application is a regular J2EE Web application, conforming to the Servlet 2.2 specification. It uses JSP for the front end and a Java servlet in the back end. All Oracle Revenue Management and Billing access is done through XAI and there is no direct connection to the database. The technical design is based on model view controller architecture.

The presentation layer is separated from the data access and business rule layer. The JSP pages contain plain HTML (there are no applets, and very little JavaScript - close to none) with the visual layout of the screens. Dynamic elements, such as customer name, address, balance, etc. are represented as Java Beans (but not EJBs) and are embedded in the page using JSP directives. It is very easy for a Web designer to modify or replace the graphic design of the page without programming knowledge. All the logic of the application, including communication with XAI, sits in the Java Beans (Java classes representing data) and in the servlet.

Each of the Java Beans represents, roughly, an object in Oracle Revenue Management and Billing, or the set of information returned by an XAI service. Each bean knows how to retrieve its data through XAI if needed, and how to make updates (e.g. enter a payment).

The servlet is the controller that holds the pages together and implements security. Every user request gets intercepted by the servlet, which first checks that the user has been authenticated, performs any background actions if necessary, creates Java Beans with all the information to be displayed to the user, and passes the beans to the appropriate JSP page. Each action in Self Service, for example, show account information, show financial history, make payment, is a separate class.

3.2 Modifying Page Layout

If you would like to keep the functionality provided by a given web application page, but you would like to change the look and feel of the page, perform the following steps:

- Find the JSP file for the page that you want to modify. The JSP files are found in the `SelfService` directory.
- Copy the file and name your new file the same name as the file you are copying, but with "CM" in the beginning of the file name.
- Make the desired changes in your new JSP file.

Assuming that you want all the other functionality to remain the same, you are finished. You do not need to modify the menu or any action classes to call your new CM JSP file. The functionality provided always looks for the existence of a "CM" version of the JSP file and uses that file instead if one is found. If a "CM" version is not found, the JSP provided by the system is used.

3.3 Adding a New Transaction

As explained in the Technical Overview section, there are two parts for each transaction:

- On the client side, the JSP page contains the UI information
- On the server side, a servlet administers the processing of the transaction using one or more java beans containing the logic to communicate with Oracle Revenue Management and Billing.

If you want to add another action, for example enroll in AutoPay, you must do the following:

- Write another class for the servlet to recognize the action
- Write another bean that would know how to capture AutoPay information
- Add one or more JSP pages for the new screens that are used to interact with the customer.

Note:

Naming Convention - In order to ensure that new files added in future releases do not override your files, be sure to use "CM" for customer modification for the first two letters of every new file.

This section explains various tasks, such as:

- Identifying a Transaction
- Creating an XAI Schema
- Creating a Java Bean
- Creating an Action Class
- Using Control Classes
- Creating a JSP Page

3.3.1 Identifying a Transaction

In principal, every transaction that exists in the system could get implemented in the Self Service application, since all Oracle Revenue Management and Billing objects are described as internal services, which can be accessed via XAI.

The question often will be whether the implementer will want to maintain the degree of diversity and sophistication that exists in many transactions, or if the maintenance of the main object attributes might already fulfill the requirements.

3.3.2 Creating an XAI Schema

This section assumes that you are familiar with the topics described in XML Application Integration. The main steps for our purpose are to go in the **XAI Schema Editor** to the **Schemas** menu item, and to create an XAI inbound service. Use the search function to assist you in finding the correct service. The schema needs to be saved and registered. You should also test the new service.

Note:

Naming Convention - In order to ensure that new files added in future releases do not override your files, be sure to use "CM" for customer modification for the first two letters of your new service. In addition, set the owner flag to Customer Modification.

3.3.3 Creating a Java Bean

The java bean is responsible for retrieving information from the respective Oracle Revenue Management and Billing object. Within the classes directory of the Self Service application you will find two kinds of java classes: One group ending their class name with `Action` (these are discussed in the next section) and the other group ending with `Bean` (so called object beans).

You can take almost any java class ending with `Bean` as an example and template for developing a new bean.

Note:

Not all classes with names ending with `Bean` are object classes with the characteristics mentioned below. Some of those beans serve only as container classes describing the data members of an Oracle Revenue Management and Billing object, without having any additional logic to retrieve data via XAI. Usually such container classes are used when there are more than two data members in the object and will be referenced by an object class retrieving more than one record. For example, in the Account Info page we are displaying the contracts for an account. The object bean retrieving those contracts is the `SAforAccountBean` and it stores the contracts in a data array, whose members are instances of the container class `SAforAccount`.

The following is a description of the common parts of all object beans:

- All beans belong to the package named `com.splwg.selfservice`.
- All beans communicating with XAI will need to import the package named `org.dom4j.*`.
- All beans implement the java class named `java.io.Serializable`.
- Variables are always defined in private scope, and a public function exists to populate and/or to retrieve their values.

- The `property` object is used to hold all properties that exist in the `SelfServiceConfig.properties` file (e.g. the XAI server address). This object needs to be transferred from the action bean to the object bean. You may either achieve this by providing the property object as an argument to the constructor, or by explicitly setting it in a separate method.
- One main function contains the XAI request to be sent to the XAI server and the code to parse the response. For this purpose it might be helpful to use the request and response xml, so that the elements and attributes can be copied into your code. You can see this xml by using the **XAI Submission** page or the **XAI Dynamic Submission** page for the appropriate XAI inbound service. In addition, if XAI logging is switched on, you may view the `xai.log` file to see all requests and responses received or sent by the XAI server. Another option is to use the **XAI Schema Editor** testing tool, in order to submit such a request to XAI. All attributes within elements you may want to retrieve from the service need to get parsed and stored into variables. Any error message returned (or created by the bean) should be stored into a variable (usually named `errorMsg`), and its value will be inquired by the calling method. Refer to Handling Error Messages section for more information.

For our test case example of creating a service for service address maintenance, you will likely want to create two main methods:

- One method to retrieve the data according to the primary id to be provided
- The other one to enable the update of the service address according to all attributes provided as input parameters.

Note:

Naming Convention - In order to ensure that new files added in future releases do not override your files, be sure to use "CM" for customer modification for the first two letters of every new file.

3.3.4 Creating an Action Class

One kind of java class has an `Action` suffix and controls the process flow, which might be more or less complicated according to the required functionality.

This action class is always called from the controller class. Refer to Using Control Classes section for more information.

The common characteristics of this type of class are:

- All action classes implement the interface `SSvcAction`. This interface defines one common method named `perform`. The HTTPServlet, HTTPServletRequest, and HTTPServletResponse instances are required arguments, ensuring accessibility to the servlet properties and servlet session to every class.
- A transaction might have different steps. For example, the first step for the update personal info transaction is to show the current personal info data. The second step processes the updated data. According to the different steps, different methods are called. For an example of a multi-step transaction, refer to PayOnlineAction.java.
- An instance of the object bean is created and after setting the input parameters, the main function is called.

- According to the result of the previous step, a JSP page is called (by calling the forward method of the servlet request dispatcher).

For your new transaction, you must create an action class to execute the appropriate steps. You must also go to the Actions section of the `CMSelfServiceConfig.properties` file to indicate the name of the class along with its action (reference) name. This is the name used to reference the class in your object bean.

The `CMSelfServiceConfig.properties` file was configured at installation time. Refer to the installation documentation for more information. This file can be found in the `WEB-INF` directory of the `SelfService` directory in your server. Examples of how to define your action name may be found in this file. The property name used to define the property must start with `com.splwg.selfservice.Action`. The `SSvcController` class will dynamically load these actions in the `initActions` method.

3.3.5 Using Control Classes

There are some classes that exist to perform a special task:

- The `SSvcController` class is the servlet called by all JSP forms on submission. The first time it is called is when starting the Self Service application. (The web application directory `SelfService` contains the file `index.html`, which refers the user to this servlet with a login action.) According to the action called, the respective action class is invoked. This class also keeps track of the servlet session instance, and if it has timed out, it forces the user to log on again.
- The `XAIHTTPCall` class administers the call to XAI. All object beans communicating with XAI call it. You won't need to change anything in this class.
- The `Util` class contains static methods to be called from various classes in the application. Currently it contains only one method that replaces null values with blanks.
- The `XMLEncoder` class may be called to make sure that any input string complies with xml or http standards. For example, special characters like an ampersand are replaced.

Our service address maintenance action class will have two steps. The first step is called when selecting the transaction from the menu, and reads the respective location record by invoking the created object bean. The second step is called upon submission of the form, and processes the changes submitted.

3.3.6 Creating a JSP Page

Every JSP page consists of three parts: the HTML part, the server script and the client script. All HTML elements are legitimate in a JSP page, and both the client script as well as the server script may dynamically control the display elements of the page.

This capability is used several times in the application. One example is the dynamic creation of lists, like the Contract list in the Account Information screen, or the address list in the Update Personal Info screen.

A server script using java may use all packages and classes available to it. Object beans are transferred from the action bean as attributes of either the session or the request object. Depending on this differentiation, the JSP routine `jsp:useBean` will have either the scope request or session.

Other JSP or html files may get included with the include command.

The client script contains mostly validation methods called before the form is submitted.

By sticking to the style sheet with its already defined styles and avoiding style definitions in HTML elements, you will keep graphically in line with the existing pages.

For the service address maintenance you will probably create two pages. One page displays all the data retrieved, where either all or some will be modifiable by the customer. The other page informs the customer that the data was updated successfully.

Note:

Naming Convention - In order to ensure that new files added in future releases do not override your files, be sure to use "CM" for customer modification for the first two letters of every new file.

3.4 Modifying the Menu

A file called `Menu.jsp` contains a `DisplayMenu` function that determines what menu items to display. This function first checks whether or not a `CMDisplayMenu` function exists. If one does, that method is called to display the menu items. If no CM menu file exists, then the `CIDisplayMenu` function is used.

To customize the items listed in the menu, perform the following steps:

- Edit `CMMenu.jsp` (found in the `SelfService` directory) and modify the `CMDisplayMenu` method to add/remove menu contents as desired. For example, you may choose to hide a menu item that you do not want to provide for your customers and/or you may add new menu items that refer to new pages you have created.

For each new JSP file that you have created, you must do the following:

- Include the files named `Menu.jsp`, `CIMenu.jsp` and `CMMenu.jsp` into your JSP file.
- Call the `DisplayMenu` method from your JSP file. Refer to any of the JSP files provided by the system as an example.

3.5 Modifying the Style Sheet

JSP UI elements have the style defined in their class attribute. The declaration of all styles is found in the `wss.css` file within the `SelfService` directory. The JSP pages reference the style sheet by including a file called `IncPreamble.jsp`. When this JSP is loaded, it checks whether or not a `CMwss.css` file exists. If one does, that style sheet is used for the page. If no CM style sheet file exists, the page uses the original `wss.css`.

To customize the style sheet provided, perform the following steps:

- Make a copy of the `wss.css` file and call it `CMwss.css`.
- Make changes to `CMwss.css` as desired.

3.6 Handling Error Messages

The main methods in the object beans (the ones communicating with XAI) have boolean return values. The action bean calls those methods.

If the call to such a method returned true, the program flow continues. A return code of false informs the action bean that an error has occurred.

As a default, the error message that may be received from the Oracle Revenue Management and Billing service or that may be created by the method is stored in a variable named `errorMsg`, and the action bean retrieves this variable.

Usually the action bean then redirects the program flow back to the calling JSP page.

Each JSP page includes a JSP file named `ErrorHandler.jsp` at the beginning of its form, which checks whether the transferred error message variable is unlike null, and if so displays a message box with the error message.

An error message returned from the server is displayed in the appropriate language defined for the self-service application. The Oracle Revenue Management and Billing message should be displayed or trapped and mapped to a more user friendly message. Refer to NewPaymentBean.java for an example of overriding a message received by Oracle Revenue Management and Billing.

All messages that are generated by a Self Service object or an action bean are also stored in the system message table to take advantage of the multi-language storage of messages. The application displays the message in the appropriate language for the self-service application.

Note:

New Error Messages - If your new page requires new messages, define the messages in the system using message category 90000 or greater, which are reserved for implementations.

3.7 Using Session Object

Data you want to transfer from an action bean to a JSP page may be put either into the http request object or into the http session object. The difference is in the scope.

If the information will only be valid for the specific request, then store the information in the request object. The session object stays active until the session is ended or timed out, and therefore any information you want to keep accessible in your application can be stored in the session object.

In the Self Service application, `ValidUserBean` is stored in the session object. On logon its information (account ID, user name, entity name, and currency symbol) is loaded and stays accessible by all Self Service JSP pages.

