Implementation Guide for Oracle Self-Service E-Billing

Version 6.0.4, Rev. A

January 2012

Contents

1 What's New in This Release

What's New in Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.4, Rev. A

Table 1 lists changes described in this version of the documentation to support release 6.0.4, Rev. A of the software.

Table 1. New Product Features in Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.4, Rev. A

Topic	Description
"Configuring International Bank Routing" on page 243	New topic. Added instructions for configuring routing numbers for international banking payments.
"Using or Simulating Single Sign-On" on page 45	Modified topic. Clarified the procedure to simulate a single sign-on system. Added a procedure to configure Oracle Self-Service E-Billing to use a single sign-on system.

What's New in Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.4

Table 2 lists some of the changes in this version of the documentation to support this release of the software.

Table 2. Product Features Added to the Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.4

Topic	Description
"Localizing the User Interface" on page 23	New topic. Describes how to localize the Billing and Payment and Customer Service Representative applications for one or more new languages.
"Customizing User Security Questions" on page 38	New topic. Describes how to customize the security questions.
"Using or Simulating Single Sign-On" on page 45	Modified topic. The procedure to simulate single sign-on has been updated for implementations using Oracle Self-Service E-Billing authentication with an external identity store.
Customizing the Content of Email Notifications on page 55	New topic. Describes how to customize the text and variables that appear in email notifications.
"Customizing the Default Display Patterns Used in Email Notifications" on page 56	New topic. Describes how to customize the default patterns used to display dates, times, date and time, integers, and amounts in Oracle Self-Service E-Billing applications.

Table 2. Product Features Added to the Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.4

Topic	Description
“Email Notification Template Content (Business Edition)” on page 59 “Email Notification Template Content (Consumer Edition)” on page 70	New topics. These topics were moved from *Application Guide for Oracle Self-Service E-Billing (Business Edition)* and *Application Guide for Oracle Self-Service E-Billing (Consumer Edition)*. Template text has been updated to support localization. The Statement Threshold notification type has been added to both topics for the new ThresholdExceedNotify job.
“Configuring Batch Reporting” on page 95	New topic. Describes the configuration options available for batch reporting.
“Creating a PDF Template for Reporting” on page 124	New topic. Describes how to create an RTF template to generate reports in PDF format.
“Customizing Charts” on page 143	New topic. Describes how to customize Oracle Data Visualization Tool (DVT) charts for reporting. Oracle DVT replaces KavaChart.
“Configurable Chart Properties” on page 145	New topic. Describes the configurable properties for each type of Oracle DVT chart available with reporting.
“Customizing the Statement Summary Chart” on page 152	New topic. Describes the configurable properties available with the vertical bar chart displayed in the Statement Summary.
“Reporting on User Audit Data” on page 153	New topic. Describes the audit data maintained on user enrollment activity recorded in Oracle Self-Service E-Billing.
“Reporting on System Administrator Audit Data” on page 157	New topic. Describes the audit data maintained on system administrator activity recorded in the Command Center application.

What’s New in Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.3

Table 3 lists some of the changes in this version of the documentation to support this release of the software.

Table 3. New Product Features in Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.3.

Topic	Description
“Auditing Database Administration Activity” on page 23	New topic. Describes the external database administrator auditing, as required by PCI DSS.
“Data Dictionary” on page 23	New topic. Describes where to find the new Data Dictionary files in the Oracle Self-Service E-Billing product directories.

Table 3. New Product Features in Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.3.

Topic	Description
“Customizing User Management and Security” on page 33	Modified topic. Added functional changes to meet the Payment Card Industry Data Security Standard (PCI DSS).
“Customizing User Enrollment” on page 34	New topic. Describes how to customize the new enrollment functionality, as required by PCI DSS.
“Customizing End User and CSR User Passwords” on page 35	New topic. Describes how to customize the end user and CSR user password validation rules, as required by PCI DSS.
“Customizing the Administrator User Password” on page 37	New topic. Describes how to customize the Command Center administrator user password, as required by PCI DSS.
“Deactivating and Reactivating the Master Customer Service Representative Administrator User” on page 39	New topic. Describes how to deactivate and reactivate the default Customer Service Representative (CSR) user account, as required by PCI DSS.
“Customizing Enrollment Validation” on page 39	New topic. Describes how to customize the new enrollment validation functionality, as required by PCI DSS.
“Customizing Account Lockout” on page 40	New topic. Describes how to customize the new user account lockout feature, as required by PCI DSS.
“Customizing Reactivate Account Lockout” on page 42	New topic. Describes how to customize the new reactivate account lockout feature, as required by PCI DSS.
“Customizing Profile Management” on page 42	New topic. Describes how to customize the enhanced profile management, as required by PCI DSS.
“Customizing Acegi Configuration” on page 44	New topic. Describes how to customize the enhanced Acegi user enrollment framework, as required by PCI DSS.
“Using or Simulating Single Sign-On” on page 45	New topic. Describes how to simulate a single sign-on user.
“Customizing Threshold Values for Batch Reporting” on page 142	New topic. Describes how to set the batch reporting threshold value for each report.
“Input File Specifications and Data Mapping” on page 273	New chapter. This chapter was moved from *Database Guide for Oracle Self-Service E-Billing*, which is no longer a part of the Oracle Self-Service E-Billing documentation set. Additional fields were added to the summary-level detail file format for PA-DSS compliance.
“Customizing the Payment Amount Format” on page 242	New topic. Describes how to customize the payment amount format.
Using the Oracle Self-Service E-Billing Payment APIs	Removed chapter. API code and lists of associated methods have been removed from the book. For information about using APIs, see Accessing Oracle Self-Service E-Billing Javadoc on page 31.

Table 4 lists some of the changes in this version of the documentation to support this release of the software.

Table 4. Product Features Added to the Implementation Guide for Oracle Self-Service E-Billing, Version 6.0.1

Topic	Description
Adding a Custom Message Provider on page 82	New topic. Describes how to add a custom message provider, such as SMS.
Debugging Oracle Self-Service E-Billing on page 28	Modified topic. Addition of a security caution on updating the log4j_cc.xml file.

2 Customizing Oracle Self-Service E-Billing

This chapter covers general information to get started customizing your application. It includes the following topics:

Overview of Oracle Self-Service E-Billing Architecture

Oracle Self-Service E-Billing includes three applications. Each application is packaged as one Enterprise Archive (EAR) file:

- **Billing and Payment.** Users interact with the Billing and Payment online interface to view their statements, make payments, manage their business hierarchies, and so on. Use the information in this guide to help you customize the preconfigured functionality for your company's implementation.
- **Command Center.** Your system administrator uses the Command Center to manage the live Oracle Self-Service E-Billing production environment. You do not customize this application, although you can create custom jobs if necessary.
- **Customer Service Representative (CSR).** Customer service representatives use the CSR application to assist Oracle Self-Service E-Billing users.

This guide assumes you have installed Oracle Self-Service E-Billing and deployed these applications, and can run and view them successfully. For information about installing Oracle Self-Service E-Billing, see *Installation Guide for Oracle Self-Service E-Billing*.

About the Billing and Payment Application

The Billing and Payment application is the online bill presentment and payment interface for users. For information about the preconfigured use cases provided in the Billing and Payment application, see *Application Guide for Oracle Self-Service E-Billing (Business Edition)* or *Application Guide for Oracle Self-Service E-Billing (Consumer Edition)*.

The Oracle Self-Service E-Billing interfaces are built upon Struts and Tiles. The Struts actions talk with the Service APIs which then access different modules, such as Hierarchy or Reporting. The Billing and Payment application includes the following feature modules:

- **Statement Module.** This is the core J2EE functionality of Oracle Self-Service E-Billing self-service software. The Statement module manages the access and display of statement data, enrollment, logging, and production (administrative) environment. Preconfigured presentment functionality includes bill summary, account, service, and usage summaries, usage detail, transaction detail, and the ability to dispute a transaction. A dashboard is provided for B2B users and includes a summary of recent charges across all accounts in a company.
- **Payment Module.** A complete payment scheduling and management with real-time and batch connections to payment gateways for Automated Clearing House (ACH) and credit card payments, and payments using various payment processing service providers. Includes user enrollment functions for both viewing and paying bills, setting up account information, making payments, scheduling payments, payment reminders, recurring payments, and so on. Administrative functions include setting up Payment jobs, Payment module settings, and viewing reports.
- **User Management Module.** A framework to authenticate and authorize a user using roles-based control (RBAC). After the user has been authenticated, the user can then access different Oracle Self-Service E-Billing features such as hierarchy, cost management, reports, and so on. Users can view personal profiles and optionally change their names, password, and email accounts.
- **Unbilled Usage Module.** Presents a report of detailed transactions, also called unbilled details, that have occurred since the last statement close date.
- **Split Billing Module.** A feature that lets a service provider define a set of rules that enables the application to categorize business and personal expenses for all transaction detail in a service agreement. The service provider can change the application order of the rules, the rule definition, and the number of rules to be applied. Users can manually change the automated split-billing categorization through the online application.
- **Notifications Module.** Users can configure personal event-based notification preferences that control the delivery of email messages to the user for events such as when a new bill is ready for viewing online or a payment is confirmed. Notification can be generated by a batch process following an event (batch notifications) or in response to a user action (instant notifications).
- **Hierarchy Management Module.** Manages the life cycle of a hierarchy: creation, modification, expiration, deletion, copy, move, search, hierarchy-based access control (HBAC), and so on.
- **Analytics Module.** Provides an analysis of group spending, group spending trends, account billing overview, details, and trends, contract billing overview and trends, contract call details, and total cost by plan. Chart type in reports (vertical bar, horizontal bar, stacked bar, multiple line, and pie chart) is selectable. All report row data can be configured into a trend report. Users can customize and save all report parameters. Ability to run reports from virtual nodes created when a user is assigned to two or more locations within a hierarchy.
- **Cost and Budget Management Module.** A set of tools to manage billing cost, such as rebill, cost reallocation, and budget management.

- **Top X Reporting Module.** Standard reports showing most expensive calls, longest calls, most frequently called numbers, destinations, or countries, highest spender, and highest spending contracts by usage type or call type. Users can also create a Find Calls report showing a list of calls based on a custom search.
- **Database Presentment Engine.** A framework to retrieve data from different data sources and present them as HTML, XML, or CSV. Offers features like paging, sorting, charting, bread-crumb, batch report, custom report, and so on.
- **DB Access.** The majority of the Oracle Self-Service E-Billing code uses Hibernate, which is an object-relational mapping tool, to access OLTP database. The access to OLAP is through direct JDBC call to boost performance.
- **Transaction Management.** Oracle Self-Service E-Billing uses distributed transaction (XA or JTA transaction) to manage database access and JMS access. The transaction is managed through the Spring Framework. The Spring framework is also used to manage object creations, and so on.
- **OLTP Database. The** Oracle Self-Service E-Billing transactional database, which includes transaction data such as user, account, services, and hierarchies.
- **OLAP Database.** The Oracle Self-Service E-Billing non-transaction database, which includes billing data. It is a star-schema based data warehouse and includes dimensional tables, fact tables, and hierarchy tables.
- **OLTP-OLAP Synchronizer.** A process which synchronizes the information from OLTP to OLAP database. Currently, the main information being synchronized is the hierarchy. Any change made to hierarchy on the OLTP side will be synchronized at OLAP. The OLAP hierarchy schema is specially designed for queue performance and different from the OLTP hierarchy schema. However, they have the exactly the same content; even the hierarchy node IDs are the same.

Default Installation Directory

The default installation directory for Oracle Self-Service E-Billing is:

- **UNIX.** `/opt/Oracle/eBilling`
- **Windows.** `Oracle\eBilling`

It is possible to change the default directory during installation. This guide refers to the directory where you have Oracle Self-Service E-Billing as *EDX_HOME*.

Billing and Payment Application EAR File Structure

This Billing and Payment application EAR file can be found in the following directories:

- **Oracle WebLogic.** *EDX_HOME*/`eBilling/J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear` (or the *EDX_HOME*`\eBilling\J2EEApps\ebilling\weblogic\ebilling-weblogic-10-6.0.4.ear` directory on Windows)
- **IBM WebSphere.** *EDX_HOME*/`J2EEApps/ebilling/websphere/ebilling-websphere-6-6.0.4.ear`

In general, you deploy the Billing and Payment application EAR file in a cluster environment for the purpose of failover and load balance.

The following components are packaged inside the ebilling-weblogic-10-6.0.4.ear file:

- **`ebilling-weblogic-10-6.0.4.ear.`** This is the root directory and contains the EJB JavaARchive (JAR) and Web ARchive (WAR) files.
- **`ebilling-weblogic-10-6.0.4.ear/lib.`** Contains the list of third-party lib files used by the Billing and Payment application.
- **`ebilling-weblogic-10-6.0.4.ear/META-INF.`** Contains the J2EE META-INF directory.
- **`ebilling-weblogic-10-6.0.4.ear/xma.`** Contains a list of internal library files used by the Billing and Payment application. In this directory there is one JAR file called api-*version_number*.jar, where *version_number* is the Oracle Self-Service E-Billing version. This JAR file has all the public Oracle Self-Service E-Billing APIs defined.

The following components are packaged under the directories inside the Billing and Payment application WAR file, ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war:

- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/_includes` and `ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/_templates.`** Contains JSP page fragments used by the Billing and Payment application; many of these are tiles.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/_assets.`** Contains images, JavaScripts, and CSS files used by the Billing and Payment application.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/hierarchy.`** Contains Hierarchy-related JSP pages.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/usermanagement.`** Contains User-Management-related JSP pages.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/reporting.`** Contains Reporting-related JSP pages.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/payment.`** Contains Payment-related JSP pages.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/contacts.`** Contains Contacts-related JSP pages**.**
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/dispute.`** Contains Dispute-related JSP pages**.**
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/unbilled.`** Contains Unbilled-related JSP pages**.**
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF.`** Contains the J2EE WAR file WEB-INF directory.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/classes/azcfg/policy.`** Contains the Oracle Self-Service E-Billing RBAC policy file.
- **`ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/classes/lib.`** Contains libraries used by the WAR file.

NOTE: Some of the WAR file subdirectories are inherited from another legacy Oracle Self-Service E-Billing application and are not used directly by Oracle Self-Service E-Billing.

About the Command Center Application

The Oracle Self-Service E-Billing Command Center is a separate application and is packaged as a separate EAR file. You deploy the Command Center on a separate application server. A system administrator uses the Command Center to run batch jobs and monitor the production environment. Command Center consists of the following components:

- **Command Center User Interface.** The Command Center user interface is based on Servlet-JSP technology, not struts and tiles.
- **Jobs.** A Command Center job is a process which an administrator for Oracle Self-Service E-Billing must schedule and run using the Command Center console (UI). Oracle Self-Service E-Billing comes with a set of predefined jobs, such as OLTP loader, batch generator, Hierarchy importer, and so on. A job consists of one or more tasks, and each task performs a specific piece of the processing. Each task is implemented as an EJB and has its own configuration parameters which the administrator also configures using the Command Center UI. When a job runs, the tasks that make up each job run sequentially.

  For more information about configuring and running jobs in the Oracle Self-Service E-Billing Command Center, see *Administration Guide for Oracle Self-Service E-Billing*.
- **PWC API.** A set of APIs used to manage jobs.

Command Center EAR File

The Command Center EAR file can be found in the following directories, where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing:

- **Oracle WebLogic.** *EDX_HOME/*`J2EEApps/commandcenter/weblogic/command-center-weblogic-10-6.0.4.ear` (or the *EDX_HOME*`\J2EEApps\commandcenter\weblogic\command-center-weblogic-10-6.0.4.ear` directory on Windows)
- **IBM WebSphere.** *EDX_HOME/*`J2EEApps/commandcenter/websphere/command-center-websphere-6-6.0.4.ear`

In general, you are not expected to modify this EAR file during deployment. Deploy the Command Center on a separate application server.

About the Customer Service Representative Application

The Customer Service Representative application is used by customer service representatives to assist customers. A CSR can impersonate an end-user. For information about preconfigured CSR use cases provided with Oracle Self-Service E-Billing, see *Application Guide for Oracle Self-Service E-Billing (Business Edition)* or *Application Guide for Oracle Self-Service E-Billing (Consumer Edition)*.

About the CSR Application EAR File

The CSR EAR file can be found in the following directories, where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing:

- **Oracle WebLogic.** *EDX_HOME*/`J2EEApps/csr/weblogic/csr-app-6.0.4.ear` (or the *EDX_HOME*\\`J2EEApps\csr\weblogic\csr-app-6.0.4.ear` directory on Windows)
- **IBM WebSphere.** *EDX_HOME*/`J2EEApps/csr/websphere/csr-app-6.0.4.ear`

Guidelines for Customizing Oracle Self-Service E-Billing

Oracle Self-Service E-Billing provides a set of core functions, such as reporting and hierarchy management, and a sample user interface (UI) to demonstrate these functions. The contract between Oracle Self-Service E-Billing core and the UI is a set of APIs. These APIs and the Java-docs are contained in the API JAR file of the EAR file. You must use these APIs for your customization purposes; do not modify or bypass these APIs unless explicitly instructed in this guide.

The sample application demonstrates how Oracle Self-Service E-Billing functions. You can customize your user interface, such as the billing, reporting, or even hierarchy.

NOTE: Because of the complexity of the Hierarchy Management UI, it is recommended that you try and keep your UI as close as possible to the sample hierarchy UI to reduce your workload.

The functions exposed by the APIs exceed the ones demonstrated through the sample UI. Please consult the API Java-docs and other topics of this guide for details. You can customize the Oracle Self-Service E-Billing application to take advantage of these functions.

When you have to change existing Oracle Self-Service E-Billing files, such as a JSP or a Velocity template, you can work either on an existing file or copy it and work on the copy. The second method could be more time consuming but will save you more time for migration. Keep the history of customization changes in a source control system.

Customizing the User Interface Files

The Oracle Self-Service E-Billing user interface-related files can be found in the following Web application folders (packaged in the application EAR file):

- **The _assets.** Contains all images, CSS files, and scripts used in the application.
- **The _templates.** Template files for formatting and screen orientation.

Every JSP can extend any one of these templates.

All Oracle Self-Service E-Billing screens pick up their styles from a common file, swan.css (in the _assets/swan/ folder). This file is imported in all the templates.

All JSP files can be found in the respective module folders.

UI customization can range from changing the look-and-feel or adding your own struts action classes.

Customizing the Existing Look-and-Feel

The Oracle Self-Service E-Billing UI is based on Tiles definitions. The user interface properties, such as color and font size, are controlled using a style sheet (CSS file).

The stylesheet defines the styles for all classes defined. You can define as many stylesheets as required, however, leave the class name the same as it is in take1.css.

The template files must also import the corresponding customized CSS files as necessary. Then the JSPs will use the new styles.

You can modify the Tiles definitions file to use your own Tiles. All the Tiles definitions are in the `ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/` directory.

The hierarchy UI-related JSP pages are in the `ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/hierarchy` directory.

Reporting-related JSP pages are in the `ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/reporting` directory. If necessary, customize these JSP pages.

In special cases, when you use the Oracle Self-Service E-Billing presentment engine to generate a report or a search page, the result of the query is not presented by JSP, instead, a set of Velocity templates are used. These templates are defined in the *EDX_HOME*`/templates/common/lib` and reporting directories. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing. Do not touch the VM files in the lib directories. For the files under reporting, you can customize them if necessary. However, most of the time you can customize reports using report xml files without touching the VM files.

Customizing Web Document Styles

Oracle Self-Service E-Billing provides Cascading Style Sheets (CSS) as a mechanism for adding style, such as fonts, colors, and spacing, to Web documents to provide a user-customized interface.

The user interface of Oracle Self-Service E-Billing uses industry standards (consistent page layout, navigation bars, bread crumbs, and logically labeled controls) to make a consistent and intuitive user experience. The use of Cascading Style Sheets ensures separation of style from presentation.

The page layout of the Oracle Self-Service E-Billing application consists of the following body areas:

- Pagewrap
  - Top_page
  - Logo
  - Userlinks
  - Tabbar and tabs
  - Subtabbar and subtabs
  - Sidecontent
    - Quicklinks: Header and Quicklinklist
    - Reportcontext: Header and Reportcontextlist

- Maincontent
  - Breadcrumb
  - Pageheading
  - Pagetabs (When applicable)
  - Errormessage
  - successmessage
  - Subtitle (Repeats at the top of each module)
  - Buttonbar downloadPrint (When applicable)
  - Buttonbar (When applicable)
  - Contextbox
  - Infomessage (When applicable)
  - Buttonbar (When applicable)
- Clearline
- Footer. All style sheets reside in the _assets/css directory. The primary style sheet is take1.css, which is the only style sheet that the application uses with the exception of all Printer Friendly-rendered pages.

  Oracle Self-Service E-Billing calls for the Cascading Style Sheet file from the main templates, which are in the _templates directory. The JSP file names are:

  - simplelayout.jsp
  - simplelayout1.jsp
  - popupLayout.jsp
  - paymentLayout.jsp
  - dashBoardLayout.jsp

Using Custom JSP Pages and Action Classes

The user interface components can be found in the ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war.

It is possible to add your own UI components such as JSP, JavaScripts and so on.

After you create your own action class, you must modify the struts-config.xml file to register it.

Using Velocity Templates

The Oracle Self-Service E-Billing reporting UI is based on Velocity templates, an open source project. The product offers a set of preconfigured templates to implement common UI features such as paging, sorting, charting, print-friendly, and download.

You can customize these preconfigured templates either by modifying them directly or by copying and then modifying. If you do copy and modify, configure the report XML files to pick up your new templates.

About Customizing Reports

All of the report XML files defined in the *EDX_HOME*/`config/rpt` directory (or the *EDX_HOME*`\config\rpt` directory on Windows) are for the preconfigured Oracle Self-Service E-Billing user interface. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.

You can add your own report XML files by following the instructions in Chapter 6, "Using the Reporting Engine." Your reports can use either the default Velocity templates provided with Oracle Self-Service E-Billing or your own templates.

Changing the URL Prefix

Oracle Self-Service E-Billing uses ebilling as its URL prefix. However, you can change this to fit your deployment environment. The prefix is defined in the application.xml file in the ebilling-weblogic-10-6.0.4.ear/META-INF directory. All URLs from Oracle Self-Service E-Billing use a relative URL. This ensures that after you change the URL prefix, the application can still work.

The Oracle Self-Service E-Billing-related action classes and other Struts are defined as a (Struts) module called ebilling. Access all resources in the EAR file, including Struts actions, images, jsp pages, and so on, with this prefix, for example:

http://host:port/ebilling/report.do

Using Spring (XMA) Configuration Files

Oracle Self-Service E-Billing uses Spring to manage JavaBean creation and transactions. The configuration of Hibernate is also through Spring. These files are also called XMA configuration files in Oracle Self-Service E-Billing terms and exist in the *EDX_HOME*/`xma` directory (or the *EDX_HOME*`\xma` directory on Windows). In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing. These are the core configuration files of Oracle Self-Service E-Billing and you must not modify them unless instructed in this document.

Possible reasons to customize these files include:

- To enable Hibernate show_sql.
- To extend Hierarchy Management, such as adding a new link target type, reimplementing a hierarchy search interface such as IAssignedObjectProvider, inserting a new loader into the OLTPProductionLoader job, configuring a new event handler to handle hierarchy events, or configuring the hierarchy UI behavior. For more information about extending Hierarchy, see *Hierarchy Developer's Guide for Oracle Self-Service E-Billing*.
- To configure the batch report job, to send email, for example.

Using the OLTP Database

OLTP is the Oracle Self-Service E-Billing transactional database. Oracle Self-Service E-Billing expects access to product tables to go through the Oracle Self-Service E-Billing APIs. Do not change the existing product schema. However, you can add your own customization tables.

Using the OLAP Database

OLAP is the Oracle Self-Service E-Billing data warehouse. It is a non-transaction database used to save billing information and has no APIs for access; these tables are accessed directly through report XML files. For information on how to use report XML files to retrieve data from the OLAP database, check the report XML files used to generate various billing reports. These files are defined in the *EDX_HOME*/`config/rpt` directory (or the *EDX_HOME*`\config\rpt` directory on Windows), where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.

The OLAP database includes three kinds of tables:

- **OLTP-OLAP Synchronization-Related Tables.** Because OLAP is a non-transactional database, it has no (or very limited) UI transactional operations. However, it requires transaction data to report on, for example, hierarchy information. This information is synchronized from OLTP to OLAP at real time. The tables related to this operation are:
  - EDX_RPT_ACCOUNT_WSPACE
  - EDX_RPT_ACCOUNT_XREF (Not used)
  - EDX_RPT_CC_CHARGE_WSPACE
  - EDX_RPT_HIERARCHY_NODE_PERIOD
  - EDX_RPT_HIERARCHY_TREE_DIM
  - EDX_RPT_HIERARCHY_TYPE_DIM
  - EDX_RPT_HIERARCHY_XREF_DIM
  - EDX_RPT_USER_HIERARCHY_WSPACE (Not used)
  - EDX_RPT_USER_SERVICE_WSPACE (Not used)

  Do not customize these tables. Operations on these tables are read-only.
- **Dimensional Tables.** Except the ones described in this topic, the remaining dimensional tables are used to save dimensional data such as accounts, services, dates, periods, and so on. Also most all the dimensional tables have some flexible fields which are for customization. Use the flexible fields to hold your custom information instead of adding your own columns. You can also create new dimensional tables.
- **Fact Tables.** Fact tables are used to hold the fact information such as call details or summaries. You can add new columns to the fact tables if necessary or add your own fact tables.

Never make any changes that could break the backward compatibility of the DB schema, such as changing the column type or renaming a column or a table.

The Oracle Self-Service E-Billing screen JSPs pick the label from the property file using a unique key.

Data Dictionary

Oracle Self-Service E-Billing provides a Data Dictionary with details about the OLAP and OLTP database tables.

The Oracle Self-Service E-Billing Data Dictionary is available in both PDF and HTML formats and can be found in the following directories:

- `*EDX_HOME*/doc/api/datadictionary/html/E-Billing_oltpindex.html`
- `*EDX_HOME*/doc/api/datadictionary/html/E-Billing_olapindex.html`
- `*EDX_HOME*/doc/api/datadictionary/pdf/E-Billing_oltpindex.pdf`
- `*EDX_HOME*/doc/api/datadictionary/pdf/E-Billing_olapindex.pdf`

In the paths, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.

Repackaging EAR Files

Whenever you want to modify a JSP, add a new action class, or add an EJB, you must repackage the EAR file.

When repackaging the EAR file, make sure you do not remove existing components and only modify the components that are recommended as modifiable in this guide, such as JSP pages, CSS files, the app-resources.jar file, and so on.

Auditing Database Administration Activity

Oracle Self-Service E-Billing does not audit database administrator activity. However, to remain compliant with the Payment Card Industry Data Security Standard (PCI DSS), you must implement auditing functionality that documents each time an administrator logs in, creates new tables or attributes, deletes information including tables, attributes, or transaction details, or runs an external script against the Oracle Self-Service E-Billing database.

Localizing the User Interface

The Oracle Self-Service E-Billing user interface is preconfigured to use English only. You can add additional languages by copying and translating the required English-language resource bundle (property) files and configuring the additional files described in this topic. The localization process involves configuring and translating files for each of the Oracle Self-Service E-Billing applications: Billing and Payment, Command Center, and Customer Service Representative applications.

You can set one language as the default for your implementation. Each user can choose a different language from the interface, and that language becomes his or her preferred language and automatically appears each time that user logs in.

To localize the user interface

1. Oracle Self-Service E-Billing must be installed and deployed. Shut down the Billing and Payment, Command Center, and Customer Service Representative application servers.
2. For each new language, add a record in the EDX_SYS_LANG database table. For example, the following SQL statement adds traditional Spanish and sets it as the default language:

   ```
   insert into edx_sys_lang(id,code,name,is_default) values(1,'es_ES','Espanol',1);
   ```

   where:

   - *id* is the ID you want to use as the primary key for this language.
   - *code* is the language code. The format can be *language_country* (such as en_US, zh_CN, or ja_JP) or *language_country_variant*, such as en_US_Traditional_WIN. The *language* argument is a valid ISO-639 Language Code in two lower-case letters. The *country* argument is a valid ISO-3166 Country Code in two upper-case letters.

     The *variant* argument is a vendor- or browser-specific code, for example: WIN for Windows, MAC for Macintosh, and POSIX for POSIX. The *variant* argument can have two parts, separated by an underscore. For example, the code for Traditional Spanish on Windows is es_ES_Traditional_WIN.
   - *name* is the name of the language that appears when a user selects a language in the interface.
   - *is_default* indicates whether this language is to be the default (0 is No; 1 is Yes).
3. Create new resource bundle property files for each new language for all three Oracle Self-Service E-Billing applications: Billing and Payment, Command Center, and Customer Self-Service applications.

   a. Extract the app-resources-1.0-SNAPSHOT.jar file from the EAR files shown in the following table (use back slashes (\) on Windows):

Application Server	Oracle Self-Service E-Billing Application	Location of the Application Resource Bundle File
Oracle WebLogic	Billing and Payment	*EDX_HOME/*J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar
	Command Center	EDX_HOME/J2EEApps/commandcenter/weblogic/command-center-weblogic-10-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar
	Customer Service Representative	*EDX_HOME/*J2EEApps/csr/weblogic/csr-app-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar
IBM WebSphere	Billing and Payment	*EDX_HOME/*J2EEApps/ebilling/websphere/ebilling-websphere-6-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar

Application Server	Oracle Self-Service E-Billing Application	Location of the Application Resource Bundle File
	Command Center	`EDX_HOME/J2EEApps/commandcenter/websphere/command-center-websphere-6-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`
	Customer Service Representative	*EDX_HOME*/`J2EEApps/csr/websphere/csr-app-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`

b Make copies of the following language resource property files, one for each language and each Oracle Self-Service E-Billing application (3), appending the locale code (languageCode_Country) to the new file names. For example, for Spanish, the copy of the application resource messages file must be called ApplicationResourcesNew_es_ES.properties. For each application, the files are located under the corresponding JAR file path in the `com/edocs/application/resources` directory (or the `com\edocs\application\resources` directory on Windows). (Place all new and updated property files in the same directories as the English language files.)

NOTE: If you want to make any customizations to the pre-configured email content, make those changes in the following files before creating copies for localization.

- **ApplicationResourcesMessages.properties**. Message text, such as validation and error messages that appear in the user interface.
- **ApplicationResourcesNew.properties**. Text of tabs, labels, and titles that appear on the user interface Web pages.
- **NotificationResource.properties**. Contains text strings used to compose email notifications.
- **Period.properties**. Contains monthly time periods that appear in lists in the user interface.
- **CurrencyText.properties.** Contains the currency name to display in reports and charts for each language.

Oracle Self-Service E-Billing comes preconfigured with a set of language files for U.S. English:

- ApplicationResourcesMessages_en_US.properties
- ApplicationResourcesNew_en_US.properties
- CurrencyText_en_US.properties
- NotificationResource_en_US.properties
- Period_en_US.properties

4 Translate the appropriate content in each new property file.

NOTE: The content of the resource bundle files are identical for all applications.

5 In the CurrencyText.properties file, for each application and in each language, add mappings between the currency code and text for any currencies you want to use. The currency text appears on reports and charts.

The following values (for the American dollar, Chinese yuan, and euro) are included in the file by default:

```
USD.CurrencyText=Dollars

CNY.CurrencyText=CNY

EUR.CurrencyText=Euro
```

6 In the ApplicationResourcesNew.properties file, for each application and each language, update the file to customize the date, time, and number formats used in the user interface. Specifying custom values lets you override the default Java language formats. You can also add new patterns to the ApplicationResourcesNew.properties file.

To apply a different date format in a report (such as using the short date format instead of the medium format), update the report XML file with the pattern you prefer. The report XML files are found in the *EDX_HOME/*config/rpt directory. Also update the report XML files where you want to apply any new patterns. Update the following code in the ApplicationResourcesNew.properties files:

```
#################### Date,Time,Number ########################

global.pattern.number.integer=#,##0
global.pattern.number.decimal=#,##0.00
global.pattern.number.percent=#0.00%
global.pattern.number.amount=\u00A4#,##0.00
global.pattern.number.amount2=\u00A4#,##0.00;\u00A4(#,##0.00)
global.pattern.number.amount3=#,##0;(#,##0)
global.pattern.number.amount4=#,##0.00;(#,##0.00)
global.pattern.date.short=M/d/yy
global.pattern.date.medium=MM/dd/yyyy
global.pattern.date.long=MMM/dd/yyyy
global.pattern.date.input=MM/dd/yyyy
global.pattern.time.short=HH:mm
global.pattern.time.long=HH:mm:ss
global.pattern.date.time=MM/dd/yyyy HH:mm:ss
```

NOTE: The amount3 and amount4 patterns display numbers that are not currency amounts.

7 For each application, update the StatementDisplay.properties file to add the Unicode currency symbol for any currencies you want to use. The currency symbols appear in the user interface and in email notification content. The StatementDisplay.properties file is found under the application JAR file path in the com/edocs/application/resources directory (or the com\edocs\application\resources directory on Windows). If your billing files contain only a single currency, then add an entry for that currency.

The following values (for the American dollar, Chinese yuan, and euro), are included in the file by default:

```
USD.CurrencySymbol=$

CNY.CurrencySymbol=\u00a5
```

```
EUR.CurrencySymbol=\u20ac
```

8 If any of the new languages use special characters, update the Payment Module validation file, validation-payment.xml. Adding the characters to this file lets a user enter the special characters in payment account names, credit card names, and so on in the Payment Module. The validation-payment.xml file is located in the `/WEB-INF` directory under the `ebilling-web-1.0-SNAPSHOT.war` path. Extract the `ebilling-web-1.0-SNAPSHOT.war` file from the EAR files shown in the following table (use back slashes (\) on Windows):

Application Server	Oracle Self-Service E-Billing Application	Location of the Payment Validation File validation-payment.xml
Oracle WebLogic	Billing and Payment	*`EDX_HOME`*`/J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF`
IBM WebSphere	Billing and Payment	*`EDX_HOME`*`/J2EEApps/ebilling/websphere/ebilling-websphere-6-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF`

9 Repackage the JAR and EAR files at the application server console and deploy the EAR files. For instructions on how to deploy an application EAR file, see *Installation Guide for Oracle Self-Service E-Billing*.

10 Generate an email template XML file for each language:

a Modify the template generator script for your implementation. The template generator script generates an email notification template XML file based on the notification properties file for each language. Change to the *`EDX_HOME`*`/bin/notification/` directory (or the *`EDX_HOME`*`\bin\notification\` directory on Windows). In the generateEmailTemplate.sh file found in this directory (or the generateEmailTemplate.cmd file on Windows), update the value of EDX_HOME and using the full path names for your installation. If you have saved the Billing and Payment application ear file in a new location, update the path to the ear file in the EBearfile variable (the default is `EBearfile="$EDX_HOME/J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear"`).

b Run the email template generator utility. You can run this tool in one of two ways:

- **Generate all new language email templates in batch.** To generate a batch of XSL template files in the corresponding languages, create a text file that contains each language code on separate lines and place the file in the *`EDX_HOME`*`/bin/notification` directory (or the *`EDX_HOME`*`\bin\notification` directory on Windows). Run the following command, where *`filename`* is the name of the language code text file you created:

  - Unix: `./generateEmailTemplate.sh -f` *`filename`*
  - Windows: `generateEmailTemplate.cmd -f` *`filename`*

  For example, the following text file content generates two files called template_ zh_CN.xsl and template_ ja_JP.xsl:

  zh_CN

  ja_JP

- **Generate a single template for one new language.** You can generate a single XSL template file for a new language, run the utility as follows, where *code* is the language code:
  - UNIX: `./generateEmailTemplate.sh -l` *code*
  - Windows: `generateEmailTemplate.cmd -l` *code*

  For example, to generate a Chinese template file (called template_zh_CN.xsl), use the following command:
  - UNIX: `./generateEmailTemplate.sh -l zh_CN`
  - Windows: `generateEmailTemplate.cmd -l zh_CN`

  The new template files generated will be saved in the *EDX_HOME*/`config/notification/templates` directory.

11 For each new language, make copies of the following template files used for generating PDF reports, appending the locale code (languageCode_Country) to the new file names. The files can be found in the *EDX_HOME*/`template/pdf` directory (use back slashes (\) on Windows). Place the new files in the same directory.

- PrintSummary.rtf
- StatementSummary.rtf
- telco_std_r1.rtf
- telco_std_r6.rtf
- telco_std_r13.rtf

NOTE: Oracle Self-Service E-Billing comes preconfigured with a set of template files for American English (appended with the American English locale code, _en_US).

12 Restart the application servers.

CAUTION: If you want to make any customizations to email notification text after localization, make the changes to the notification property files for the language and regenerate the corresponding template file. Changes made directly to a template file will be lost if the template regeneration runs again for that language.

Debugging Oracle Self-Service E-Billing

Oracle Self-Service E-Billing produces various logging information for you to use to debug problems.

Oracle Self-Service E-Billing has three logging mechanisms:

- **Log4j.** Log4j is the main logging mechanism. Each EAR (application) requires different log4j files to avoid conflicting with each other. For more information about logs, see *Administration Guide for Oracle Self-Service E-Billing*.

  CAUTION: Because of security concerns, update the log4j_cc.xml file to write Command Center logs to the database, not to a file. There are no file appenders to Command Center logs. An *appender* is a named entity that represents a specific output destination for messages. It is technically valid to write the Oracle Self-Service E-Billing and CSR application logs to either the database or files as specified in the log4j.xml and log4j_csr.xml files.

- **DB-logging.** Most Command Center jobs also use DB-logging for job-level information and log4j is still used to log API-level information. The DB-logging writes log information into DB tables and can be viewed from the Command Center.
- **Java-option-logging.** The logging is controlled by pass-in a JVM -D option. This is usually used to log debug-level information and mostly for development purpose.

In addition, in the majority of use cases, Oracle Self-Service E-Billing prints out the exception stack trace to the console or as part of the JSP error output page when an exception occurs.

Viewing log4j Log Files

Each application (Billing and Payment, Command Center, and Customer Service Representative) maintains log files.

You can configure the log4j.xml and log4j_cc.xml files for the log level.

Billing and Payment Application Log Files

The Billing and Payment application maintains multiple log files:

- hierarchy.log
- reporting.log
- umf.log
- ebilling.log

See the log4j.xml file in the *EDX_HOME*/config/ directory for details. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.

Command Center Application Log Files

The Command Center application maintains the following log files:

- log4j_eStatement.log
- log4j_scheduler.log

See the log4j_cc.xml file in the *EDX_HOM*/config/ directory for details.

Customer Service Representative Application Log Files

The CSR application maintains the log4j_csr.xml file in the *EDX_HOME*/config directory.

Viewing Command Center Logs

The Command Center jobs use a combination of DB-logging and log4j to log information.

When there is a problem with a Command Center job, you can view the DB-logging for log4j logs. For more information about viewing Command Center message logs, see *Administration Guide for Oracle Self-Service E-Billing*.

If DB logging does not provide enough information, go to the log4j files described in "Viewing log4j Log Files" on page 29.

Displaying SQL Statements

One of the most useful debug features is to display the SQL statements issued to the database.

To view the Hibernate SQL statements

1. Open the persistence.xma.xml file for editing. This file is found in the *EDX_HOME*/xma/config/modules directory, where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.
2. Edit the hibernate.show.sql property, changing the value from false to true:

   ```
   <prop key=hibernate.show_sql>false</prop>
   ```

3. To be able to view the SQL binding values as well (the hibernate.show_sql property allows you to view the SQL statements only), edit the log4j files (log4j.xml, log4jcc.xml, and log4jccenter.xml) which are found in the *EDX_HOME*/config directory. Change the log level for these two loggers to debug:

   ```
   <logger name="org.hibernate.SQL" additivity="false">

   <level value="TRACE"/>

               <appender-ref ref="cba-log"/>

   </logger>


   <logger name="net.sf.hibernate.type" additivity="false">

    <level value="error"/>

               <appender-ref ref="cba-log"/>

   </logger>
   ```

4. These configurations apply to Hibernate-based DB access. One exception is reporting-related SQL statements, which are issued without using Hibernate. To view the report SQL statements and their binding values, add a Java -D option:

   ```
   java –Ddatasource.debug=true
   ```

Accessing Oracle Self-Service E-Billing Javadoc

Oracle Self-Service E-Billing API Javadoc is available in your product installation.

To access Oracle Self-Service E-Billing Javadoc

1. Unzip the apidoc.jar file found in the *EDX_HOME*/docs/api directory (or the *EDX_HOME*\docs\api directory on Windows).
2. Open the index.html file.

3 Customizing User Management

This chapter covers the public APIs available for customizing the Oracle Self-Service E-Billing user management functionality. It includes the following topics:

Customizing User Management and Security

User management involves managing users and security. Security involves authentication, authorization, encryption, and decryption. User management involves enrolling different users and managing their profiles and roles.

You manage users and their roles using two main classes:

- **IUserManager**. Use to add, delete and update users. User is represented with IUser object.
- **ISecurityProfileManager**. Use mainly to manage a user's password and roles. The sec profile is represented by ISecurityProfile and role is represented by ISecRole.

To get an instance of IUserManager and ISecurityProfileManager implementation classes, use the following code:

```
IUserManager _userMgr=UserFactory.getUserManager();

ISecurityProfileManager _secProfileMgr=UserFactory.getSecurityProfileManager();
```

In addition to these two primary managers, there is an API in the service layer called IUserService for managing high-level user-related functions. This API is driven by use cases. There are one or more methods for use cases in the application.

Customizing User Enrollment

You can customize the User Enrollment use case using XMA and APIs.

Configuring User Enrollment XMA

You can configure the IUserService JavaBean in the userService.xma.xml file, found in the *EDX_HOME/* xma/config/modules/services directory. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing. The IUserService JavaBean contents are as follows:

```
<bean id="IUserService"

        class="com.edocs.common.services.umf.UserService">

        <property name="userAccountDao">

            <ref local="userAccountDao"/>

        </property>

     <property name="userServiceAgreementDao">

            <ref local="userServiceAgreementDao"/>

        </property>

    </bean>
```

Using User Enrollment APIs

When customizing user enrollment, you can call the enrollB2BUser or enrollB2CUser APIs to enroll B2B or B2C users, for example:

```
IUserService usrService=EBillingServiceFactory.getUserService();

usrService.enrollB2BUser(c_user, role, user.getUserProfile().getEmail1() );
```

Create a JavaBean called com.edocs.common.api.services.IUserEnrollProfile:

```
IUserEnrollProfile enrollProf = new UserEnrollProfile();
```

Use the following code to set the properties:

```
enrollProf.setRole(role)

enrollProf.setCompanyId(companyId);
```

```
    enrollProf.setServiceAgreementExtKey(saExtKey);
    enrollProf.setAccountExtKeyList(acctExtKeyList);
    enrollProf.setAdminEmail(adminUser.getUserProfile().getEmail1());
```

Use the following code for enrolling a single B2B user:

```
    usrService.enrollB2BUser(newUser, enrollProf, audit);
```

Using Bulk Enrollment API

To use the bulk enrollment API, pass the input stream CSV file for bulk enrollment, the filename, import time, administrator user, and audit as shown in the following code. The method enrolls the users in the file and returns the success number.

```
    IUserService usrService = EBillingServiceFactory.getUserService();

    int succeedEnrolledUsersNum = usrService.enrollB2BUser(csvFile.getInputStream(),
    csvFile.getFileName(),importTime, adminUser, audit);Status OpenFixedClosed
```

Customizing End User and CSR User Passwords

You can customize the password rules for end user and CSR user passwords.

You can modify the strength of a password by customizing the regular expression rule in each form where the end user or CSR user enters a password in Oracle Self-Service E-Billing. You can specify different password validation rules for end user and CSR user passwords. The default password rules requires that the password have at least one capital letter, one lowercase letter, one number, and no spaces.

You can also change the minimum and maximum password lengths, though the minimum password length cannot be less than 7 as required by the Payment Card Industry Data Security Standard (PCI DSS).

You must use the same password validation rule each time an end user or a CSR user enters his or her password. Table 5 lists the form elements you must update in the validation.xml files for each type of password.

Table 5. End User and CSR User XML Form Elements for Validating the Password

Password Type	Use Case	XML Form Element to Update
End User	Enrollment	`<form name="setSecQuestionForm">`
	Forgot Password and Reset Password	`<form name="ResetPwdForm">`
	Manage Profile	`<form name="changePwdForm">`

Table 5. End User and CSR User XML Form Elements for Validating the Password

Password Type	Use Case	XML Form Element to Update
CSR User	CSR Enrollment	`<form name="csrAdministratorForm">`
	Manage CSR Profile	`<form name="updateSecQuestionForm">`
	CSR Forgot Password and Reset Password	`<form name="ResetPwdForm">`
	Reset an Expired CSR Password	`<form name="resetExpPwdForm">`

To customize end user or CSR user passwords

1 Edit the validation.xml files, found in the following directories.

End user passwords:

- **Oracle WebLogic.** *EDX_HOME*`\J2EEApps\ebilling\weblogic\ebilling-weblogic-10-6.0.4.ear\ebilling-web-1.0-SNAPSHOT.war\WEB-INF`
- **IBM WebSphere.** *EDX_HOME*`\J2EEApps\ebilling\websphere\ebilling-websphere-6-6.0.4.ear\ebilling-web-1.0-SNAPSHOT.war\WEB-INF`

CSR user passwords:

- **Oracle WebLogic.** *EDX_HOME*`\J2EEApps\csr\weblogic\csr-app-6.0.4.ear\csr-web-1.0-SNAPSHOT.war\WEB-INF`
- **IBM WebSphere.** *EDX_HOME*`\J2EEApps\csr\websphere\csr-app-6.0.4.ear\csr-web-1.0-SNAPSHOT.war\WEB-INF`

2 In the validation.xml file, modify the following regular expression that validates the password input in each form element for the type of password rule you are setting (end user or CSR user). See Table 5 on page 35 for a list of form elements to update for each type of password.

```
<constant>

  <constant-name>pwd</constant-name>

  <constant-value>^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])[^\s]*$</constant-value>

      </constant>
```

3 To customize the minimum and maximum password length, update the following code for each form element for the type of password rule you are setting (end user or CSR user):

```
<var>

  <var-name>minlength</var-name>

  <var-value>8</var-value>   <!-Password minimum length -->

</var>

<var>
```

```
  <var-name>maxlength</var-name>
  <var-value>24</var-value>  <!-Password maximum length -->
</var>
<var>
  <var-name>mask</var-name>
  <var-value>${pwd}</var-value>
</var>
```

Customizing the Administrator User Password

A database administrator can customize the password validation rule for the Command Center administrator using a regular expression.

To customize the administrator user password

1. Log on to the Oracle Self-Service E-Billing OLTP database instance using SQL*Plus (not as SYSDBA):

   `OLTP schema username/OLTP schema password@OLTP TNS name`

   where:

   - *OLTP schema username* is the name of the OLTP schema user.
   - *OLTP schema password* is the password of the OLTP schema user.
   - *OLTP TNS name* is the TNS name for the OLTP instance.

2. Enter the following command, where *param_password_rule* is the regular expression with the new password rule you want to implement:

   ```
   SQL>exec change_pwd_validate_rule(param_password_rule) ;
   ```

Customizing the CSR User Password Configuration

You can customize the configuration for Customer Service Representative user passwords.

Oracle Self-Service E-Billing forces a CSR user to change his or her password every 90 days, and the new password cannot be the same as any of the last 4 passwords used by the same user. You can change these values.

To customize the CSR user password configuration

- Edit the security.xma.xml file, found in the *EDX_HOME*\xma\config\modules\security directory. Modify the values in the PasswordManageRule JavaBean:

```
<bean id="PasswordManageRule"

  class="com.edocs.common.security.authenticate.PasswordManageRule"
scope="singleton">

  <property name="daysBeforeExpiration">

    <value>90</value> <!--User password will be expired in the given days after
created-->

  </property>

  <property name="minUniqueNumOfPwd">

    <value>4</value> <!--New password can not be the same as the last given
password used by the same user-->

  </property>

</bean>
```

Customizing User Security Questions

You can customize the security questions that appear when a user forgets his or her password. You can add, delete, or update security questions using the following methods in the IUserService API:

- `"public ISecurityQuestion addSecurityQuestionDef(String question, String application, String locale);`
- `"public List<ISecurityQuestion> getSecurityQuestionDefList(String application, String locale);`
- `"public void deleteSecurityQuestionDef(long securityRequestionId);`
- `"public ISecurityQuestion updateSecurityQuestionDef(long securityRequestionId, String newQuestion);`

The following code shows an example of these methods:

```
IUserService _usrService = EBillingServiceFactory.getUserService();

String question1 = "This is a security question 1";

long securityRequestionId = 5327;

//Add new question definition

ISecurityQuestion sq1 = _usrService.addSecurityQuestionDef (question1,null,null);

//Get all questions

List<ISecurityQuestion> sqlist = _usrService.getSecurityQuestionDefLis (null,null);
```

```
//Delete a question definition

_usrService.deleteSecurityQuestionDef(securityRequestionId);

//Update an existed question definition

ISecurityQuestion sq1 = _usrService.updateSecurityQuestionDef(securityRequestionId,
question1);
```

NOTE: English is the only language supported for security questions. The locale must be NULL.

Deactivating and Reactivating the Master Customer Service Representative Administrator User

You can deactivate and reactivate the master Customer Service Representative (CSR) administrator user when needed.

To deactivate or reactivate the default CSR administrator user

1. Log on to the Oracle Self-Service E-Billing OLTP instance using SQL*Plus (not as SYSDBA).
2. To deactivate the master CSR administrator user, run the following command:

   ```
   SQL>exec disable_default_csr_admin;
   ```

3. To reactivate the master CSR administrator user, run the following command:

   ```
   SQL>exec enable_default_csr_admin;
   ```

Customizing Enrollment Validation

You can configure the validation code generator and use the validation API to customize enrollment validation.

Configuring the Validation Code Generator

The characters in the verification code and its expiration time are configurable. You can configure the following constraints in the security.xma.xml file, found in the *EDX_HOME*\xma\config\modules\security directory (or the *EDX_HOME*/xma/config/modules/security directory on Windows). In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing. These constraints are found in the section for the IVCodeGenerator JavaBean:

- **length.** An integer that represents the length of the verification code. The default value is 7; must be equal to or larger than 7.
- **exclude.** A string that contains the characters that cannot appear in the verification code.
- **includeSpecial.** Whether the validation code can include special characters such as %$(). Value can be true or false.

- **minUppercase.** An integer, the validation code must contain at least this number of upper case letters.
- **minLowercase.** An integer, the validation code must contain at least this number of lower case letters.
- **minNumber.** An integer, the validation code must contain at least this number of digital characters.
- **expirationTime.** An integer and a unit (D means day, H means hour, M means minute); for example, 4H means the verification code expires after 4 hours.

Using Enrollment Validation API

The ISecurityProfileManager API provides the method checkValidationCode for validation code and security profile ID validation:

```
ISecurityProfileManager spManager = UserFactory.getSecurityProfileManager();

       secProfile = spManager.checkValidationCode(secProfileId, validationCode);

  If validate is expired, throw ValidationCodeExpireException


ISecurityXMAService provide method getValidationCode() to get a validation code.


LookupService lookUp = LookupServiceFactory.getInstance();

ISecurityXMAService securityXMAService = (ISecurityXMAService)
lookUp.getModule("security");

IValidationCode =
securityXMAService.createValidationCodeManager().getValidationCode();
```

Customizing Account Lockout

You can customize the maximum attempt thresholds in the Account Lockout use case.

Oracle Self-Service E-Billing locks a user account after a maximum number of attempts (5) to enter information during the following use cases:

- **Log In**. When an end user tries to log into the application.
- **Forgot and Reset Password.** When an end user tries to enter a user name, account number, or service number.
- **Forgot and Reset Password.** When an end user tries to enter a security question or security answer.

By default, each of these activities uses the same threshold. You can specify one new threshold for all three activities, or set a different threshold value for each activity.

To configure the maximum attempt thresholds

1. Edit the user.xma.xml file, found in the *EDX_HOME*`\xma\config\modules\umf` directory.
2. In the IUserManager JavaBean, three lockers are defined as properties under the tag, each associated with one action. All three lockers reference one locker, which means all three actions have the same threshold value (maximum number of attempts). You can specify one new value for the max_attempts property, or specify a different locker and configure the threshold for each activity:

```
<bean id="IUserManager"
        class="com.edocs.common.umf.core.UserManager">
        <property name="userManagerDao">
            <ref local="userManagerDao"/>
        </property>
        <property name="loginLocker">
        <ref local="locker"/>
        </property>
        <property name="forgotPwdAccountLocker">
        <ref local="locker"/>
        </property>
        <property name="forgotPwdSecQstLocker">
        <ref local="locker"/>
        </property>
</bean>
<bean id="locker" class="com.edocs.common.umf.core.Locker">
    <property name="max_attempts" value="5"/>
</bean>
```

Using APIs

In the action layer, you can call the hasActionThresholdReached method to judge whether an action reached the maximum attempt threshold:

```
IUserService usrService=EBillingServiceFactory.getUserService();
usrService. hasActionThresholdReached(user, action);
```

The method hasActionThresholdReached adds the specific number of times attempted by 1, then compares the number of attempts with the maximum threshold. If the threshold is reached, the method returns true; if the threshold is not reached, it returns false.

Customizing Reactivate Account Lockout

You can customize the Reactivate Account Lockout use case APIs.

CSR administrator user can reactivate a locked out account. In the action layer, you can call reactivateAccount API to reactivate an account:

```
IUserService usrService=EBillingServiceFactory.getUserService();

usrService. reactivateAccount(usrId, audit);
```

These APIs use the following parameters:

- **usrId**. User ID. The user's account will be reactivated.
- **audit**. Audit data of the reactivate user account action.

Customizing Profile Management

Oracle Self-Service E-Billing provides APIs for customizing profile management. The IUser API provides setUserProfile and getUserProfile to set and get IUserProfile. Use the IUserProfile API to manage a user’s profile. Also, setSecQuestionRespons(Set<ISecQuestionResponse> secQuesRes), and getSecQuestionRespons in IUser can set and get the security question response for a user.

About Deleting Users

When a user is deleted, Oracle Self-Service E-Billing marks the user as deleted, not removed. A user is marked as deleted by setting the active flag on the user to false. After the user is deleted by making user inactive, the same username cannot be used again to enroll a new user; user names are not recycled.

Customizing Acegi Security

Oracle Self-Service E-Billing uses the Acegi framework for authentication and authorization. Acegi provides hooks for single sign-on implementation.

DaoAuthenticationProvider from ACEGI is implemented for authentication. DaoAuthenticationProvider leverages a UserDetailsService in order to lookup the username, password and GrantedAuthority[]s. IBillingUserDetailsService extends UserDetailsService and the implementation class implements loadUserByUsername(String userId) method to provide authentication mechanism. This method returns IBillingUserDetails object.

Authorization in Oracle Self-Service E-Billing is based on roles and permissions. Each user is assigned a role. Authorization to access a particular resource is determined by the permissions for a user's role. These permissions are defined in the azpolicy file. Permission is represented by an object called EBillingPermission.

User roles are grouped to form high-level roles, called *azPolicyRoles,* and permissions are assigned to the azPolicyRoles.

Role mapping is defined in the `WEB-INF/classes/azcfg.properties` subdirectory, under the application directories for the EAR and WAR files.

The following examples show how to map azPolicyRoles:

- com.edocs.common.security.rolemappers.secrole.ALL_USERS=Admin,User, Payer,Manager, Subscriber,CSR,CSRAdministrator
- com.edocs.common.security.rolemappers.secrole.ADMIN=Admin
- com.edocs.common.security.rolemappers.secrole.MANAGER=Manager
- com.edocs.common.security.rolemappers.secrole.SUBSCRIBER=Subscriber
- com.edocs.common.security.rolemappers.secrole.CSR=CSR
- com.edocs.common.security.rolemappers.secrole.CSR_ADMIN=CSRAdministrator

Once user roles are mapped to azPolicyRoles, define permissions for resources in the azpolicy.xml file, found in the `/WEB-INF/classes/azcfg/policy` subdirectory, under the application directories for the EAR and WAR files.

Example of sample permission code:

```
    <permission>

<name>perm_company_tab</name>
<cpath>com.edocs.common.security.authorize.az.permissions.EBillingPermission</
cpath>

<rule>

            <name>admin</name>   <!--Defines the name of the rule -->

            <type>SecurityRole</type> <!-- Type of the rule -->

<values>ADMIN</values> <!-- The role(s) which can access this resource, comma
separated. Note it can be an alias defined in azcfg.properties -->

        </rule>

     </permission>
```

A permission called perm_company_tab is defined to control the company UI tab. This permission specifies that the company tab is accessible for the azPolicyRole ADMIN. ADMIN is mapped to a user role administrator, making company tab accessible for users whose role is administrator.

After permissions are defined, the code or resources that must be authorized are surrounded by a tag called <authz:authorize>.

This example shows you how to use this tag:

```
<authz:authorize  ifAnyGranted="perm_company_tab">

<li class="sts"><span ><a href="companyProfile.do" title="<bean:message
key="global.myAccount.subNavTab3"/>"><bean:message
key="global.myAccount.subNavTab3"/></a></span></li>

</authz:authorize>
```

You can have a list of permissions separated by commas in the ifAnyGanted attribute. If any of the permissions in the list are granted, the body of the tag is written.

The authz:authorize tag can have the following attributes:

- **ifAllGranted.** All the listed permissions must be granted for the tag to output its body.
- **ifAnyGranted.** Any of the listed permissions must be granted for the tag to output its body.
- **ifNotGranted.** None of the listed permissions must be granted for the tag to output its body.

Customizing Acegi Configuration

You can customize the Acegi configuration in the acegi-security.xml file, found in the *EDX_HOME/* config/acegi subdirectory (under the application directories for the EAR and WAR files).

The filter defined to handle HTTP form authentication, formAuthenticationProcessingFilter, uses AuthenticationProcessingFilter to process a log in form. This is the most common way to authenticate users. Form-based authentication is entirely compatible with the DAO and JAAS authentication providers.

The following code defines the filter that handles form authentication in the acegi-security.xml file:

```
<!-- Define filter to handle FORM authentication -->

<bean id="formAuthenticationProcessingFilter"

="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">

  <property name="filterProcessesUrl">

    <value>/j_acegi_security_check</value>

  </property>

  <property name="authenticationFailureUrl">

    <value>/login.do?login_error=1</value>

  </property>

  <property name="defaultTargetUrl">

    <value>/reportStart.do</value>

  </property>

  <property name="authenticationManager">

    <ref bean="authenticationManager" />

  </property>

</bean>
```

The configured AuthenticationManager processes each authentication request.

If authentication is successful, the resulting Authentication object is placed into the SecurityContextHolder and the browser is redirected to the defaultTargetUrl property. The default target URL is reportStart.do. You can customize the defaultTargetUrl property to a particular target URL (action, jsp, or html).

If authentication fails, AuthenticationException is placed into the HttpSession attribute indicated by AbstractProcessingFilter.ACEGI_SECURITY_LAST_EXCEPTION_KEY, which provides a reason on the error page displayed to the user. If authentication fails, the browser displays the URL in authenticationFailureUrl, which you can also customize.

Using or Simulating Single Sign-On

You must configure Oracle Self-Service E-Billing to use an external identity store. Follow the procedure for your implementation:

- **Single sign-on (SSO) system.** If you are using an external identity store and user authentication or user management module such as a an SSO or LDAP system, follow "To configure Oracle Self-Service E-Billing to use a single sign-on system."
- **Simulating a single sign-on system.** If you use an external store but use Oracle Self-Service E-Billing to authenticate the user (by sending the user name and password from a URL), follow "To configure Oracle Self-Service E-Billing to simulate a single sign-on system."

Configuring Oracle Self-Service E-Billing to use a Single Sign-on System

Follow this procedure to configure Oracle Self-Service E-Billing to use a single sign-on system.

To configure Oracle Self-Service E-Billing to use a single sign-on system

- Prepopulate the OLTP table data shown in Table 6 on page 45. You do not need to populate the PASSWORD column in the EDX_BSL_AUTH_SECPROFILE table.

Table 6. User Data for Single Sign-on

Database Table	Columns to Populate	Notes
EDX_BSL_UMF_USER	ID	Required; unique for all users.
	USERID	Required; unique for all users.
	SECURITYPROFILEID	References the PROFILEID column in the EDX_BSL_AUTH_SECPROFILE table. Required; unique for all users.
	COMPANYID	Required for B2B.
	EMAIL1	Required.
	DATE_CREATED	Required.
	DATE_MODIFIED	Required.

Table 6. User Data for Single Sign-on

Database Table	Columns to Populate	Notes
	VERSION	Required; the default is 0.
EDX_BSL_AUTH _SECPROFILE	PROFILEID	Required; unique for all users.
	USERID	Required; unique for all users.
	STATUS	Required for the activity user; the default is 2.
	PASSWORD	Required if you are simulating single-sign, or using an external identity store but using Oracle Self-Service E-Billing authentication. The password can have any value.
	ISLOCKED	Required for the activity user; the default is 0.
	SECURE_SUBKEY_ID	Data type is NUMBER(19,0). This column references the ID column in the EDX_SECURE_SUBKEY table. Required; the default is 1.
EDX_BSL_SEC_PROF _ROLES_LINK	PROFILE_ID	References the PROFILEID column in the EDX_BSL_AUTH_SECPROFILE table. Required; unique for all users.
	ROLE_ID	References the ID column in the EDX_BSL_AUTH_SECROLE table. Required.
EDX_UMF_USER_ACCT _LINK	USERID	References the USERID column in the EDX_BSL_UMF_USER table. Required for a B2C user.
	ACCOUNT_KEY	References the NODEID in the EDX_BSL_AMF_BACCOUNT table. Required for a B2C user.
EDX_UMF_USER_ACCT _LINK (OLAP)	USERID	Required for B2C reporting.
	ACCOUNT_KEY	References the ACCOUNT_KEY column in the edx_rpt_account_dim table. Required for B2C reporting.

Configuring Oracle Self-Service E-Billing to Simulate a Single Sign-on System

Follow this procedure to simulate a single sign-on system if you use an external identity store but use Oracle Self-Service E-Billing to authenticate users.

To configure Oracle Self-Service E-Billing to simulate a single sign-on system

1. Populate the OLTP database with the data shown in Table 6 on page 45.

   The PASSWORD column in the EDX_BSL_AUTH_SECPROFILE is required and must have an appropriate encrypted value. You can get the decrypted password from the external identity store. Use the following APIs for password encryption and to update the subkey ID in the user security profile table:

   ```
   LookupService lookUp = LookupServiceFactory.getInstance();

   ICryptography cryptography = (ICryptography) lookUp.getModule("cryptography");

   String hashPassword = cryptography.hash(plainPassword);

   ISubkeyCrypto  SubkeyCrypto =
   EBillingServiceFactory.getSubkeyCryptoService().createSubkeyCrypto(null);

   String enPassword = SubkeyCrypto.encrypt(hashPassword);

   securityProfile.setSubkeyId(SubkeyCrypto.getSecureSubkeyID());
   ```

2. Log in to the Billing and Payment application using the following URL and credentials (password and user name). The j_password parameter can be plain text or encrypted by the cryptography module when sending the log in request URL:

   ```
   http://ServerName:PortNum/ebilling/
   j_acegi_security_check?j_username=Username&j_password=Password
   ```

   where:

   - *Username* is the name of the user you are impersonating.
   - *Password* is the password of the user you are impersonating.

4 Customizing Billing Statements

This chapter covers APIs you can use to customize online billing statements. It includes the following topics:

The online statement feature reduces operational costs when subscribers adopt online statements instead of a printed one. Oracle Self-Service E-Billling can render a statement that looks similar to the paper statement. Taking advantage of the dynamic of the Web, Oracle Self-Service E-Billling can expand or collapse the amount of data displayed, drill down into details, show a more up-to-date statement, and display a previous statement.

About Statement Presentment APIs

Oracle Self-Service E-Billling provides the following Statement Presentment API:

```
List<IStatement> getStatements(String billerId, String accountNumber, int maxCount,
Date fromTime, Date toTime);. getStatements retrieves a list of IStatement objects
for an account.
```

The list of statements returned have a statement date between the fromTime and toTime date. The maxCount is to limit to the number of statements returned where there are a lot of statements.

About Split Billing Rules Management APIs

The split-billing rules management feature enables providers to categorize business and personal mobile phone usage and expenses included on the same statement. Service providers define the split billing rules, and during the data load, Oracle Self-Service E-Billling uses these rules to categorize the expenses.

There are two ways to categorize mobile phone usage:

- All call records are automatically categorized at ETL time, based on the predefined rules. The rule ID is recorded in the EDX_RPT_SERVICE_DETAIL_FACT table for each service detail entry, so Oracle Self-Service E-Billling can track which split billing rule was applied when categorizing the service details.

- Users can manually change the categorization when viewing their statements online if any of the automatic categorization is not appropriate.

About the OLAP.EDX_RPT_ETL_PLUG_RULE Table

This table is where all the categorization rules used for split billing were specified. Upon ETL processing, the rule execution loops through all the records in this table, following the path pointing to an individual rule specified in the RULE_STORED_PROC column, to find the corresponding stored procedure and execute it.

Rules are executed in the order specified in the RULE_EXEC_ORDER column.

If you have additional rules to execute, you can develop a customized store procedure. Insert a new rule entry in the table, and the Oracle Self-Service E-Billling ETL process will apply the rule automatically.

A special entry (with RULE_TYPE='MAN') in this table represents a manual categorization activity.

When a user recategorizes certain service detail records in the application, the rule ID of the special entry will be recorded in EDX_RPT_SERVICE_DETAIL_FACT table, indicating that the user has manually categorized the service detail and the manual rule ID overwrites the previous split billing rule ID.

About the OLAP.EDX_RPT_SPLIT_CATEGORY_TYPE Table

This table records the category type used in split billing. Currently the category is either business or personal, (CORPORATE CALLS/PERSONAL CALLS). Choose one to be the default category using the CATEGORY_DEFAULT_CODE column, which represents a Boolean flag, indicating the default category type to be used when ambiguity occurs during the categorization process.

The last split billing rule (stored procedure) defined in EDX_RPT_ETL_PLUG_RULE categorizes all of the remaining uncategorized service detail records using the default category specified in this table.

If a customer has more categories to be used in the split-billing feature, a new entry can be added into the table.

Oracle Self-Service E-Billling provides the following Split Billing Rules Management APIs:

- **ISplitBillingService.** The service API for split billing feature, used to retrieve a particular service detail fact record, and change the category on a particular transaction detail. It can also be used to retrieve the list of valid categories defined for categorizing transaction detail records. (API is in package com.edocs.common.api.statement.splitbilling.ISplitBillingService)
- **IServiceTransactionDetail.** Represents a single one single service detail fact record in EDX_RPT_SERVICE_DETAIL_FACT table. (API is in package com.edocs.common.api.statement.IServiceTransactionDetail)

Transaction Dispute APIs

Oracle Self-Service E-Billling provides the following Transaction Dispute APIs:

- DisputeManagerFactory
- IDispute
- IDisputeManager
- IDisputeManagerFactory
- IDisputeReason
- IDisputeService
- IServiceAgrDetail
- IServiceAgrDetailService

Unbilled Usage APIs

Oracle Self-Service E-Billling provides the following Unbilled Usage APIs:

- IUnBilledActivity
- IUnBilledActivityFilterCriteria
- IUnBilledActivityManager
- IUnbilledActivityService
- UnBilledActivityManagerFactory

Contact APIs

Oracle Self-Service E-Billling provides the following Contact APIs:

- ContactManagerFactory
- ICCLManager
- ICCLService
- IContact
- IContactXMAService
- IContactsRetrievalFilter
- ICorporateContact
- IPCLManager
- IPCLService
- IPersonalContact
- IUserServiceAgreement

e

5 Using and Customizing Email Notifications

This chapter covers customizing the email notification feature in Oracle Self-Service E-Billling. It includes the following topics:

Configuring an Email Host and Other Messaging Properties

You must configure the properties that control email message delivery for your organization to specify the name of your email host and an SMTP host as part of the installation process. For details about setting these properties, see *Installation Guide for Oracle Self-Service E-Billing*.

You can also configure the following optional email-specific properties as well:

- **Global notification settings.** Settings for each notification type, which indicate whether to send or suppress the notification type globally (to all applicable users) or to allow individual users to choose whether to receive the email notification.
- **Maximum email queue threads.** The maximum number of email threads to create when sending email. The default is 10 threads.
- **Maximum queue elements per thread.** Email messages are sent in batches, by thread. The maximum number of messages that each thread must send in each batch. The default is 30 messages.
- **Queue dispatcher sleep period.** The time period, in seconds, that the dispatcher must sleep between sending email, to allow other threads to complete sends before removing queued messages. The default is 5 seconds.
- **Queue hanging timeout period.** The time period, in seconds, that the dispatcher must wait before deciding the email host is not responding and queue messages. The default is 15 seconds.
- **Queue storage directory.** The directory, located under the *EDX_HOME*`\config\` directory (or the EDX_HOME`\config\` directory on Windows), used to temporarily store undeliverable email. The default value is mailqueue.

To configure an email host and other messaging properties

1 Modify the properties in the notification.xma.xml file, which can be found in the *EDX_HOME*/xma/config/com/edocs/common/notification directory (or the *EDX_HOME*\config\notification directory on Windows). In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billling.

2 To specify global settings for each notification type valid in your edition of Oracle Self-Service E-Billling (Business or Consumer), specify one of the following values under the globalNotificationConfigMap property:

- **True.** All applicable users receive the email type; individual users cannot set a preference.
- **False.** The email notification type is not generated globally; individual users cannot set a preference.
- **notSet.** No global setting is specified; individual users can set their own preference.

For example, the following code shows where you specify the global setting for the bill-ready notification type:

```
- <property name="globalNotificationConfigMap">

- <map merge="default">

- <!--  Key/value for Bill Ready Notification
```

Only those settings for notification types valid in your edition are recognized. For information about valid email notification types in the Business Edition, see "Email Notification Template Content (Business Edition)" on page 59. For information about valid email notification types in the Consumer Edition, see "Email Notification Template Content (Consumer Edition)" on page 70.

3 To update any of the following email notification properties, specify the value under the corresponding property in the notification.xma.xml file:

Email Notification Property Function	Property Name in the notification.xma.xml File
Maximum email queue threads	mailQueueThreadMax
Maximum queue elements per thread	mailQueueElementsPerThread
Queue dispatcher sleep period	mailQueueDispatcherSleepPeriod
Queue hanging timeout period	mailQueueHangingTimeout
Queue storage directory	mailQueueStorageDirectory

Customizing the Content of Email Notifications

Oracle Self-Service E-Billling provides email notification templates for each type of message it supports. The content for each type of notification is described in "Email Notification Template Content (Business Edition)" on page 59 and "Email Notification Template Content (Consumer Edition)" on page 70. You can customize some of the notification content for your organization by updating the notification properties file contained in the application resource bundles and regenerating the email template XSL files using the automated template generator. The notification properties file contains the strings and text used in the composition of email messages.

Oracle Self-Service E-Billling can send email notifications to users based on enrollment events as well as various billing and payment lifecycle events, such as when a new bill is ready for viewing online, a payment is due, scheduled, sent, and so on. Email notifications are classified into batch and instant notifications, based on whether the notification is generated by a batch job defined and run in the Command Center or a user's action.

NOTE: Note that validation URLs are inserted automatically and cannot be changed; users are required to complete the enrollment process.

If you have localized your Oracle Self-Service E-Billling applications, be sure to update the corresponding notification property file for each language (if appropriate). For more information about localization, see "Localizing the User Interface" on page 23.

To customize the content of email notifications

1. Oracle Self-Service E-Billling must be installed and deployed. Shut down the Billing and Payment, Command Center, and Customer Service Representative application servers (if running).
2. Extract the app-resources-1.0-SNAPSHOT.jar file from the Billing and Payment and the Customer Service Representative EAR files shown in the following table (use back slashes (\) on Windows):

Application Server	Oracle Self-Service E-Billing Application	Location of the Application Resource Bundle File
Oracle WebLogic	Billing and Payment	*EDX_HOME*/`J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`
	Command Center	*EDX_HOME*/`J2EEApps/commandcenter/weblogic/command-center-weblogic-10-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`
	Customer Service Representative	*EDX_HOME*/`J2EEApps/csr/weblogic/csr-app-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`

Application Server	Oracle Self-Service E-Billing Application	Location of the Application Resource Bundle File
IBM WebSphere	Billing and Payment	*EDX_HOME/*`J2EEApps/ebilling/websphere/ebilling-websphere-6-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`
	Command Center	*EDX_HOME/*`J2EEApps/commandcenter/websphere/command-center-websphere-6-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`
	Customer Service Representative	*EDX_HOME/*`J2EEApps/csr/websphere/csr-app-6.0.4.ear/xma/app-resources-1.0-SNAPSHOT.jar`

3 Edit the NotificationResource_en_US.properties file with your customizations, located under the corresponding application JAR path in the `com/edocs/application/resources` directory (or the `com\edocs\application\resources` directory on Windows).

4 Repackage the JAR and EAR files at the application server console and deploy the EAR files. For instructions on how to deploy an application EAR file, see *Installation Guide for Oracle Self-Service E-Billing*.

5 Generate a new email template XML file:

   a Modify the template generator script for your implementation. The template generator script generates an email notification template XML file based on the notification properties file. Change to the *EDX_HOME/*`bin/notification` directory (or the *EDX_HOME*`\bin\notification` directory on Windows). In the generateEmailTemplate.sh file script found in this directory (or the enerateEmailTemplate.cmd file on Windows), update the value of EDX_HOME and using the full path names for your installation. If you have saved the Billing and Payment application ear file in a new location, update the path to the ear file in the EBearfile variable(the default is `EBearfile="$EDX_HOME/J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear"`).

   b Run the template generator script to regenerate the template_en_US.xsl file:

   UNIX: `./generateEmailTemplate.sh -l en_US`

   Windows: `generateEmailTemplate.cmd -l en_US`

6 Restart the application servers.

Customizing the Default Display Patterns Used in Email Notifications

You can change the default display patterns Oracle Self-Service E-Billling uses to display the following data in email notifications:

- Date

- Time
- Date and time
- Integers
- Decimals (double)
- Amounts

The default notification display patterns are defined in the notification.xma.xml file. The valid display patterns are defined in the ApplicationResourcesNew.properties file, as shown:

```
################### Date,Time,Number ########################

global.pattern.number.integer=#,##0
global.pattern.number.decimal=#,##0.00
global.pattern.number.percent=#0.00%
global.pattern.number.amount=\u00A4#,##0.00
global.pattern.number.amount2=\u00A4#,##0.00;\u00A4(#,##0.00)
global.pattern.number.amount3=#,##0;(#,##0)
global.pattern.number.amount4=#,##0.00;(#,##0.00)
global.pattern.date.short=M/d/yy
global.pattern.date.medium=MM/dd/yyyy
global.pattern.date.long=MMM/dd/yyyy
global.pattern.date.input=MM/dd/yyyy
global.pattern.time.short=HH:mm
global.pattern.time.long=HH:mm:ss
global.pattern.date.time=MM/dd/yyyy HH:mm:ss
```

The defaults set in the notification.xma.xml are used in all three Oracle Self-Service E-Billling applications: Billing and Payment, Customer Service Representative, and the Command Center. For example, you can change the default date pattern from global.pattern.date.medium (M/d/yy), to global.pattern.date.long (MMM/dd/yyyy).

To customize the default display patterns used in email notifications

- In the *EDX_HOME*`\xma\config\com\edocs\common\notification\notification.xma.xml` file (use back slashes (\) on Windows), edit the appropriate values in the patternConfig property in the `<bean id="GlobalConfigurationBean"` JavaBean. Specify one of the display patterns defined in the ApplicationResourcesNew.properties file:

```
- <property name="patternConfig">

- <map merge="default">

- <entry key="defaultDatePattern">

  <value>global.pattern.date.medium</value>

  </entry>

- <entry key="defaultTimePattern">

  <value>global.pattern.time.long</value>

  </entry>
```

```
- <entry key="defaultDateTimePattern">
  <value>global.pattern.date.time</value>
  </entry>
- <entry key="defaultIntPattern">
  <value>global.pattern.number.integer</value>
  </entry>
- <entry key="defaultDoublePattern">
  <value>global.pattern.number.decimal</value>
  </entry>
- <entry key="defaultAmountPattern">
  <value>global.pattern.number.amount</value>
  </entry>
  </map>
  </property>
```

Email Notification Template Content (Business Edition)

Table 7 describes the template text provided for automated email notifications in Oracle Self-Service E-Billling (Business edition). For the Oracle Self-Service E-Billling (Consumer edition) notification text, see "Email Notification Template Content (Consumer Edition)" on page 70.

For instructions on configuring a Notifier job, see *Administration Guide for Oracle Self-Service E-Billing*.

Table 7. Notification Types and Email Templates (Business Edition)

Notification Type	Email Template (Business Edition)
Bill Ready	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have a New Bill Dear *$User*, The following accounts registered under your email address have new e-Bills: Account Number: *$Account_Number* Service Number: *$Service_Number* of Account *$Account_Number* You have additional bills; only *$Cutoff_Number* accounts have been displayed above. To view your e-bills, click *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Enrollment (B2B User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome! Your administrator has enrolled you in the Customer Care Application. Contact your company administrator at *$Admin_Email* to receive your assigned username. Then visit *$Secure_Billing_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Enrollment (CSR User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome! Your administrator has enrolled you in the Customer Care Application. Contact your company administrator at *$Admin_Email* to receive your assigned username. Then visit *$Secure_Billing_CSR_URL* to finish the enrollment process. Log in now and you will be redirected to create your own personal password. This is an automatically generated email. Please do not reply to this message.
Enrollment (CSR Administrator)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome to the Customer Care Application. Now you can start managing other customer service representatives and assisting customers with e-billing. Please visit *$Secure_Billing_CSR_URL* to finish the enrollment process. Log in now and you will be redirected to create your own personal password. This is an automatically generated email. Please do not reply to this message.
Enrollment (Organization Administrator)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome! Your administrator has enrolled you in the Customer Care Application. Contact your company customer service representative at *$Admin_Email* to receive your assigned username. Then visit *$Secure_Billing_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Enrollment (Migrated B2B User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome to the Customer Care Application. Please visit *$Secure_Billing_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.
Enrollment (Migrated CSR User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome to the Customer Care application. Please visit *$Secure_Billing_CSR_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.
Job Alert Success	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Job Completed Successfully Dear *$User*, The following job completed successfully: *Job_Name Job_Type$ Job_Instance_ID $DDN*
Job Alert Failure	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Job Failure Dear *$User*, The following job did not complete successfully: *Job_Name Job_Type$ Job_Instance_ID $DDN $Exception*

Notification Type	Email Template (Business Edition)
Recurring Payment Confirmation	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Welcome to Automatic Bill Payment Dear *$User*, You have successfully enrolled in the Automatic Bill Payment program for your online accounts. The first automatic bill payment will take place after your next billing cycle. You will receive a payment confirmation email when the payment is submitted. Account Number: *$Account_Number* Payment Type: *$Payment_Type* Amount: *$Amount* You can continue to make other payments on your account using the Quick Payment option on the Payments Menu. Click *$Billing_URL* to view your e-bills anytime. Thank you for using Automatic Bill Payment. This is an automatically generated email. Please do not reply to this message.
Recurring Payment Configuration Update	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Automatic Bill Payment Settings Have Been Updated Dear *$User*, The Automatic Bill Payment settings for your online accounts have been updated. Account Number: *$Account_Number* Payment Type: *$Payment_Type* Amount Due: *$Amount_Due* Thank you for using Automatic Bill Payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Recurring Payment Delete	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Automatic Bill Payment Participation Cancelled Dear *$User*, This notification confirms that your participation in Automatic Bill Payment has been cancelled. Account Number: *$Account_Number* Payment Type: *$Payment_Type* Amount: *$Amount* Please continue visiting *$Billing_URL*. Thank you for using Automatic Bill Payment. This is an automatically generated email. Please do not reply to this message.
Successful Quick Payment	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Thank You for Your Quick Payment Dear *$User*, Thank you for submitting the following one-time Quick Payment: Account Number: *$Account_Number* Amount: *$Amount* Master Reference Number: *$Reference_Number* Thank you for using Quick Payment. Visit us again to make your next online payment at *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Quick Payment Failure	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Quick Payment Failed A problem occurred during your one-time Quick Payment transaction. The following payment did not process successfully: Amount: *$Amount* Master Reference Number: *$Reference_Number* Please visit *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Payment Due	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have a Payment Due Dear *$User*, You have a payment due for each of the following accounts: Account: *$Account* Due Date: *$Due_Date* Due Amount: *$Amount_Due* You have additional bills; only *$Cutoff_Number* accounts have been displayed above. Thank you for using online payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Payment Scheduled	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Automatic Payment Schedule Confirmed Dear *$User*, Automatic payments have been scheduled for the following accounts: Account: *$Number* Due Date: *$Due_Date* Total Amount Due: *$Due_Amount* Amount Paid: *$Configured_Amount* Outstanding Balance: *$Balance_Due* *$Statement_Credit* has been credited to your account. Thank you for using online payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Successful Payment	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Thank You for Your Payment Dear *$User*, Your payments for the following accounts completed successfully: Account: *$Number* Amount Due: *$Due_Amount* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Payment Failure	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Payment Did Not Complete Successfully Dear *$User*, Your online payment for the following accounts has failed: Account: *$Number* Amount Due: *$Due_Amount* Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Statement Threshold	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** 1st Notice - Your Payment Threshold Exceed Dear *$User*, Your most recent statement has exceeded the payment threshold you have set for yourself. Please log into your self-service application at *$Secure_Billing_URL* and make the necessary changes. Thank You
Payment Threshold	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Automatic Payment Threshold Has Been Reached Dear *$User*, The amount due exceeds the threshold you set for automated payment for the following accounts: Account: *$Number* Amount Due: *$Due_Amount* Configured Amount: *$Configured_Amount* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Credit Card Expiration	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Credit Card is About to Expire Dear *$User*, The following credit card account used for online payments is about to expire: Credit Card Number: *$Credit_Card_Number* Expiration Date: *$Expiration_Date* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.
Payment Account Create	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Payment Account Created Successfully Dear *$User*, The following payment account has been created successfully: Account Name: *$Account_Name* Thank you for using online payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Payment Account Update	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Payment Account Updated Successfully The following payment account below has been updated successfully: Account Name: *$Account_Name* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Payment Account Delete	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Payment Account Deleted Successfully Dear *$User*, The following payment account has been deleted successfully: Account Name: *$Account_Name* Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Batch Report Ready	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Batch Reporting Status Dear *$User*, This notification is to inform you of the status of your batch report: Report Name: *$Report_Name* Create Date: *$Create_Date* Start Date: *$Start_Date* End Date: *$End_Date* Status: *$Status* Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Login Password Changed	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Login Password Has Been Changed Dear *$User*, Your login password has been changed; you can use the new password to login. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
End User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated. You can now continue managing and paying your bills online using the online Customer Care Application. Why wait! Please visit *$Secure_Billing_URL* to finish the reactivation process. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.
CSR User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated; you can once again access your online Customer Care Application. Please visit *$Secure_Billing_CSR_URL* to finish the reactivation process. This is an automatically generated email. Please do not reply to this message.
Migrated End-User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated. You can now continue managing and paying your bills online using the online Customer Care Application. Why wait! Please visit *$Secure_Billing_URL* to finish the reactivation process. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Business Edition)
Migrated CSR User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated; you can once again access your online Customer Care Application. Please visit *$Secure_Billing_CSR_URL* to finish the reactivation process. This is an automatically generated email. Please do not reply to this message.
Password Expired	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Administrator Password is Expiring Dear *$User*, Your account password will expire in *$Number_Days* days. Please set a new password. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Email Notification Template Content (Consumer Edition)

This topic shows the content of the email templates provided for each type of notification in Oracle Self-Service E-Billling (Consumer edition). Table 8 shows the template content for each notification type. For the Oracle Self-Service E-Billling (Business edition) notification text, see "Email Notification Template Content (Business Edition)" on page 59.

Table 8. Notification Types and Email Templates (Consumer Edition)

Notification Type	Email Template (Consumer Edition)
Bill Ready	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have a New Bill Dear *$User*, The following accounts registered under your email address have new e-Bills: Account Number: *$Account_Number* Service Number: *$Service_Number* of Account *$Account_Number* You have additional bills; only *$Cutoff_Number* accounts have been displayed above. To view your e-bills, click *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Enrollment (B2C User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome! You have successfully enrolled in the Customer Care Application. You can now start managing and paying your bills online. Why wait! Please visit *$Secure_Billing_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Enrollment (CSR User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome! Your administrator has enrolled you in the Customer Care Application. Contact your company administrator at *$Admin_Email* to receive your assigned username. Then visit *$Secure_Billing_CSR_URL* to finish the enrollment process. Log in now and you will be redirected to create your own personal password. This is an automatically generated email. Please do not reply to this message.
Enrollment (CSR Administrator)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome to the Customer Care Application. Now you can start managing other customer service representatives and assisting customers with e-billing. Please visit *$Secure_Billing_CSR_URL* to finish the enrollment process. Log in now and you will be redirected to create your own personal password. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Enrollment (Migrated B2C User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome to the Customer Care application. You can now start managing and paying your bills online. Why wait! Please visit *$Secure_Billing_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.
Enrollment (Migrated CSR User)	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have Been Enrolled Dear *$User*, Welcome to the Customer Care application. Please visit *$Secure_Billing_CSR_URL* to finish the enrollment process. This is an automatically generated email. Please do not reply to this message.
Job Alert Success	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Job Completed Successfully Dear *$User*, The following job completed successfully: *Job_Name Job_Type$ Job_Instance_ID $DDN*
Job Alert Failure	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Job Failure Dear *$User*, The following job did not complete successfully: *Job_Name Job_Type$ Job_Instance_ID $DDN $Exception*

Notification Type	Email Template (Consumer Edition)
Recurring Payment Confirmation	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Welcome to Automatic Bill Payment Dear *$User*, You have successfully enrolled in the Automatic Bill Payment program for your online accounts. The first automatic bill payment will take place after your next billing cycle. You will receive a payment confirmation email when the payment is submitted. Account Number: *$Account_Number* Payment Type: *$Payment_Type* Amount: *$Amount* You can continue to make other payments on your account using the Quick Payment option on the Payments Menu. Click *$Billing_URL* to view your e-bills anytime. Thank you for using Automatic Bill Payment. This is an automatically generated email. Please do not reply to this message.
Recurring Payment Configuration Update	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Automatic Bill Payment Settings Have Been Updated Dear *$User*, The Automatic Bill Payment settings for your online accounts have been updated. Account Number: *$Account_Number* Payment Type: *$Payment_Type* Amount Due: *$Amount_Due* Thank you for using Automatic Bill Payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Recurring Payment Delete	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Automatic Bill Payment Participation Cancelled Dear *$User*, This notification confirms that your participation in Automatic Bill Payment has been cancelled. Account Number: *$Account_Number* Payment Type: *$Payment_Type* Amount: *$Amount* Please continue visiting *$Billing_URL*. Thank you for using Automatic Bill Payment. This is an automatically generated email. Please do not reply to this message.
Successful Quick Payment	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Thank You for Your Quick Payment Dear *$User*, Thank you for submitting the following one-time Quick Payment: Account Number: *$Account_Number* Amount: *$Amount* Master Reference Number: *$Reference_Number* Thank you for using Quick Payment. Visit us again to make your next online payment at *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Quick Payment Failure	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Quick Payment Failed A problem occurred during your one-time Quick Payment transaction. The following payment did not process successfully: Amount: *$Amount* Master Reference Number: *$Reference_Number* Please visit *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Payment Due	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** You Have a Payment Due Dear *$User*, You have a payment due for each of the following accounts: Account: *$Account* Due Date: *$Due_Date* Due Amount: *$Amount_Due* You have additional bills; only *$Cutoff_Number* accounts have been displayed above. Thank you for using online payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Payment Scheduled	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Automatic Payment Schedule Confirmed Dear *$User*, Automatic payments have been scheduled for the following accounts: Account: *$Number* Due Date: *$Due_Date* Total Amount Due: *$Due_Amount* Amount Paid: *$Configured_Amount* Outstanding Balance: *$Balance_Due* *$Statement_Credit* has been credited to your account. Thank you for using online payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Payment Success	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Thank You for Your Payment Dear *$User*, Your payments for the following accounts completed successfully: Account: *$Number* Amount Due: *$Due_Amount* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Payment Failure	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Payment Did Not Complete Successfully Dear *$User*, Your online payment for the following accounts has failed: Account: *$Number* Amount Due: *$Due_Amount* Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Statement Threshold	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** 1st Notice - Your Payment Threshold Exceed Dear *$User*, Your most recent statement has exceeded the payment threshold you have set for yourself. Please log into your self-service application at *$Secure_Billing_URL* and make the necessary changes. Thank You
Payment Threshold	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Automatic Payment Threshold Has Been Reached Dear *$User*, The amount due exceeds the threshold you set for automated payment for the following accounts: Account: *$Number* Amount Due: *$Due_Amount* Configured Amount: *$Configured_Amount* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Credit Card Expiration	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Credit Card is About to Expire Dear *$User*, The following credit card account used for online payments is about to expire: Credit Card Number: *$Credit_Card_Number* Expiration Date: *$Expiration_Date* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.
Payment Account Create	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Payment Account Created Successfully Dear *$User*, The following payment account has been created successfully: Account Name: *$Account_Name* Thank you for using online payment. Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Payment Account Update	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Payment Account Updated Successfully The following payment account below has been updated successfully: Account Name: *$Account_Name* Please continue visiting *$Billing_URL*. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Payment Account Delete	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Payment Account Deleted Successfully Dear *$User*, The following payment account has been deleted successfully: Account Name: *$Account_Name* Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Batch Report Ready	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Batch Reporting Status Dear *$User*, This notification is to inform you of the status of your batch report: Report Name: *$Report_Name* Create Date: *$Create_Date* Start Date: *$Start_Date* End Date: *$End_Date* Status: *$Status* Please continue visiting *$Billing_URL*. This is an automatically generated email. Please do not reply to this message.
Login Password Changed	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Login Password Has Been Changed Dear *$User*, Your login password has been changed; you can use the new password to login. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
End User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated. You can now continue managing and paying your bills online using the online Customer Care Application. Why wait! Please visit *$Secure_Billing_URL* to finish the reactivation process. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.
CSR User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated; you can once again access your online Customer Care Application. Please visit *$Secure_Billing_CSR_URL* to finish the reactivation process. This is an automatically generated email. Please do not reply to this message.
Migrated End-User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated. You can now continue managing and paying your bills online using the online Customer Care Application. Why wait! Please visit *$Secure_Billing_URL* to finish the reactivation process. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Notification Type	Email Template (Consumer Edition)
Migrated CSR User Account Reactivated	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Account Has Been Reactivated Dear *$User*, Your account has been reactivated; you can once again access your online Customer Care Application. Please visit *$Secure_Billing_CSR_URL* to finish the reactivation process. This is an automatically generated email. Please do not reply to this message.
Password Expired	**From:** admin@yourcompanydomain.com **Sent:** *Date Time* **To:** User **Subject:** Your Administrator Password is Expiring Dear *$User*, Your account password will expire in *$Number_Days* days. Please set a new password. Thank you for using online payment. This is an automatically generated email. Please do not reply to this message.

Adding a Custom Message Provider

Use the following procedure to add a custom message provider.

To add a custom message provider

1 Create a new message type and accompanying class to override the com.edocs.common.api.notification.AbstractMessage class.

2 Initialize the new class with all the information necessary to send a message of that type using the custom messaging provider.

   It is not necessary to include static information that does not vary by individual message. For example, an email address is necessary if you are using an SMTP provider and the address changes for every message. However, the SMTP hostname does not change with every message.

   If an existing implementation has all the necessary information, you can use that implementation without modification, except that you must provide a unique message by calling the setMessageType method on the object after object creation. The class com.edocs.common.notification.extensions.InternalMessage provides a reference implementation.

3 Create a new transport class with the logic for sending messages using the new message provider, overriding the com.edocs.common.notification.extensionsapi.AbstractTransporter class.

   This transport class contains the methods for sending the message. All of the information necessary for this class is available from the IMessage object.

4 Add a JavaBean definition to the notification.xma.xml file. Give the file a name that ends with Bean, such as CustomTransportBean.

   This name (without the Bean part) is used inside the NotificationService class to return the correct messenger, for example: IMessenger messenger = MessengerFactory.getMessenger(CustomTransport).

5 In the JavaBean definition, add all the properties required by the transporter to use the messaging provider, such as the SMTP name. Do not include items like the email address, which the message object provides.

   The class com.edocs.common.notification.extensions.TrueTransporter and the JavaBean definition for TrueTransporterBean in the notification.xma.xml file provide a reference implementation.

6 Create a new NotificationService class that determines when to use the transport class, which you wrote in Step 3, that overrides the com.edocs.common.api.notification.INotificationService class.

   INotificationService has two methods for sending instant messages and batch messages. These methods decide which transport to use based on the IMessage object being passed in. You must supply the logic to call the MessengerFactory with the transport in Step 3 as the transport type, for the types of Messages you want that transport to send.

7 After you create the NotificationService class, add (or modify if it already exists) a JavaBean called NotificationService to the notification.xma.xml file. NotificationServiceFactory looks up the JavaBean called NotificationService from the XML file.

   The class com.edocs.common.notification.core.NotificationService and the JavaBean definition for NotificationService in the notification.xma.xml file provide a reference implementation.

About Email Notification Processing

Oracle Self-Service E-Billling interacts with the Oracle Self-Service E-Billling database to determine what email to send. Each message composed is stored, then the email dispatcher takes the stored email and sends the messages based on selected external transport type, for example, SMTP.

Figure 1 shows an overview of email processing.

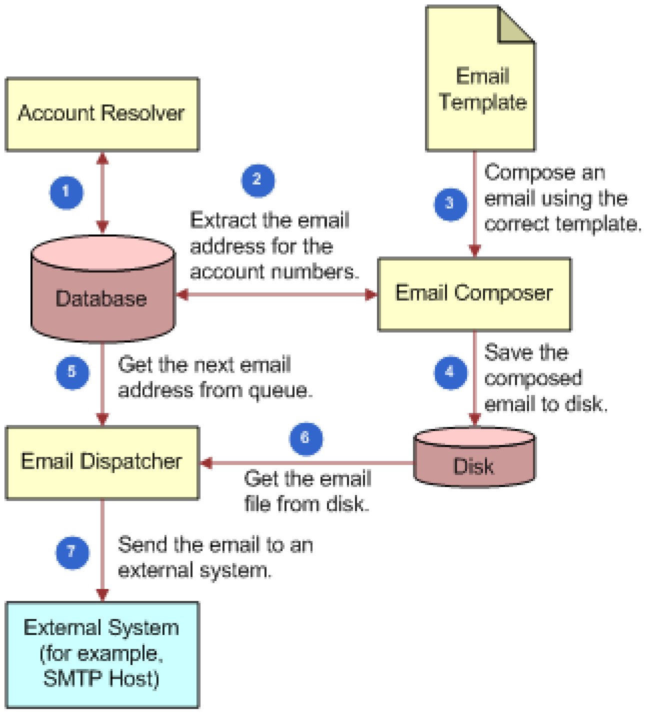

Account Resolver
Email Template
1
2
Extract the email address for the account numbers.
3
Compose an email using the correct template.
Database
Email Composer
5
Get the next email address from queue.
4
Save the composed email to disk.
Email Dispatcher
6
Get the email file from disk.
Disk
7
Send the email to an external system.
External System (for example, SMTP Host)

Figure 1. Email Processing

Oracle Self-Service E-Billling merges the message template with runtime information to create the email message. The messenger then calls the gateway, configured in the app-config.properties file, to send the email message.

The email composer consists of three components that function together to group multiple account numbers by email address to roll-up messages. The components of the email composer perform the following functions:

- Group account numbers by send-to address
- Compose a group message based on a template
- Create a grouped message for a given queue

Figure 2 shows how email groups email messages for delivery.

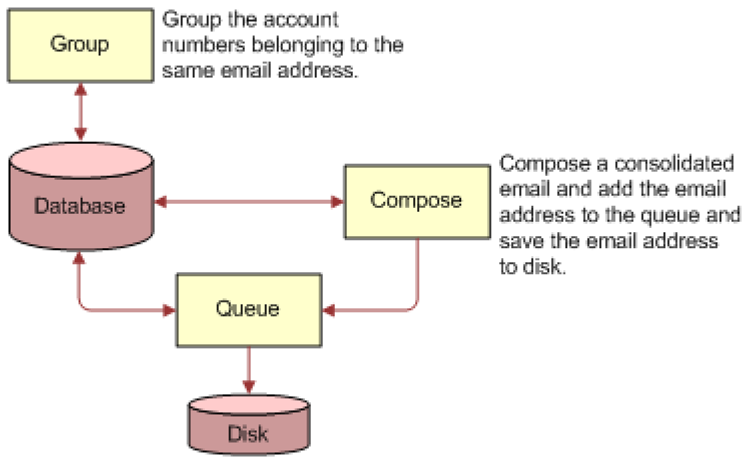

Group
Group the account numbers belonging to the same email address.
Database
Compose
Compose a consolidated email and add the email address to the queue and save the email address to disk.
Queue
Disk

Figure 2. Email Grouping

6 Using the Reporting Engine

This chapter covers using the Business Reporting Engine feature in Oracle Self-Service E-Billling. It includes the following topics:

Reporting Engine Features

The Reporting Engine is used for much more than just reporting. The Reporting Engine can present any data you can retrieve from any data sources, such as RDBMS or CSV files.

Possible use cases supported by the Reporting Engine include:

- Viewing statements and invoices
- Analytic reports, such as the 10 most expensive calls
- Cost center reports (hierarchy report), such as cost summary by cost centers
- Reports, such as most frequent users or logging analysis
- Address book
- Email content composition
- AR file generation

The Reporting Engine offers great tools to help you implement these use cases. It uses XML to describe how you want to present a report. Then the Reporting Engine does the rest of the work for you, including retrieving data from data source, formatting, and then presenting the data to the end user through Velocity templates.

The Reporting Engine is designed to do the following:

- **Use XML Files:** Create an XML file to describe the report you want to create. The Reporting Engine automatically generates that report for you, in variety of formats, such as HTML or CVS.
- **Have an extendable, customizable UI:** You can extend the Reporting Engine to support any UI customization. The Reporting Engine uses Velocity templates, which is a powerful reporting tool based on Model-View-Controller (MVC) technology.
- **Be maintainable:** The Reporting Engine is MVC-based and offers the best separation of presentation logic and business logic, which makes it maintainable.

The following features are offered by the Reporting Engine:

- **Multiple data sources.** The Reporting Engine connects to multiple data sources, including SQL data source, object data source, and DSV data source.
- **Prompts.** Prompts allow you to select desired data from data source.
- **Interactive sorting.** Sorting can be case sensitive or insensitive.
- **Interactive grouping.** Data is grouped by a particular column's values.
- **Calculator operations.** Summary, Boolean, minimal, average and count operations are supported.
- **Charting.** This feature supports bar, stack bar charts, line, and pie charts.
- **Template.** Template-based presentation for both Web-based and non-Web based applications.
- **Formatting.** Support is provided for locale based format for numeric values and dates.
- **Printer friendly view.** This feature allows you to generate a printer friendly view for printing.
- **CSV download.** CSV download lets you download the report in CSV format.
- **XML download.** XML download lets you download the report in XML format.
- **PDF download.** PDF download lets you download the report in PDF format. PDF format is not generated automatically for all the reports. You must create an RTF template file for particular reports to generate in PDF format.
- **Paging.** Pages through a large set of data.
- **Custom report.** Custom reports allow users to create their own reports and save them for later retrieval.
- **Internationalization.** Standard Java resource bundle based internationalization.
- **Drilldown and Breadcrumb links.** The Report engine offers a way to drill down to different reports and drill back through breadcrumb links.
- **Seamless integration with Struts and Tiles.** The Reporting Engine is not tied to a particular presentation framework, but offers excellent support for Struts and Tiles.
- **Batch report.** When it takes a long time to generate a report online, you can use the batch report feature to send a request which will be processed offline.
- **Unlimited Paging.** If the data source has too many rows and if there is a performance issue to retrieve all the rows, the Reporting Engine can retrieve them in batches. The paging though these batches is seamless and retrieving result set in batches is invisible to the end user.

If you are working on the UI, consider using the Reporting Engine whenever you want to present a tabular table with sorting and paging functionality. For a non-tabular based UI, use JSP files.

The Reporting Engine is also useful for generating dynamic text files, such as AR files or email content.

It is recommended that you use JSP for most parts and only use Velocity templates for reporting-related UIs.

The Reporting Engine uses Velocity instead of JSP because:

- Velocity offers a better MVC module, which keeps most of the business logic in the core Reporting Engine APIs.
- The Reporting Engine is meant to be used by both front end and back end applications. For back end applications, JSP is not available.
- The views (templates) must be publishable and versioned. This is important if you want to use the Reporting Engine to present bills. Note that there is no easy way to publish JSP pages.

It is not recommended to download a newer version of Velocity and replace the one in the EAR file. The new Velocity version has not been tested with Oracle Self-Service E-Billling. Also, the default velocity.properties file has been changed for Oracle Self-Service E-Billling. These changes include: the velocityCount starts from 0 instead of the default 1, and the templates can be loaded as file and also as class.

You cannot define your own data source. One way to get around this is to retrieve your data as a list of objects and then use the Object Datasource feature to present it through the Reporting Engine.

You cannot extend report XML to add your own custom tags.

Reporting Engine Architecture

Figure 3 shows the Reporting Engine architecture based on the UML component model.

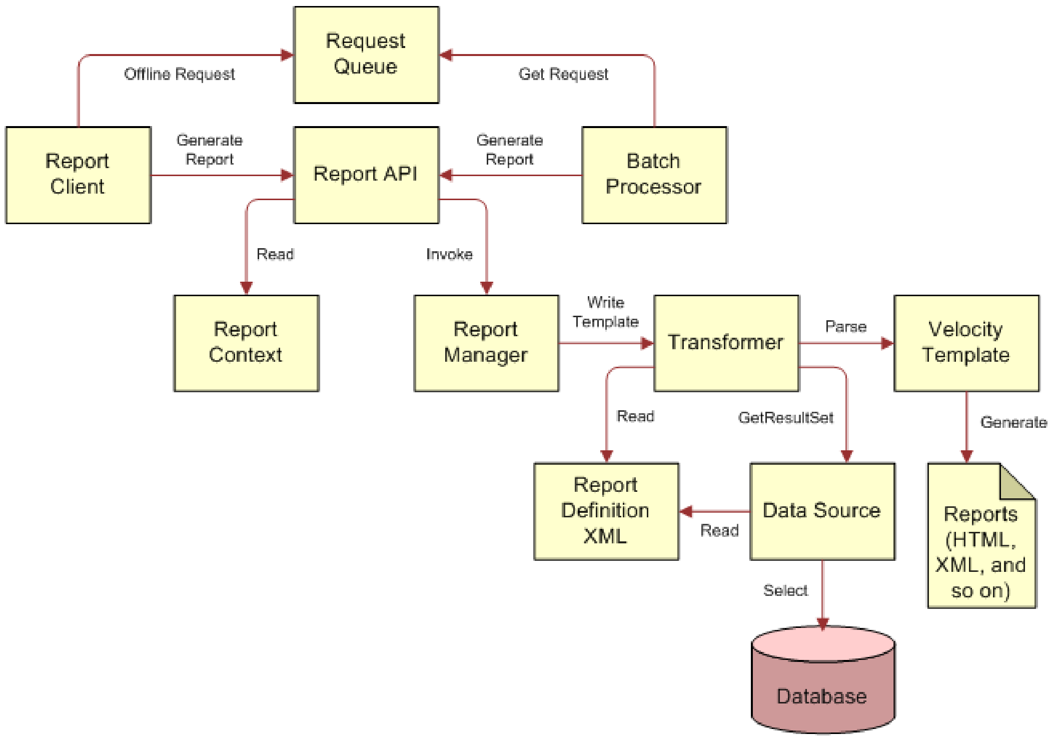


Figure 3. Reporting Engine Architecture

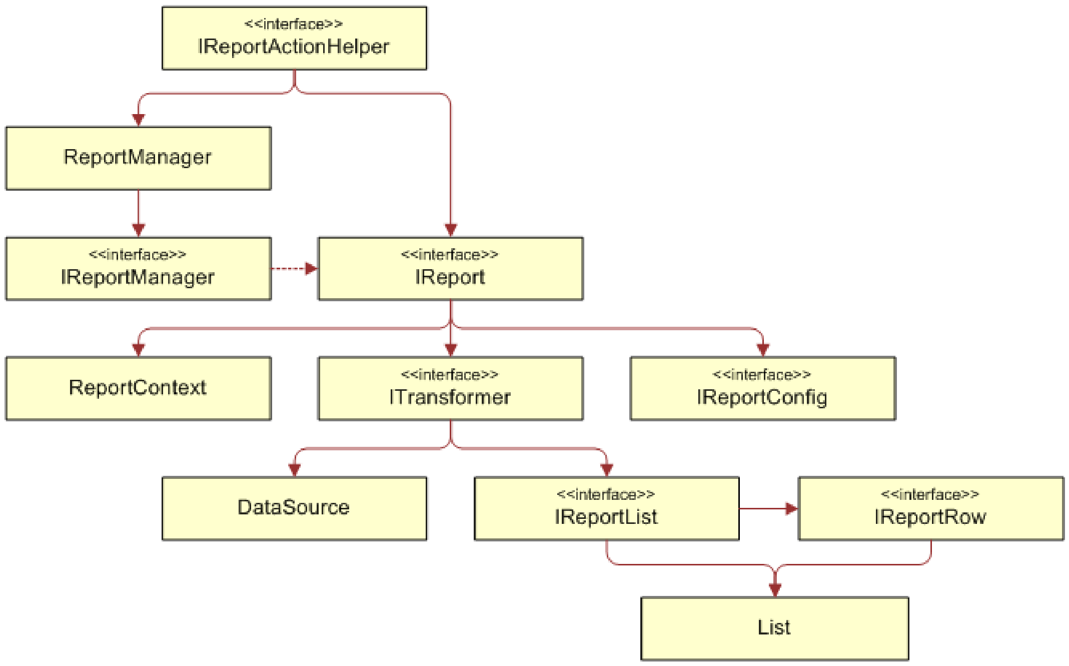
The overall Reporting Engine architecture follows the MVC model; the data source is the model, the report manager, transformer and report XML are the controller, and the template is the view.

- **Report Client.** This client calls the Report API to generate reports. The client can be a Web Client, such as JSP and Servlet or Struts and Tiles, or it can be a regular standalone application.
- **Report API.** This is a set of APIs that the reporting client can use to generate a report. For information about how this API works, go to the Oracle Self-Service E-Billling Javadoc as described in "Accessing Oracle Self-Service E-Billing Javadoc" on page 31.
- **Report Context.** The Report context is used by the Report Client to exchange information with the Report Engine. It includes the information passed from the client that is used to bind the SQL query parameters and parse the templates. For example, the context can contain user session information, such as login name, current role and organization level. Or it can contain report input information, such as the date range used to generate reports. All the objects in the context can be accessed using Velocity templates.
- **Request Queue.** This queue holds all offline batch report requests. Users can generate reports immediately, or they request that the reports be generated offline. Offline reports send email notification when the reports are ready. The Request Queue is a JMS queue, and holds all offline report requests.

- **Batch Processor.** The processor retrieves offline report requests from the Request Queue, and sends them to the Report Engine for processing. The batch processor is a batch job that runs in Command Center.
- **Report Manager.** The Report Manager is the central controller of the report engine. It receives reporting requests from the client, and invokes the appropriate data source and transformer to perform the desired processing.
- **Data source.** This item represents the data source. The data source can be an SQL statement, an Object or a CSV file.
- **Transformer.** The transformer transforms the query result from presentation, and applies a set of computations on it, including sorting, grouping, paging, aggregation (summary, average, Boolean, minimal, count), and formatting. The transformer can also cache the data retrieved from data source so that the operations can be performed in the cache data (which reduces database accesses).
- **Velocity Template.** Templates are used to generate desired report output views. The templates are based on Velocity, and can generate any text reports, such as HTML or CSV. However, it is not currently possible to use Velocity to generate binary reports.
- **Report Definition XML.** Report XML files control how reports are generated. To create your own report, create a report definition in a report XML file. You can have multiple report XML files, and each report XML file can define multiple reports.

Reporting Engine Object Model

Figure 4 shows the Reporting Engine object model. Only the main objects are shown.



<<interface>> IReportActionHelper
ReportManager
<<interface>> IReportManager
<<interface>> IReport
ReportContext
<<interface>> ITransformer
<<interface>> IReportConfig
DataSource
<<interface>> IReportList
<<interface>> IReportRow
List

Figure 4. Reporting Engine Object Model

- **ReportActionHelper.** This class was designed to be called by the Servlet or Struts action class. It performs a number of tasks, such as parsing the request parameters, and then does the sorting, paging, and so on. It returns an IReport object, that you can use to render a report, or manipulate further before rendering it. Though it is possible to avoid using this class by using other APIs, it is strongly recommended that you use this class to reduce your customization work.
- **ReportManager.** Use this class to get an instance of IReportManager.
- **IReportManager.** This is the entry class to the Reporting Engine APIs. For example, to get an instance of IReport and other objects.
- **ReportContext.** This class is a Map, which allows the Reporting Engine client to pass information to the Reporting Engine. For example, the binding values to SQL "?" parameters, and the objects used in Velocity templates.
- **IReportConfig.** This interface represents the report XML definition. For example, the SQL used to query, instructions to bind the report context objects to the SQL, instructions to format the report. There are a set of Config objects related to this class that represent the report XML elements. For more information about this API, go to the Oracle Self-Service E-Billling Javadoc as described in "Accessing Oracle Self-Service E-Billing Javadoc" on page 31.
- **ITransformer.** This object represents the transformer defined in the report XML. It offers a set of APIs that manipulate the format, such as format value, write the template, and so on.

- **DataSource.** This API is not a public. It represents the datasource defined in the report XML, and allows you to retrieve report data from that data source.
- **IReportList and IReportRow.** The report data retrieved from DataSource is represented as `IReportList`, which is a `java.util.List`. `IReportList` includes a list of `IReportRow` objects, which represents rows in a report. The objects in `IReportRow` are basic Java objects, such as Integer, Double, String, Date, and so on. For more details, please check Java APIs of Reporting Engine.

The object model of Reporting Engine is straightforward. Figure 5 shows how the Reporting Engine objects interact with each other to generate a report.

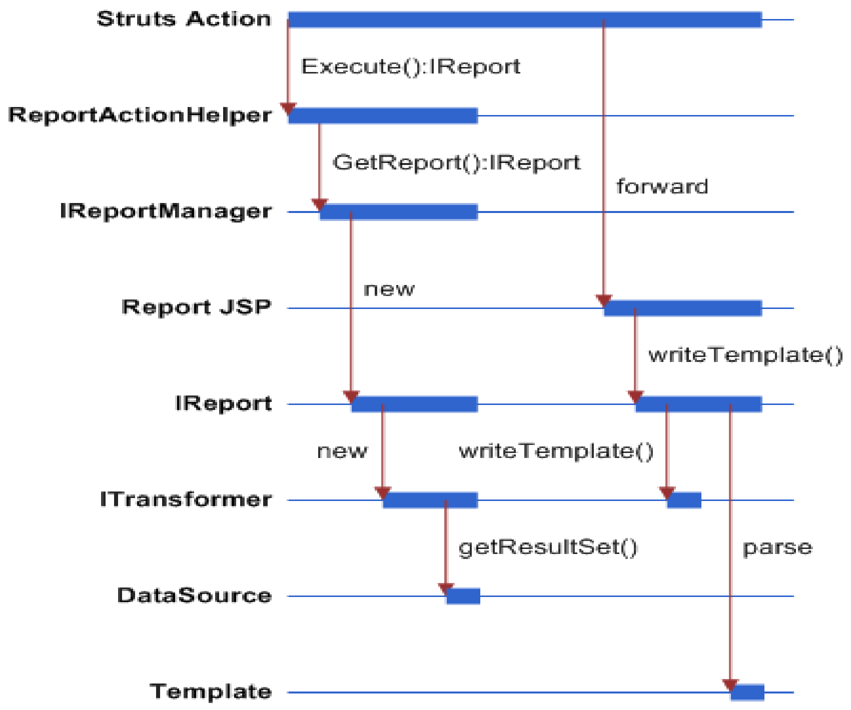


Figure 5. Reporting Engine Object Interaction

For more information on how to write action class and report JSP pages, see "Customizing the Reporting Engine" on page 138.

Components Used by the Reporting Engine

The Reporting Engine uses the following components:

- Reporting XML
- Templates

- Reporting API

The Reporting Engine is packaged as part of the Oracle Self-Service E-Billling application. However, the Reporting Engine is an individual component which can be used in any other application.

The following list describes the components of the EAR file required to customize Reporting:

- **Velocity-*version_number*-custom.jar and Velocity-*version_number*-tools.jar.** Contains the Velocity template engine and related files. Note, the property file for the Velocity engine has been updated for Oracle Self-Service E-Billling; do not replace the property file with any other version of Velocity JAR files. In the file name, *version_number* is defined by Velocity.
- **Api-*version_number*-SNAPSHOT.jar.** This archive includes the public APIs that a Reporting Engine client can use to access the Reporting Engine. The APIs are under com.edocs.common.api.reporting. For more information about APIs, see the Oracle Self-Service E-Billling Javadoc as described in "Accessing Oracle Self-Service E-Billing Javadoc" on page 31. In the file name, *version_number* is defined by Velocity.
- **Reporting-core-*version_number*-SNAPSHOT.jar and Reporting-forms-*version_number*-SNAPSHOT.jar.** This archive includes the Reporting Engine implementation classes. In the file names, *version_number* is defined by Velocity.
- **App-resources-*version_number*-SNAPSHOT.jar.** This archive includes the resource bundles used by the Oracle Self-Service E-Billling application. The resource bundles are loaded at the EAR level instead of the WAR level in order to localize the batch reports, which are generated offline. Currently, all reporting related resource bundles are in the com.edocs.app.reporting.resources.ApplicationResources*Language*.properties files. In the file name, *version_number* is defined by Velocity. In the properties file, *Language* is blank for U.S. English, _es_US for Spanish, _zh_cn for Chinese, and _it_IT for Italian.
- **struts-config.xml.** The Oracle Self-Service E-Billling user interface is defined as a Struts module. All the Oracle Self-Service E-Billling related Struts configurations, such as the resource bundles, are defined in this file. (This file is found in ebilling-web-1.0-SNAPSHOT.war.)
- **Skin.css.** This file defines the report UI related CSS. (This file is found in ebilling-web-1.0-SNAPSHOT.war.)
- **Report.jsp and other jsp files.** This item consists of the Oracle Self-Service E-Billling reporting related JSP files. The report.jsp renders the major reporting UI. You call IReport.writeTemplate() to invoke Velocity template parsing. The view rendering is done through Velocity templates. (This file is found in ebilling-web-1.0-SNAPSHOT.war.)

In addition to the EAR file, which includes classes and JSPs, there are a set of files packaged outside the EAR. These files are Velocity template files and report XML files.

Using the Report List Properties File

The report list properties file, reportList.properties, includes the list of report XML files to be loaded into report engine. You must have your report XML file defined in this file. The file format is:

```
name=xml_file_path
```

In this definition, name must be unique for each report XML and the XML file must be either under *EDX_HOME* or on the class path, where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billling.

The following example, the telco.xml file is found under the EDX_HOME/config/rpt/ directory or on the class path. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billling:

```
telco_xml=config/rpt/telco.xml
```

Configuring Batch Reporting

You can specify the following parameters for batch reporting in Oracle Self-Service E-Billling:

- **batchReport.failTries.** This parameter specifies the number of times to retry the batch report job after a transaction timeout exception.
- **batchReport.processingTimeOut.** This parameter specifies the time out period after which a user can delete a batch report request with PROCESSING status.

To configure batch reporting

- Update the appropriate parameters in the globalConfig.properties file, found in the *EDX_HOME/* config/rpt directory (or the *EDX_HOME*\config\rpt directory on Windows):

```
batchReport.failTries=3

#The unit is hour.

batchReport.processingTimeOut=12
```

Reporting XML

The Reporting XML is central to the Reporting Engine. It describes how to generate a report.

The Reporting XML includes the dataSource and the transformer sections. The dataSource describes how to retrieve data from data source, and the transformer manipulates the data before sending it to the template.

There are samples of report XML files in the *EDX_HOME/*config/rpt directory. To get a complete list of all the valid report XML elements and attributes, see the report XSD file, report.xsd, under the *EDX_HOME/*config/rpt directory. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billling.

The following topics describe some of the main features of the Reporting Engine and explain how to use report XML to implement them.

<reports> Element of Report XML

This is the root element of report XML. This element can include <report>, <localizer>, <prompts> and <templates> elements. The following XML shows that structure:

```
<reports>
   <templates>…</templates>
   <localizer>…</localizer>
   <prompts>…</prompts>
   <report>…</report>

</reports>
```

<localizer> Element of Report XML

This element defines how the localization of the reports will be done. For details, see "Internationalization and Localization of Reporting" on page 133 for more information.

The <localizer> element has the attributes described in Table 9.

Table 9. Attributes for <localizer>

Name	Required	Description
enableMessageResources	No	This attribute allows you to use Struts MessageResource to look for the resource bundles for reports. This means you can use the same copy of resource bundle files defined in a struts config file without reloading another copy of it. The default is true.
defaultCode	No	This attribute enables you to define the default behavior if a resource is not found. ■ A value of 0 means to use the key as the default value. ■ A value of 1 means to use Struts notion of "???<locale>.<key>???" A value of -1 means to throw an exception.

Localizer can include <resourceBundle> as its child elements.

<resourceBundle> Element of Report XML

This element specifies one resource bundle property file name to be used for report localization. See Internationalization/Localization on page 55 for more information.

For example:

```
<resourceBundle name="config/l10n/message" />
```

This example means the property file, config/l10/message_<locale>.properties, found in the *EDX_HOME* directory (the directory where you installed Oracle Self-Service E-Billling) is used for localization.

<prompts> Element of Report XML

The <prompts> element has the same format as the one defined under <dataSource>. However, because it is defined at the global level, it can be shared and referenced by other reports. This significantly reduces duplication of the report XML contents, and makes it easier to maintain report XML files.

For more details, see the <prompts> definitions in "<dataSource> Element of Report XML" on page 98.

<templates> Element of Report XML

This element allows you define a list of global templates that can be included and parsed into other templates. For example, the paging.vm is used to generate paging UI and could be included by other templates, like report_body.vm.

For example, to define a template:

```
<templates>
    <template id="paging.vm" name="template/common/reporting/paging.vm"/>
    </templates>
```

This example means there is a template named paging.vm, located in the `EDX_HOME/template/common/reporting/` directory.

Then you can include the paging.vm from another template like this:

```
#parse ($transformerConfig.getTemplateName("paging.vm"))
```

The method `transformerConfig.getTemplateName("paging.vm")` returns this template, paging.vm, from the *EDX_HOME/*`template/common/reporting/paging.vm` directory.

NOTE: If you have a template that has the same ID defined inside the transformer element, then the ID in transformer takes precedence over the is in the global template list. This allows an individual transformer to use its own template. See <transformer> for detail.

<template> Element of Report XML

This element defines a global template, which has following attributes described in Table 10.

Table 10. Attributes for <template>

Name	Required	Description
ID	Yes	The ID is a unique identifier among all the global templates. Note you can use the same ID for the transformer template ID; in this case, the transformer template takes precedent of the global one.
name	Yes	The name attribute is the full class path name of the template.

<report> Element of Report XML

This element defines a report. A report can include zero or more <dataSource> elements, one or more <transformer>s, and zero or one of <customList>, <printList> and <downloadList>.

```
<report id="reportId" name="MyReport">
   <downloadList>…</downloadList>
   <printList>…</printList>
   <customList>…</customList>
   <dataSource>…</dataSource>
   <transformer>…</transformer>
</report>
```

The <report> element has two attributes:

- **id.** The ID identifies this report. All the reports defined in the report XML files in reportList.properties must have unique IDs. This ID must start with an alphabetic character, and can include numbers and underscores.
- **Name.** This is the name of the report. This name is used to search the report bundle to get a localized version of the report name. For example, in the Report List page, the names of reports are from this attribute.

<dataSource> Element of Report XML

This element defines how to retrieve data from the data source.

```
<dataSource id="" uri="jdbcJNDI:edx.report.databasePool">

   <query dynamic="true">

   </query>

   <columns>

      <column id="" type=""/>
```

```
        </columns>
        <inputBindings>
            <inputBinding />
        </inputBindings>
    </dataSource>
```

The data retrieved from the data source is represented as a List of Lists of simple Java objects, such as Strings, Date/Time/Timestamp or Numbers. It does not use a two dimensional array because: a List of Lists gives you the potential to increase its size if required, and Velocity does not support accessing array elements through the [ ] operator.

The <dataSource> element has following attributes:

- **id:** A unique ID identifies this data source in this report. You must define it even there is only one data source. It is not required that the ID be unique across all reports. This ID must start with an alphabetic character, and can include numbers and underscores.
- **uri:** A Universal Resource Identifier identifies the location of the data source. Oracle Self-Service E-Billling supports three data sources: SQL data source, object data source, and DSV data source. This example focuses on the SQL data source. For information about object data sources, see "Object Data Source" on page 136 and for DSV data source, see "DSV Data Source" on page 136.

  For an SQL data source, there are three URIs:

  - **jdbcJNDI:<dataSource_JNDI_NAME>.**The jdbcJNDI indicates that this is a JDBC data source identified by its JDNI name. For example, jdbcJDNI:edx.report.databasePool means there is a JDNI data source named edx.report.databasePool.
  - **jdbcRef:<dataSource_REF_NAME>.** The jdbcRef indicates that this is a JDBC data source identified by its local reference name, either defined in the web.xml or ejb-jar.xml file. For example, you can have an entry similar to this in the web.xml file:

    ```
    <resource-ref>
       <res-ref-name>jdbc/rptDataSource</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
    </resource-ref>
    ```

    With this entry, you can use following URI: jdbcRef:jdbc/rptDataSource. You must also resolve this local reference through the weblogic.xml file or another vendor-specific XML file.
  - **jdbcDirect:<jdbc_config_property_file_class_path>.** The jdbcDirect means that there is no connection pool and the Reporting Engine must make a direct JDBC connection to the database. You must specify the class path to the DB config file. For example, jdbcDirect:config/db/jdbcConfig.properties. For the format of the config file, look at the sample jdbcConfig.properties file coming with the product. Avoid using this URI if your application can access a connection pool.

    This element can include <query>, <inputBindings>, <prompts> and <columns> elements.

<query> Element of Report XML

This element defines the query used to retrieve data from the data source. It applies to an SQL data source but not an object data source.

```
<query dynamic="false" maxRows="1000"> <![CDATA[select name, amount from summary
where user_id=? ]]></query>
```

The value for <query> is enclosed in a CDATA topic, which can include any SQL.

The question mark in the SQL means that a variable must be resolved (bound) before the SQL can be executed. Variables are resolved through the <inputBindings> element.

Table 11 describes the attributes of the <query> element.

Table 11. Attributes for <query> Element

Name	Required	Description
dynamic	No	This attribute indicates whether to parse this SQL as a Velocity template before execution. This allows you to use a Velocity template to generate a SQL dynamically. For information about how to write dynamically generated SQL, see "Dynamic SQL" on page 132. True or False; the default is False.
maxRows	No	This attribute indicates the Boolean number of rows will be retrieved from the data source. An integer; the default is 1000.

<inputBindings> Element of Report XML

This element defines a list of input bindings that are used to bind the SQL variables defined in the <query> element. It has no attribute, and includes an <inputBinding> element.

<inputBinding> Element of Report XML

This element defines a single input binding. There are two kinds of bindings: objects and prompts. The order of the <inputBinding> elements is the same as the order of the SQL variables. That means the nth <inputBinding> is used to bind the nth SQL variable. Object binding means binding an object or its property to an SQL variable.

For example:

```
<inputBinding object="bean" property="userId" />
```

This means there is an object called bean in the report context, this object is a JavaBean, and it has a property named userId. The value returned by bean.getUserId() will be used to bind the SQL variable. Usually, the JavaBean is a Struts ActionForm object. If the object returned by the property is a Collection, then each element in the Collection will be used for binding.

```
<inputBinding object="myObject" />
```

In this case, there is no property defined, so myObject is not assumed to be a JavaBean. If the myObject is not a Collection, then myObject is used to bind to the SQL variable directly. If the myObject is a Collection, then each element in the myObject Collection will be used to bind to the SQL variables in its natural order in the collection. This latter case is very useful where the number of SQL variables is dynamic, such as a name in a (?…?) clause. For more information about using dynamic SQL, see "Dynamic SQL" on page 132.

Prompt binding is a special case of object binding. Prompt binding means that the binding object is from the user prompt, which allows you to bind the value of the prompt to a SQL variable.

```
<inputBinding object="form" property="<bean_property>" prompt="<prompt_id>" />
```

You can use a map-backed ActionForm also. For example, the ReportForm from the Oracle Self-Service E-Billling application is a map-backed form. It has map-methods, such as getParameter(String name) and setParameter(String name, Object value). You can use this syntax in a property or prompt attribute:

- <inputBind object="form" property="parameter(callType)" />
- <inputBind object="form" property="parameter(callType)" prompt="parameter(callType)"/>

Table 12 describes the attributes of the <inputBinding> element.

Table 12. Attributes for inputBinding Element

Name	Required	Description
object	Yes	The name of the object in the report context used for binding. This object must be put into report context. In the case of prompt binding, the Reporting Engine automatically retrieves the prompt value from the prompt form, and puts this object into the report context. The `ReportActionHelper` class puts the value of the prompt into the context with that name. In the non-prompt case, the caller of Report engine must put this object into context.
property	No	This attribute is optional. When it appears, it means the object is a JavaBean and the value of the property of this JavaBean is used to bind SQL variable. If this property is not there, then it means the object identified by object attribute is used for binding. **NOTE:** A map-backed property is supported, such as parameter(callType).
prompt	No	This attribute indicates that this input binding is from a prompt, and the value of it must be the ID of the prompt defined in the <prompts> element.

NOTE: The object name for the prompt form is fixed to form and you must use object=form for prompt.

<prompts> Element of Report XML

This element defines an HTML form whose input is used for data source input bindings. Each input field in the form is called a prompt. You configure where the prompt gets its original data (from a database or from a fixed value list), and how it will be presented by the report XML. The Reporting Engine builds the report prompt (input) UI, which is fully customizable (it uses a template to generate the UI).

To control the look and feel of prompts, reporting uses a technique similar to tiles; layout.vm controls the layout format, and prompt.vm controls the prompt rendering.

The <prompts> element has a list of prompt blocks. Each block is separated by that dark blue bar at the top, and you can define a label for each blue bar. Inside each block, you can define a list of groups, where each group has a list of prompts. Each prompt group acts like <tr> in an HTML table, and all prompts within a prompt group display horizontally in a row. Each prompt must belong to a group. Prompts can be HTML input or a plain label. In the preceding example UI, Data range is a group with two prompts: the start date and end date. Usage type is another group that has two prompts: usage type and call type.

The <prompts> definition used to generate the example UI is:

```
<prompts id="prompts1" formName="reportForm" action="report.do"
      method="post" templateID="layout.vm">
  <block>
<group label="Date Range:" >
<text id="fromDate" size="12" value="1/1/2004"
    imgSrc="_assets/images/calendar.gif" label="From:"
    labelPosition="top"/>
     <text id="toDate" size="12" value="12/1/2004"
         imgSrc="_assets/images/calendar.gif" label="  To:"
                 labelPosition="top"/>
      </group>
      <group>
     <select id="parameter(usageType)" report="prompt_usageType"
    displayColumnId="usage_type_name"
    valueColumnId="usage_type_key" value="2"
    label="Usage Type:"/>
     <select id="parameter(callType)" report="prompt_callType"
    displayColumnId="call_type_name"
    valueColumnId="call_type_key" value="2"
    label="Call Type:"/>
            <image name="display" src="_assets/images/display.gif" />
  </group>
  <group label=" Billing Reports">
              <select report="prompt_reportList" value="first" name="reportId"
onChange="cleanupHiddenValues()"/>
      </group>

  </block>
</prompts>
```

You can define <prompts> under <reports> and it will be global. To refer to a global <prompts> from inside <dataSource>, use the following:

```
<prompts id="billingPrompts"/>
```

This expression means that there is a global <prompts> whose ID is billingPrompts. If the same <prompts> is used across multiple data sources, the global <prompts> helps you to maintain only one copy of it.

The <prompts> element has the attributes described in Table 13.

Table 13. Attributes for <prompts> Element

Name	Required?	Description
id	Yes	A unique ID is used to identify this prompts list in this data source. Oracle Self-Service E-Billling supports one prompts element for each data source.
formName	No	The name of the HTML form and default reportForm. It is only useful if you want to use JavaScript to manipulate the form.
action	Yes	The action of the HTML form. Use report.do for the action because it is used as the default. If you change the action name defined in Struts config XML, you must search all your JSP pages and Velocity templates to replace it.
method	No	The default is post.
templateID	Yes	The template ID specifies the layout template ID. The template must be either defined in corresponding transformer’s <templates> or in the global <templates>.
enctype	No	The encryption type.
onReset	No	The name of JavaScript being called when Reset is called on the HTML form.
onSubmit	No	The name of JavaScript being called when Submit is called on the HTML form.

The <prompts> elements contain one or more <block> elements.

<block> Element of Report XML

This is an optional element. If you do not define it, then you can define group directly under <prompts>, and all the groups will be put, implicitly, under a block. You can define a label for a block and the label will be displayed in the blue bar of the prompt.

<group> Element of Report XML

This element defines a group of prompts. This group of prompts will be displayed horizontally in one line. Different groups of prompts will be displayed vertically.

The <group> element has the attributes described in Table 14.

Table 14. Attributes for <group> Element

Name	Required?	Description
label	No	The label displays at the beginning of the each prompt group.
description	No	The description displays for the rollover question mark.

There are eight types of prompts, which correspond to input types in an HTML form (except Label).

Some supported HTML forms are: text, check box, select, radio, image, submit, reset and label. Image, submit, reset, label are purely for HTML form rendering and manipulation; their values are not used for report SQL input bindings. Check box, select, radio and text can be used for SQL input bindings.

Attributes for prompt related configuration in XML file, most of attributes are from an HTML form, others are required by the Report Engine.

The <group> element can include one of the following attributes: <checkBox>, <select>, <radio>, <text>, <image>, <label>, <submit> and <reset>.

<select> Element of Report XML

This element defines a select prompt. A select prompt allows you to select one or more values from a list of values. A select prompt must associate with a report whose result set is used to populate the select list. For example:

```
<select id="parameter(callType)"
  report="prompt_callType"
  displayColumnId="call_type_name"
  valueColumnId="call_type_key"
  value="2"
  label="Call Type:"/>
```

A select list requires two types of information: display values and actual values. The display values are for displaying, and the actual values are for querying. For example, you can display May 2010, but use an internal value 5 for a query. For example:

```
<select>
     <option value=”5”>May 2010</option>
</select>
```

To render the preceding UI, get the options values and display names from the associated reports. Table 15 describes the select options.

Table 15. Attributes for <select> Element

Name	Required?	Description

id	Yes	Identifies this prompt in this prompts list. The ID is used as the name of the input prompt in the HTML forms, which means that it determines which `ActionForm` property is used to hold this input value. In the example, the `billPeriod` property of `ActionForm` holds the value of the select box.
		
		If there is no corresponding property in the `ActionForm` (if it is a map-backed form), you can use the Parameter property (a map-backed property) to get the value into the `ActionForm`.
		
		The following example creates a prompt for call type, which is not a property of `ActionForm`:
		
		`<inputBinding object="form"`
		`property="parameter(callType)"`
		`prompt="parameter(callType)"/>`
		
		where prompt is declared as:
		
		`<select id="parameter(callType)" label="Call Type:">`.
		
		NOTE: When using `parameter(calltype)` as id (and therefore the HTML input file name), JavaScript might not recognize the name. In that case, you might want to extend your ActionForm implementation to be a regular JavaBean property, which allows you to use `<select id="callType" >`.

label	No	The label of this prompt. Used for display.
labelPosition	No	Display the label position against the prompt. Top, bottom, left, and right are supported: ■ **Top.** The label is at the top of the prompt. ■ **Bottom.** The label is at the bottom of the prompt. ■ **Left.** The label is to the left of the prompt. ■ **Right.** The label is to the right of the prompt.
size	No	Size of the HTML input field.
report	Yes	The ID of the report, whose result set will be used to populate the Select element. The report can load data from the database or it can load from a DSV data source which is useful if the data in the list is fixed.

Table 15. Attributes for <select> Element

Name	Required?	Description
displayColumnId	No	The column ID of the report, whose values will be used as the display names of the <option> fields of <select> list. The first column of the report is used when `displayColumnId` is not specified.
valueColumnId	No	The column ID of the report, whose values will be used as the values of the <option> fields of <select> list. The second column of the report is used when `valueColumnId` is not specified.
value	No	The default value for the <select> list. It can be: ■ first, using the first value in the `valueColumnId` column of the report ■ last, using the first value in the `valueColumnId` column of the report An integer N, such as 1 or 2, which indicates the nth value in `valueColumnId` column of the report. Note the index starts from 1.
multiple	No	Specifies that multiple items can be selected. True or False; The default is false.
onBlur	No	Name of JavaScript being called for onBlur event.
onChange	No	Name of JavaScript being called for onChange event.
onClick	No	Name of JavaScript being called for onClickevent.
onFocus	No	Name of JavaScript being called for onFocus event.

The report used to generate <prompt> must meet the following requirements:

- Have two columns: one column for display, and another for prompt value. The display column ID must match the displayColumnId attribute defined, and the value column ID must match the valueColumnId attributed defined. If the report only has only one column, you can have both displayColumnId and valueColumnId point to the same column.
- The report ID of the prompt report must match the report attribute defined.
- You can format the prompt display names by using pattern attribute of column element of the report.

<checkBox> Element of Report XML

The checkBox prompt allows you to print the prompt values in a list of check boxes. For example:

```
<checkBox id="billPeriod" label="Bill Period:"
  report="prompt_billPeriod"
  onClick="alter(‘onClick’)"
```

```
    displayColumnId="bill_period_name"
    valueColumnId="bill_period_key"
  value="last"/>
```

In the example, the bill period prompt is defined as a set of check boxes, where you can check one or more bill periods. The display names and values of bill period come from the prompt_billPeriod report. The <checkBox> element has the same attributes as <select>, except multiple does not apply. For information about using the <select> element, see "<select> Element of Report XML" on page 105. You can think of the checkBox element as just another view presenting the same prompt, similar to a multiple-select list. The data retrieved from data source for the <checkbox> element must be either true or false.

<radio> Element of Report XML

This prompt presents a list of radio buttons, only one of which can be selected.

```
<radio id="billPeriod" label="Bill period:"
  report="prompt_billPeriod"
  onClick="alert('onclick')"
  value="last" />
```

In the example, the bill period prompt is defined as a set of radio buttons, where you can only check one of the bill periods. The display names and values for bill period come from the prompt_billPeriod report.

The <radio> has the same attributes as <select>, except multiple does not apply. See <select> for more information. In fact, you can just think radio as another view of presenting the same prompt. <radio> is like a single-select list.

The data retrieved from the data source used for <radio> must be either true or false, and only one can be true.

<text> Element of Report XML

This element allows you to define a text box and use the user-entered value as the prompt value.

```
<prompt id="billPeriod" label="Bill period:">
  <text
     report="prompt_billPeriod"
     maxLength="10"
     onBlur="alert('onBlur')"
     onChange=" alert('onChange')"
     onFocus=" alert('onFocus')"
     onSelect=" alert('onSelect')"
     size="10"
     value="06/2004"/>
</prompt>
```

In the text prompt, size attribute determines the width of the prompt.

<image> Element of Report XML

This element allows you to define an image. For example this usage creates an image submit button:

```
<image name="display" src="_assets/images/display.gif" />
```

NOTE: The <image> element is different from the <img> HTML tag.

Table 16 describes the attributes for the <image> element.

Table 16. Attributes for <image> Element

Name	Required?	Description
name	Yes	The display name of the image.
scr	Yes	The image src.
align	No	Left or right.

<label> Element of Report XML

This element defines text to display in the form. For example:

```
<label name="ccc_toll_lbl" value="  and  " />
```

Table 17 describes the attributes for the <label> element.

Table 17. Attributes for <label> Element

Name	Required?	Description
name	No	Not used.
Value	Yes	The text to be displayed as it is on the screen

<reset> Element of Report XML

This element displays an HTML reset button. For example:

```
<reset name="reset" value="reset" />
```

Table 18 describes the attributes for the <reset> element.

Table 18. Attributes for <reset> Element

Name	Required?	Description
name	Yes	Name of the reset button.
value	Yes	The display value of the reset button.
onClick	No	JavaScript to invoke.

<submit> Element of Report XML

This element displays an HTML submit button. For example:

```
<submit name="submit" value="ok" />
```

Table 19 describes the attributes for the <submit> element.

Table 19. Attributes for <submit> Element

Name	Required?	Description
name	Yes	Name of the submit button.
value	Yes	The display value of the submit button.
onClick	No	The JavaScript to invoke.

<columns> Element of Report XML

This element, under <dataSource>, defines the list of columns retrieved from the data source. As described previously, the data retrieved from the data source is a two-dimensional matrix with rows and columns. For an SQL query, the rows are the rows from the SQL table, and the columns are the SQL table columns. Most of the transformer operations, such as sorting, grouping and calculation, are based on the types of the columns. Only the type of the column is important, not the definition of the column. For example, you can summarize if the type is Number; it does not matter if the definition is Air Fee or Toll Charge. That is the primary reason to use a List of Lists of objects to present all the data.

You must define all the columns retrieved from the data source in this element, in the same order as the data source. For example, if you are using a SQL data source, the order of selected columns from `Select` must be the same as the order defined in the XML element. The same is true for object data sources.

<column> Element of Report XML

This element describes the column retrieved from the data source. You must define the type of the column in this element. The order of <column> elements must be the same as the order of columns retrieved from the data source and for each column in the data source, you must have one of this XML element defined for it.

The element <column> includes the attributes described in Table 20.

Table 20. Attributes for <column> Element

Name	Required?	Description
ID	Yes	Uniquely identifies this column in the data source.
type	Yes	Type of column. The legal types are all simple Java object types. A column can be sorted if its type is java.lang.Comparable. or it can take a calculator operation (aggregation), if its type is java.lang.Number.
default	No	This attribute indicates the default value for this column if the value returned from data source is null.

Column types can be one of the following:

- **Java.lang.Object.** A generic type. Avoid using this if you want to sort or format on the column. Use a more specific type instead.
- **Java.lang.Double.** A double value, which can be sorted and aggregated.
- **Java.lang.Float.** A float value, which can be sorted and aggregated.
- **Java.lang.Integer.** An integer, which can be sorted and aggregated.
- **Java.lang.Long.** A Long value, which can be sorted and aggregated.
- **Java.lang.Short.** A Short value, which can be sorted and aggregated.
- **Java.lang.BigDecimal.** A BigDecimal, which can be sorted and aggregated.
- **Java.long.String.** A String value, which can be sorted.
- **Java.sql.Date.** A Date value (a Date has no time information). It can be sorted.
- **Java.sql.Time.** A Time value (a Time has no date information). It can be sorted.
- **Java.sql.Timestamp.** A Timestamp value, which includes both date and time information. It can be sorted.
- **Java.lang.Boolean.** A Boolean value, which can be sorted.
- **Java.lang.Byte.** A Byte value, which can be sorted.

Attributes can be one of the following:

- **Number.** The default value is parsed as a Number string using the parseXXX method on the corresponding Java class. For example, use `Double.parseDouble()` if it is a double. It can only include digits and decimal point.
- **Timestamp.** You must supply the default value formatted as yyyy-mm-dd hh:mm:ss.
- **Date.** You must supply the default value formatted as yyyy-mm-dd.
- **Time.** You must supply the default value formatted as hh:mm:ss.
- **String.** The default value is used as it is.
- **Boolean.** The default value can be true or false.

<transformer> Element of Report XML

This element defines a transformer for this report. A report can include zero or more transformers. Transformer is key element of the report engine; it is responsible for transforming the data retrieved from data source into a format suitable for presentation.

The <transformer> element has the attributes described in Table 21.

Table 21. Attributes for <transformer> Element

Name	Required?	Description
Id	Yes	Uniquely identifies this transformer in this report. Note, you are allowed to have two transformers with same ID if they are from different reports.
datasourceId	No	The ID of the data source where the transformer gets data. Note, a transformer is not required to have a data source. If it does, then the Reporting Engine is usually used as a pure Template engine, and no meaningful data transformation is done inside transformer. That means that all the reporting functionality, such as sorting and paging, will not apply. For example, in the telco.xml file, the transformer with report_header.vm defined has no data source.
pageSize	No	This attribute enables paging and defines the number of rows that will be displayed in one page. All the data will be presented in one page if this attribute is not specified.

<columns> Element of Report XML

This element defines a list of columns for the transformer. You are not required to define a column in the transformer for each column in the data source. It is not necessary that the order of columns in the transformer match the order of the columns in the data source. However, following those two rules will make your code easier to maintain.

This XML element has no attribute and contains <column> elements.

<column> Element of Report XML

This XML element defines a column for the transformer. The transformer will render the columns in a table format. This is one of the most important XML element.

```
<column
  id="myColumnId"
  name="Column Name"
  Hidden="false"
  sortable="true"
  defaultSort=”true”
  caseInsensitiveSort=”true”
  pattern="MM/dd/yyyy"
  link="report.do?reportId=myReport&#x26;parameter(myColumnId)=$col"
  localize=”true”
/>
```

Table 22 lists all the attributes for the <column> XML element.

Table 22. Attributes for <column> Element

Name	Required?	Description
id	Yes	This ID must match one of the IDs defined in the data source.
name	Yes	The name of the column. The name is localized and presented as the table column name.
hidden	No	A template string; the parsing result is either true or false.
sortable	No	This attribute defines whether this column is sortable. If true, the template generates a URL link for this column. True or false; the default is false.
defaultSort	No	This attribute defines whether to sort this column when the report generates. True or false; the default is false.
caseInsensitiveSort	No	This attributes defines whether you want a case-insensitive sort when the column type is java.lang.String. True or false; the default is false.
onlineOnly	No	This attribute defines whether this column shows on the Web page only, and not in the CSV download file. True or false; the default is false.
downloadOnly	No	This attribute defines whether this column shows only in CSV download file only, and not on the Web page. True or false; the default is false.

<link> Element of Report XML

This element allows you to define a drilldown link, which can also be defined as an attribute of the <column> element. The benefit of using it as an attribute is that you can wrap the content in CDATA without escaping the special characters.

<templates> Element of Report XML

This element includes a list of template elements. It has no attributes, and includes only one element, `template`.

<template> Element of Report XML

This element defines one template used by the transformer. A transformer can define one or more templates and each template represents a presentation view. For example, you can define one template for HTML, one for XML and another for CSV. You specify which view or template to use to render the UI by passing the template ID through Ireport.writeTemplate().

```
<templates>
  <template
     id="HTML_TEMPLATE"
     name="template/common/reporting/report_body.vm"/>
</templates>
```

Table 23 describes the attributes for the <template> element.

Table 23. Attributes for <template> Element

Attribute	Required	Description
id	Yes	The ID identifies this template inside this transformer. An ID must be unique to this transformer.
name	Yes	The class path of the template name. Because the class loader loads the template by default, this template must exist on the classpath (such as on the `WEB-INF/classes` directory or packaged into a JAR file). For example, if your template is located under the `template/templ/my.vm` directory and that is on the class path, then you must use the `template/temp/my.vm` directory as the name.
localize	No	True or false. True means this template is localized. There is one template for each locale, and the report engine finds the correct template based on the locale. For example, the email template has a lot of static text, therefore define one template for each locale and specify this attribute as true to associate the correct template for each locale.

For information about creating PDF templates, see "Creating a PDF Template for Reporting" on page 124.

<groups> Element of Report XML

This element allows you to group the data retrieved from a data source into groups, where each group is presented inside a table. For example, you might want to group on all types, so that all the local calls are presented in one table, and international calls are presented in another table. Only single column grouping is supported.

You can define multiple groups. You can define one of them as default grouping, so when the data is retrieved from the data source, it will be grouped by that default grouping. Call Itransformer.group() in your calling program to switch to another group.

This element has no attributes, and can include the <group> element.

<group> Element of Report XML

This XML element defines a single group. The <column> element defines the columns you want to group on. You can only define one column. For example:

```
<group id="group_by_type" default="true">
  <column id="type"/>
</group>
```

Table 24 describes the <group> element attributes.

Table 24. Attributes for <group> Element

Name	Required	Description
id	Yes	Defines a unique ID that identifies this group in this transformer. The group ID must only be unique among the groups defined in this transformer.
default	Optional	This flag indicates that this group is the default one, so when data is retrieved from data source, the data will be grouped (only one group can be default). The data will not be grouped if there is no default group defined. The default is False.

<column> Element of Report XML

This <column> element is defined as part of the <group> element, and identifies the column where grouping will happen. It has the attributes described in Table 25.

Table 25. Attributes for <column> Element

Name	Required	Description
id	Yes	This is the column ID defined in data source. This ID must match the ID of the column of the data source where you want the grouping to happen.

<calculator> Element of Report XML

This element defines a calculator for the report. The calculator can perform a set of operations, for example: summarize (subtotal), average, Boolean and minimal. The operations are grouped together into an operation group. calculator contains one or more <operationGroup> elements. For example:

```
<calculator>
  <operationGroup name="Total">
     <operation type="sum" columnId="Charges" />
     <operation type="sum" columnId="taxes" />
  </operationGroup>
  <operationGroup name="Average">
     <operation type="ave" columnId="Charges" />
     <operation type="ave" columnId="taxes" />
  </operationGroup>

</calculator>
```

For example, the Reporting Engine generates a table similar to the example in Table 26.

Table 26. Example of Table Generated by the Reporting Engine

Invoice Number	Charges	Taxes
12345	10.01	0.23
23456	12.11	1.03
Total	22.12	1.26
Average	11.06	0.63

<operationGroup> Element of Report XML

This element defines a group of operations. Different operations in the group must operate on different columns, but it is not required they have the same operation types. That is, you can mix sum with avg in the same operation group.

In general, do not define an operation on the first visible column of the table; that column will be used to display the name of the operationGroup. However, if necessary to define an operation on the first visible column, you can change the report_body.vm by replacing the operationGroup name with the operation value you define.

The <operationGroup> element has one attribute, name, which is described in Table 27.

Table 27. Attributes for <operationGroup> Element

Name	Required	Description
name	Yes	The name of this group of operations. The default template, report_body.vm, presents it as the first column of the operation row of the table.

This element can contain one or more <operation> elements.

<operation> Element of Report XML

The <operation> element defines a single calculator operation on a single column. It has the attributes described in Table 28.

Table 28. Attributes for <operation> Element

Name	Required	Comments
type	Yes	The type of operation: ■ **summary**. Finds the summary of all the values of the column identified by columnId attribute. ■ **avg**. Finds the average of all the values of the column identified by columnId attribute. ■ **max**. Finds the Boolean value of all the values of the column identified by the columnId attribute. ■ **min**. Finds the minimal value of all the values of the column identified by the columnId attribute. **count**. Finds the total number of rows. In this case, columnId is optional.
columned	Yes	The ID of the column that the operation will apply to.

<charts> Element of Report XML

This element allows you to define one or more charts for a single transformer. For example:

```
<charts>
  <chart id="c1"
     type="BAR_VERT_CLUST"
     style="config/chart/vertical_bar_chart.properties"
     chartTitle="global.title.accountBillingOverview"
     xAxisTitle="global.label.accounts"
     yAxisTitle="global.label.dollars">
     <datasets>
     <dataset><column id="Total"/></dataset>
     </datasets>
     <xlabel><column id="Billing_Account"/></xlabel>
  </chart>
  <chart id="c2"
     type="PIE"
     style="config/chart/pie_chart.properties"
     chartTitle="global.title.plan">
     <datasets>
     <dataset><column id="total"/></dataset>
     </datasets>
     <xlabel><column id="rate_plan"/></xlabel>
     <compress threshold="2" label="global.label.other" append="true"/>
  </chart>
</charts>
```

<chart> Element of Report XML

This element defines a single chart for this transformer. Oracle Self-Service E-Billling supports two chart types: Bar chart and Pie chart. The data of the chart must come from the columns of the data source.

The <chart> element includes the attributes described in Table 29.

Table 29. Attributes for <charts> Element

Name	Required	Description
id	Yes	Uniquely identifies this chart among all the charts defined in this transformer. Note, you can use the same chart IDs in different transformers.
type	Yes	The type of the chart. Oracle Self-Service E-Billling supports the following types of chart: ■ **BAR_VERT_CLUST**. Vertical bar chart. ■ **BAR_HORIZ_CLUST**. Horizontal bar chart. ■ **BAR_VERT_STACK**. Vertical stack bar chart. ■ **BAR_HORIZ_STACK**. Horizontal stack bar chart. ■ **PIE**. Pie chart. ■ **LINE**. Line chart.
style	Yes	Path to the name of the DVT chart properties file. For information on configuring the properties file for DVT charting, see "Customizing Charts" on page 143.
chartTitle	No	Defines the title of the chart.
xAxisTitle	No	The title of the X-axis.
yAxisTitle	No	The title of the Y-axis.

The <chart> elements also include following two elements: <datasets> and <xlabel>.

<datasets> Element of Report XML

This element allows you to define multiple data sets used to draw the chart. Only one dataset for each chart is supported.

<dataset> Element of Report XML

This element defines a data set used for charting. A data set must come from the column of the data source. Currently, you can only define one column for on dataset. It has no attributes and contains one element: <column>.

<column> Element of Report XML

This element defines the column whose values will be used as the data set for DVT charting. For example, for the BAR_VERT_CLUST chart, the dataset is used for the Y-axis values; for PIE, the dataset is used for the pie chart data.

The <column> element of Report XML includes one attribute, which is described in Table 30.

Table 30. Attributes for <column> Element

Name	Required	Description
id	Yes	The ID of the column where the chart will get its data. The type of the column must be a number.

<xlabel> Element of Report XML

This element defines the values for the x-axis. The x-label must come from the data source column. It has no attributes, and contains one element: <column>. You can only define one column for each x-label.

<column> Element of Report XML

This element defines the column used for the x-label. The values of the column are used for the x-axis values. This element only includes one attribute, which is described in Table 31.

Table 31. Attributes for <column> Element

Name	Required	Description
id	Yes	The ID of the column where the chart will get its x-axis values.

<downloadList> Element of Report XML

This element defines a list of downloads available for this report. For example, you can define XML, CVS, and PDF downloads. For each download, the template generates a download link. You can define multiple downloads for one report. For example:

```
<downloadList name="Download">
  <download
  name="Download CSV"
  type="csv"
  description="CSV download"
  templateId="CSV_TEMPLATE" />
</downloadList>
```

The <downloadList> element has one attribute, which is described in Table 32.

Table 32. Attributes for <downloadList> Element

Name	Required	Description
name	No	The name of this downloadList. Depending on your template, you can use this name for different purposes. For example, you can build a list of downloads and use this name as the name of the list.

<download> Element of Report XML

The <download> element defines one download for the report. It has the attributes described in Table 33.

Table 33. Attributes for <download> Element

Name	Required	Description
type	Yes	The type of the download. You can name any type you want. The type is used as the download file extension. For example, use csv for CSV download and use xml for XML download.
name	No	The name of the download. Depends on the template; it can either be shown as a URL link or as a list item.
description	No	Description of the download. Currently it is not used by template, but you can modify template to use it for a pop-up help window
templateId	Yes	The template ID used to generate the download of the report. It is possible that the same template ID list appears in multiple transformers. If so, the templates will be parsed and appended together in the order of the templates defined in XML

<printList> Element of Report XML

This element defines a list of print-friendly available for this report. Though it is possible, it is unlikely you will define more than one print-friendly. For each print friendly, a print friendly link will be generated through the template.

For example:

```
<printList name="Print friendly">
  <print
     name="Print friendly"
     description="print friendly account details"
     templateId="PRINT_TEMPLATE" />
  </printList>
```

The <printlist> element has one attribute, which is described in Table 34.

Table 34. Attributes for <printList> Element

Name	Required	Description
name	No	The name of this printList. It is not used by current template

<print> Element of Report XML

The <print> element defines one print-friendly for the report. It has the attributes described in Table 35.

Table 35. Attributes for <print> Element

Name	Required	Description
name	No	The name of the print-friendly. The default template renders it as a URL link.
description	No	Description of the print-friendly. Currently it is not used by template, but you can modify the template to use it for a pop-up help window.
templateId	Yes	The template ID used to generate the print-friendly report. It is possible that the same template ID can appear in multiple transformers, so all these templates will be parsed and appended together, in the order of the templates defined in XML.

<customList> Element of Report XML

This element defines a list of custom reports available for this report. Though possible, it is unlikely that you must define more than one custom report. For each custom report, a custom report link will be generated through the template.

For example:

```
<customList name="Customize">
  <custom
     name="Customize"
     description="Create a custom report for contract call details"
     reportId="telco_cust_std_r4" />
  </customList>
```

The <customList> element has one attribute, which is described in Table 36.

Table 36. Attributes for <customList>

Name	Required	Description
name	N	The name of this customList. It is not used by current template.

<custom> Element of Report XML

The <custom> element defines one custom report for the current report. Each custom report must be itself defined as a report. This tag is used to build a link to that custom report. It has the attributes described in Table 37.

Table 37. Attributes for <custom> Element

Name	Required	Description
name	No	The name of the custom report. The default template renders it as a URL link.
description	No	Description of the custom report. Currently it is not used by template but you can modify template to use it for a pop-up help window.
reportId	Yes	The report ID of the report used to define the custom report: the custom report itself is a report and you must define it as a report.

Using Report Templates

All the report UIs are generated through Velocity templates. For information about how the Velocity templates work, see:

http://jakarta.apache.org/velocity/index.html

Oracle Self-Service E-Billling has changed some of the default Velocity templates. The most important one is that inside for each loop, the $velocityCount variable starts from 0 instead of the default 1.

Oracle Self-Service E-Billling offers a set of example templates that generate useful UIs. These templates are very generic, are not tied to a particular application, and can be used as the base for your customization work.

The templates are all defined in the *EDX_HOME/*`template/`common directory, where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billling.

The lib subdirectory includes some Velocity MACRO library files and the reporting subdirectory includes report template files.

Table 38 explains the libraries that are included with the report package.

Table 38. Libraries Included with Reporting Package

Name	Description
Lib/report_library.vm	This file defines some common MACROs used by the Reporting Engine. You must use it as it is.

Table 39 explains the templates that are included with the report package.

Table 39. Templates Included with Reporting Package

Name	Description
Common/report_header.vm	This is the header part of the report. Note, this is not the header of the tiles. The tile header is usually the Navigation Tabs. The report header usually includes the report name and the download, print friendly, and custom report links.
Common/report_body.vm	This template renders the table associated with the transformer. Because a report can define multiple transformers, the template can be parsed multiple times for a report.
Common/paging.vm	This template renders the paging navigation part, which has previous, forward buttons for a user to page through the report.
Common/layout.vm	This template is used to define the layout of the prompts of the report.
Common/promt.vm	This template renders each individual prompt of the report.
Common/promt.vm	This template renders each individual prompt of the report.
Common/csv.vm	This template renders the CSV format of a report. The current CSV format does not consider the case of how to escape the special characters like a comma. You must write code to handle that case.
Common/print.vm	This template renders the print friendly format of a report.
Common/custom_report.vm	The template is used for custom report. It displays the custom report detail and allows you to type a name for the report to save into database.
common/batch_report.vm	The template is used for batch reporting. It displays the batch report detail and lets you type a name for the report to save into database.
common/xml.vm	This template renders the XML format of a report.
pdf/PrintSummary.rtf	This template renders PDF format for the Print Summary quick link on B2C pages.
pdf/StatementSummary.rtf	This template renders PDF format for the Statement Summary report.
pdf/telco_std_r1.rtf	This template renders PDF format of the Account Billing Overview report.
pdf/telco_std_r6.rtf	This template renders the PDF format of Total Cost by Plan report.
pdf/telco_std_r13.rtf	This template renders the PDF format of Service Details report.

Creating a PDF Template for Reporting

Oracle Self-Service E-Billling provides preconfigured PDF templates for reporting (listed in Table 39 on page 123). You can also define additional templates to present other reports in PDF.

To create a PDF template for reporting

1 Download and install Oracle Business Intelligence(BI) Publisher:

   http://www.oracle.com/technetwork/middleware/bi-publisher/downloads/index.html

2 Download the existing or new report in XML format and use this file to create a new PDF template (an RTF) file. See the Template Builder for Word Tutorial in Oracle BI Publisher for assistance creating the RTF template. Training is also available on the Oracle Technology Network.

3 Using the toolbar in Oracle BI Publisher, load the report XML file.

4 Use Microsoft Word formatting and Oracle BI Publisher insert functions to add fields, tables, charts, and conditional formatting to the template appropriate for your organization. Save the file as an RTF.

5 After you create the template file *filename*.rtf (where *filename* is the name of the report template), copy the file to *EDX_HOME*/template/pdf/*filename_ll_CC*.rtf, where *ll_CC* is your default system language code, from the EDX_SYS_LANG table in the OLTP database.

6 If the application supports multiple languages, you can translate your template and name it using the different language code. For example, if you support Spanish (es_ES) as well as American English (en_US), create a template called *filename*_en_US.rtf and create a translated template called *filename*_es_ES.rtf.

7 In *EDX_HOME*/config/rpt/*.xml, add the following line in <downloadlist> tag for the new report:

```
<download name="global.dropdown.pdf" type="pdf" description="PDF download"
templateId="PDF_TEMPLATE"/>
```

8 Add the following line in the <templates> tag for the report, where *ll_CC* is the default language:

```
<template id="PDF_TEMPLATE" downloadable="true" name="template/pdf/
filename_ll_CC.rtf"/>
```

   NOTE: You do not need to specify additional languages in the XML. Oracle Self-Service E-Billling finds the template that corresponds with the selected language.

Predefined Context Variables

When you call the IReportActionHelper.execute() method to generate reports, the reporting engine puts a list of predefined context variables into the report context, which are then available to the Velocity template.

Table 40 lists some of the variables that you can use. If the overwrite flag is Y, then you can pass a variable with the same name through ReportContext to overwrite the default values set by ReportActionHelper.

Table 40. Predefined Context Variables

Name	Type	Over write?	Description
form	Action Form	No	This is the struts ActionForm object currently being processed.
gifDir	String	Yes	The directory where the image files used by report are saved, for example, the paging arrow images. It is default to "_assets/ images.
link	String	No	This is the URL link base of this page and it is equivalent to the html <base> tag. The default value is: HttpServletRequest.getContextPath() + HttpServletRequest.getServletPath() The URL is similar to the following: http://host:port/<web-root>/report.do
user	IUser of UMF	Yes	The current user logged in. IUser is passed in as a session variable, USER_PROFILE. If it is not in the session, Oracle Self-Service E-Billling does not put it in the context. User is just used for query purposes and its absence does not affect the functionality of the reporting. For example, for some reason, you might not use UMF IUser and you can use your own user object. **NOTE:** Some templates, like report_header.vm, might expect IUser to get user name and if you do not supply IUser, the template might not display user name properly.
contact Profile	IContactProfile of UMF	Yes	The contactProfile is a profile of IUser named as contact_profile. Oracle Self-Service E-Billling uses it to retrieve the user's first name and last name and is currently only used in report_header.vm. The absence of this information does not affect the function of Reporting Engine.

Table 40. Predefined Context Variables

Name	Type	Over write?	Description
locale	String	Yes	The default value set by ReportActionHelper is the from http session: `session.getAttribute("org.apache.struts.action.LOCALE")`. **NOTE:** This locale is put into session by the Struts framework.
reportId	String	No	The report ID of current report.
transformer	ITransformer	No	You can use transformer object to do work such as formatting data. Never call `ITransformer.writeTemplate()` in the template.
reportConfig	IReportConfig	No	Represents the report configuration.
dataSourceConfig	IDataSourceConfig	No	Represents the data source configuration.
dataSource ColumnConfigs	A list of IDataSource ColumnConfig	No	Represents the list of data source column configurations.
transformerConfigs	ITransformerConfi g	No	Represents the transformer configuration.
transformer ColumnConfigs	A list of ITransformer ColumnConfig	No	Represents the list of transformer column configurations.
operationGroup Configs	A list of IOperation GourpConfig	No	Represents the list of operation groups defined inside calculator for the transformer.
chartConfigs	A list of IChartConfig	No	Represents the list of chart configurations for the transformer.
templateConfigs	A list of ITemplateConfig	No	Represents the list of template configurations for the transformer.
rowlist	IReportList	No	Represents the original data retrieved from the data source. The data could be sorted and so the order could be changed. Though you cannot overwrite this variable, you can certainly change the content of the list.

Table 40. Predefined Context Variables

Name	Type	Over write?	Description
groupSet	Set	No	To support grouping, the transformer maintains a Map of List objects. In the case of no grouping, there is only one entry in the map, the key is the report name, and the value is the List returned from data source; In the case of grouping, the original list from the data source is regrouped into multiple lists. Each list has the same group value, and the group value becomes the map key. This variable is looped through in report_body.vm to build the HTML table.
dataMap	Map	No	This is the map of group keys to the List as described previously.
reportContext	ReportContext	No	The ReportContext object used to generate reports. **NOTE:** Note: You cannot overwrite reportContext, but you can change the content.
URLEncoder	URLEncoder	No	This is a wrapper class around java.net.URLEncoder, because Velocity cannot invoke a static method directly through class name, and java.net.URLEncoder does not have a constructor. Use this class to encode the parameter values you passed through the URL.

Integration with Struts and Tiles

The Reporting Engine can be used with any presentation framework. However, because Oracle Self-Service E-Billling is based on Struts and Tiles, the Reporting Engine has special extensions to help it integrate with Struts and Tiles. This topic describes that integration.

Struts Action Class

The Struts action class does following processing:

```
ReportContext ctx = new ReportContext()
ctx.put(…) //put whatever your stuff used in template
IReportActionHelper helper = ReportManager.getReportActionHelper()
IReport report = helper.execute(ctx, form, request, response); //IReport will be in
session

return mapping.findForward(" page.reports.report ");
```

It creates a ReportContext object which you can put your own objects into. These objects can then be used in report templates. Then it calls IReportActionHelper.execute() method to get an IReport object. If this is the first time to access the report, a new IReport object will be created; if this is a sorting or paging operation, the IReport object cached in the session will be returned. In case a new IReport object is necessary, the report data will be retrieved from the DataSource defined in the report XML of this reportId.

Next it calls IReportActionHelper.execute() method to get an IReport object. If this is the first time to access the report, a new IReport object will be created; if this is a sorting or paging operation, the IReport object cached in the session will be returned. In case a new IReport object is necessary, the report data will be retrieved from the DataSource defined in the report XML of this reportId.

For the last action of this class, control is forwarded to the tile, page.reports.report, which is defined in the tiles definition file.

Tiles Definition

Tiles are defined in the ebilling-tiles-defs.xml file in the WAR file of the EAR file.

```
<definition name="page.reports.report" extends="simpleLayout_1">

        <put name="pageName" value="Billing Report"/>

    <put name="leftBar" value="/_includes/sidebar_left_analytics.jsp"/>

        <put name="pageDesc" value=""/>

     <put name="header" value="/_includes/header_analytics.jsp"/>

        <put name="footer" value="/_includes/footer_relative.jsp"/>

        <put name="subtab" value="/_includes/subtab_billing.jsp"/>

        <put name="body" value="/reporting/report.jsp"/>

    </definition>
```

The key to this tile is that the body tile is report.jsp, which generates the main body of reporting UI.

Report.jsp

The report.jsp page is used to render the view. In fact, there is almost no HTML code in this page. Instead, this page just invokes the Velocity template engine to parse the templates:

```
IReport report = (IReport)request.getSession().getAttribute(reportId);
```

```
IReport.writeTemplate(jspWriter, templateId);
//template is the one defined in report xml and default to “HTML_TEMPLATE”
```

The Reporting Engine goes through the Transformers defined in the report XML for this reportId and for each transformer, parsing the template whose ID matches templateId. Note a transformer will be ignored if it has no template with a matching templateId defined in the transformer configuration of the report XML.

The matching templates will be parsed in the same order as defined in the report XML, and the results will be written back into JSPWriter sequentially.

Reporting API

The reporting API offers an interface to interact with the Reporting Engine. These APIs manage common reporting features, such as sorting, grouping and paging. They also offer report clients the flexibility to customize reporting.

The reporting API is not tied to a particular presentation framework; you can use struts and tiles or servlets and JSP to access it. However, you could find that using struts and tiles is the easiest way to implement your own reporting UI, because that is the default presentation framework used for the reporting UI of Oracle Self-Service E-Billling.

The core reporting APIs are: ReportContext, IReportManager, IReport, ITransformer, IReportConfig and ReportActionHelper. For more information about reporting APIs, go to the Oracle Self-Service E-Billling Javadoc as described in “Accessing Oracle Self-Service E-Billing Javadoc” on page 31.

ReportContext is the carrier of information between the reporting caller and the Reporting Engine. ReportManager is a factory that gets an instance of IReportManager. IReportManager is the factory for IReport objects. IReport represents a report defined in XML. ITransformer represents the transformer defined inside a report in XML. IReportConfig represents the configuration information in XML.

The following example shows how to generate a report:

```
ReportContext context = new ReportContext();
context.put(“form”, StrutsActionForm);
IReportManager rptmgr = ReportManager.getInstance();
IReport rpt = rptmgr.getReport(“reportId”, context);
Rpt.writeTemplate(“templateId”, Writer);
```

In the example, a Struts ActionForm is put into the reportContext, which means this object is available to the Velocity template. You can use the syntax of $form.name in the Velocity template; assume there is a name property in the form.

After you get an instance of IReportManager, call its getReport method to get a report. The reported must match the one defined in report XML. It will return an object that represents the report defined in the XML with the same reportId.

After you get an instance of IReport, it calls its writeTemplate() method to parse the Velocity template identified by templateId in the report XML, and writes the content into a Writer output. This method loops through all the transformers in the report and calls transformer.writeTemplate(). If the same template IDs appear in different transformers, then multiple templates can be parsed and the content of the parsed templates will be appended together in the order in which they appear in the report configuration XML.

You can also call the individual APIs of ITransformer to do sorting, grouping or paging.

However, it is tedious to call these APIs: they are usually used for back-end based applications. For the common UI features, such as sorting, grouping, and paging, the reporting API offers a Web helper class, ReportActionHelper, to shield you from the low-level APIs. This class is a facade to the Report Engine APIs. In most cases, your struts action must call this helper class instead of calling the lower-level reporting APIs. However, you can always access the report APIs directly if you want to. The action used by the product, Com.edocs.app.reporting.actions.ReportAction, calls this helper class. You can similarly do this in your action class.

Core Reporting Features

This topic describes some of the most important features of Reporting Engine, and how to use them in your application.

Sorting Feature of the Reporting Engine

Sorting is a built-in feature of the report engine. It is available when you use the ReportActionHelper class from your action class. With the reporting XML and template, enabling sorting is like configuring a transformer’s column. For example:

```
<column sortable=”true” …/>
```

Only single column sorts are supported. The sorting is done in-memory, to eliminate accesses to the data source.

Set the column attribute sortable to true. The Reporting Engine reads the configuration, instructs the template to generate a sort-able link for the corresponding table column name, and the ReportActionHelper class calls the ITransformer.sort() API.

When a column is defined as sort-able, the report_body.vm template renders the column of the HTML table with a URL link. For example:

```
<a
href="$link?sortColumn=$x&reportId=$reportId&transformerId=$transformerConfig.id&c
urrentSortColumn=$currentSortColumn&ascending=$ascending&currentGroup=$groupIndex"
>
```

Table 41 describes the parameters in the URL.

Table 41. Sorting Parameters in the URL

Parameter	Description
$link	The URL context base, which is set to http://host:port/<web-context>/report.do in ReportActionHelper class, where <web-context> is the Web context you defined in your EAR file.
SortColumn=$x	This is the column index of the column being sorted in the transformer configuration.
Reported=$reported	The report ID of the report.
TransformerId=$tranformerConfig.id	The ID of the transformer currently being sorted
CurrentSortColumn=$currentSortColumn	This is the current column being sorted in this transformer.
Ascending=$ascending	Defines the sort order, true or false.
CurrentGroup=$groupIndex	Not used but can be used for grouping.

The Web component must process the URL request, and calls the ITransfomer.sort() method to sort the column. The Helper class, ReportActionHelper does this work for you.

Just call the ReportActionHelper in your struts action. It processes this request and calls Itransfomer.sort() to sort the column, then reorders the newly sorted report for you.

Paging Feature of the Reporting Engine

Paging is a built-in feature of the Reporting Engine. Use the ReportActionHelper class and the default templates (or templates based on the defaults) to access that function. The main paging template is paging.vm, which is included in report_body.vm.

Paging is enabled when:

- you specify pageSize for transformer in report XML.
- <transformer <pageSize=”20” />

Because reports are loaded and cached in the user session, paging is done on cached data. This method of paging does not scale when there are a large number of rows of data. For that case, you must limit the number of rows retrieved using the maxRows attribute of the <query> element.

Dynamic SQL

Some situations require you to generate SQL dynamically. For example, you might have a report that searches the call details. One of the criteria is the call date. You want to search for call date equals a particular date, or you want to search for call dates between a start date and end date. Because the where clause is different for these two search cases, without dynamically generated SQL, you would be forced to write two reports with two SQL clauses. Dynamically generated SQL can solve this problem; the where clause of the SQL statement can be generated based on the current operation (equal or between), so only one report is required.

The Reporting Engine allows you to write an SQL query in a Velocity template, so that the SQL query will be parsed before it is executed. You must set the dynamic attribute of <query> to true. For example:

```
<query dynamic=”true”> <![CDATA[
  select * from my_table where #if ($equal) date = ? #else date >= ? and date <= ?
#end
]]></query>
```

The variable $equal is set by the caller through the IreportActionCallback interface. It is true if the user selects the date equal operation, and false if the user chooses the date between operations.

NOTE: The number of question marks is different based on operation types: one for equal and two for between. To solve this problem, the report engine supports binding a Collection object to question marks. The report engine loops through the Collection and binds each element to question marks.

The following example shows how to bind:

```
<inputBinding object=”form” property=”parameter(dateList)” />
```

The method form.getParameter(dateList) returns a list of Date objects, and each date in the list is bound to the question marks in the query. The caller of Reporting Engine is responsible for collecting the list of dates and passing them to ActionForm.setParameter(dateList, dateList). (This assumes that ActionForm is as map-backed form, and has a pair of setParmeter and getParameter methods).

Another common use case is to generate the IN operation in a WHERE clause. The number of question marks is based on the size of a Collection object.

For example, if you have a list of categories saved in a List, and want to generate a where clause, use

```
Where category in (?,?,..,?,?)
```

In this clause, the number of question marks is the size of the List.

When doing the input binding, there is only one List, but loops through the List to set the question marks in the SQL as appropriate. This ensures that the number of question marks match the number of variables passed in.

There is a macro to help you generate the number of question marks based on the collection size:

```
#macro getSQLVariablesIgnoreNull($list $columnName)
```

The macro generates the list.size() number of question marks. For example:

```
select * from my_table where date in getSQLVariablesIgnoreNull($dateList “date”)
```

If the dateList size is 2, and it is Oracle database, then the result is:

```
select * from my_table where date in (NVL(?,date), NVL(?,date))
```

In this clause, NVL means ignore this question mark if the value is null.

Internationalization and Localization of Reporting

Resource bundles are used to support internationalization. This topic discusses internationalization for Velocity templates.

Because the Reporting Engine uses Velocity templates, you cannot take advantage of the JSP <message> tag or the Struts internationalization framework. Instead, the Reporting Engine has its own internationalization mechanism specially designed for Velocity templates, which has following features:

- Allows a user to specify any resource bundle, just like Struts `config` does.
- Allows a user to format a string as Y does, for example, My name is {0}.
- Provides a seamless integration with Struts if it is used. For example, sharing the same resource bundle.
- Offers a better way to handle default messages than Struts. In Struts, a resource that is not found is either returned as null or as ???<locale><resource_key>???. With the Reporting Engine, you can configure it to return the key itself when the value of the key is not found.

For internationalization of reporting, you translate the following text:

- Regular text on the report user interface.
- Some text coming from the data source.
- Chart, title and amount format, and so on.
- Date format, number format, and so on.

Resource Bundle Definition

The resource bundle files used by the Reporting Engine templates are defined in the report XML files under the <reports> tag. The following example comes with Oracle Self-Service E-Billling, and is defined in the telco_global.xml file.

```
<localizer enableMessageResources="true" defaultCode="1">
  <resourceBundle name="com/edocs/app/reporting/resources/ApplicationResources" />
</localizer>
```

Follow these naming guidelines for localizing resource bundles:

- You must use “/ instead of “.” in the name of the resource bundle, which differs from Struts message resource.
- The <localizer> tag defines how text will be localized. You can define multiple <resourceBundle> tags. Each resourceBundle tag defines a resource bundle file, and its name is defined by name attribute.

When the Reporting Engine searches for the resource bundle, it first checks whether this bundle exists as a file under *EDX_HOME* (the directory where you installed Oracle Self-Service E-Billling), or the current directory if *EDX_HOME* is not defined. If that fails, it will try to find it as a class.

The attribute enableMessageResources enables you to use Struts MessageResource to search for a resource.

The attribute defaultCode enables you to define the default behavior if a resource is not found. 0 means to use the key as the default value; 1 means to use Struts notion of "???<locale>.<key>???" and -1 means throw an exception. The default value for the attribute defaultCode is 0.

The search order for finding a resource is:

1. If the attribute enableMessageResources is true, and the Struts MessageResource does exist (it might not exist for a non-Struts application), search the resource from Struts MessageResource, and return the resource if it is found.
2. For each resource bundle defined in resourceBundle, load the bundle as either file or class, and then search the resource in the order it appears, return if found.
3. If nothing is found, use defaultCode described previously.

Follow these guidelines when defining a resource bundle:

- If you check the resource bundle name in the struts configuration file, you will notice that the same file, `com/edocs/app/reporting/resources/ApplicationResources`, is defined in both the Struts and report XML files. The only difference in the definitions is the file separators; reporting uses a back slash (/) and Struts uses a period (.). The same file is in two locations in order to support batch reporting. A batch job is not a Web application, so it does not have access to Struts MessageResource. This is also true if you are using the Reporting Engine at the EAR level. For example, you can generate an email message from an MDB event handler or from an EJB. However, if you are using Struts, and you using the Reporting Engine for online applications only (not batch reporting), then do not define a resourceBundle, because the online Web application can always find resources from MessageResource.
- Because the same resource is defined twice, both Struts and the Reporting Engine load the same resource bundle and cache them (twice). Usually, this is not a problem, because a resource bundle file is small. However, if you do want to reduce memory usage, you can put all the template related resources into one file. Or, you can be more selective by putting only the batch report, email, and AR-related resources into one file, and load it by using the resourceBundle tag in report xml.
- It is recommended that you define the resource bundle as flat file under *EDX_HOME*, which lets you modify the file and reload it using this URL without restarting:

  http://localhost:7001/ebilling/reporting/reloadReportConfig.jsp
- If you want to use a struts message source, which is loaded from the classpath, you can disable it during the development stage by setting enableMessageResource to false and loading a resource bundle from file system.
- Set the defaultCode to 1 to find all the text not being internationalized properly. You might want to set it to 0 for demonstration purposes.

Localization of Report Text

The localization of text in report is done through the #localize macro, which is defined in reporting_library.vm. It is defined as:

```
#macro (localize $name)
```

For example, in your template, you can call this macro as follows. This expression searches the report bundle to find a key with a value that matches name:

```
#localize(“name”)
```

All the texts defined in the report.xml file are treated as resource bundle keys. For example, report names and column labels. In the report template files, all the texts are localized through the #localize macro.

Localization of Report Data from a Data Source

By default, the text data retrieved from data source is not localized. You must turn on this option. In this case, the text data from data source will be used as keys to search reporting resource bundles.

The localization of data from data source is done through the localize attribute of transformer column configuration in report XML.

```
<column id=”call type” localize=”true” />
```

The column data retrieved from the database will be localized.

Localization of Charts

The chart components (chart title, labels and data) are localized by the ITransformer.writeChart() method. The chart tile is searched as a regular resource bundle name. Label and data are localized if the localize attribute is set to true for the corresponding columns.

Locale

To support internationalization, you must pass the Locale object to ReportContext by calling setLocale(). If ReportContext does not have a locale defined, when you call the IReportActionHelper.execute() method, it puts the Struts locale object in session.

Dynamic Localization

Velocity is used to support localization. Velocity acts similar to the way java.text.MessageFormat does, and achieves the same result. The Reporting Engine parses the resource value as a Velocity template, whose resource key ends with .vm, and returns the parsed value. For example,

```
rpt.test.vm=My name is $name.
```

Object name must come from the report context. This feature can make any text in your report dynamic. For example, if you are on the account detail page, to display the report tile as Account detail for <account_number> instead of the default text, define the report title as a .vm resource bundle. In the following example, accountNumber is from the Struts ActionForm:

```
rpt.accountDetail.title=Account detail for $form.accountNumber
```

Object Data Source

Because you might not have access to the database, the Reporting Engine provides an API to get back a list of Objects, which can be presented in a table with paging or sorting. The Reporting Engine offers an Object data source to provide that feature.

The object data source is defined as:

```
<dataSource id="ds1" uri="object:reportList">
    <columns>
        <column id="id" type="java.lang.String"/>
        <column id="name" type="java.lang.String"/>
    </columns>
</dataSource>
```

This example states that there is an object called reportList in ReportContext, and you must put that object into ReportContext before calling IReportActionHelper. This object can either be a List (java.util.List), List of objects, or a List of JavaBean Objects. If the object is a List of List of objects, then it is assumed that the objects in the inner list are basic Java objects, such as String or Integer. The objects must also match the types defined in the dataSource column.

Usually, the object is a List of JavaBean objects. For example, as shown in the example XML, reportList is a List of IReportConfig objects. (For more information about APIs, see the Oracle Self-Service E-Billling Javadoc as described in "Accessing Oracle Self-Service E-Billing Javadoc" on page 31.) The Reporting Engine uses reflection to get the property values of the JavaBeans, whose property names match the column IDs defined in the example XML, and converts this List of JavaBeans into a List of Lists of JavaBean property objects (more precisely, into a IReportList of IReportRow objects). It is also assumed that the JavaBean properties are basic Java types. In the example, for each IReportConfig object in the list, the report engine calls IReportConfig.getId() and IReportConfig.getName(), and converts the List of IReportConfig objects into an IReportList object. Each element in IReportList is an IReportRow object. Each IReportRow includes two elements, the report IDs and the report names.

Then define the rest of the report XML, including transformers, as usual.

The object data source enables the Reporting Engine to connect to other data sources currently not directly supported by the Reporting Engine. For example, you might have a CORBA interface that retrieves financial data from a legacy system. You can still use the report engine to present the data, as long as you can convert the data into a List of Lists of objects.

DSV Data Source

This feature allows you to read a delimiter separated string as a data source. The URI format of this data source is as follows:

```
"dsv:inline:,:|"
```

In the data source, dsv stands for Delimiter Separated Values; inline means that the data can only be embedded in the report XML (support is not available for reading data from a file); the comma is the column separator, and | is a line separator, as shown in the following example:

```
<dataSource id="ds" uri="dsv:inline:,:|">
     <query><![CDATA[0,Business|1,Personal]]></query>
     <columns>
            <column id="value" type="java.lang.Integer"/>
            <column id="name" type="java.lang.String"/>
     </columns>
</dataSource>
```

The data source will be transferred into an IReportList with two IReportRows. The first row has values of 0 and Business and the second row has values of 1 and Personal. You can use this data source to implement the split-billing feature. For example, you can generate a list for call details and allow the user to change a call from personal to business or conversely.

Drilldown and Breadcrumb Link

The Reporting Engine allows you to build a breadcrumb link while you are drilling down from report to report.

To build drilldown link, define a <link> for a transformer column:

```
<report id=”testrpt0”>
  <transformer id=”tr1” dataSourceId=”ds1”>
  <column id="invoice_number" name="Invoice number"  >
  <link title="Drill down to the invoice detail."><![CDATA[

report.do?reportId=testrpt1c&invoiceNumber=$row.get(1)&parameter(parentNode)=root
  ]]></link>
  </column>
  </transformer>
</report>
```

The <link> element instructs the Reporting Engine to build a drilldown link for each account number. You must construct the link, which must point to another report. The link will be parsed as a Velocity template.

This link also has a title attribute, which allows you add an HTML title to the link. In most browsers, the title will be displayed as popup help.

When you click an account number, you will drilldown to testrpt1 report. However, by default, there is no breadcrumb link built to allow you to go back to the testrpt0 report. To enable the breadcrumb link, add enableDrillUp=true to the column definition:

```
<report id=”testrpt0”>
  <transformer id=”tr1” dataSourceId=”ds1”>
    <column id="invoice_number" name="Invoice number" “enableDrillUp”=true >
  link title="Drill down to the invoice detail."><![CDATA[

report.do?reportId=testrpt1c&invoiceNumber=$row.get(1)&parameter(parentNode)=root
  ]]></link>
```

```
        </column>
      </transformer>
    </report>
```

When this flag is set to true, and you drilldown from testrp0 to testrpt1, there will be a breadcrumb link in the testrpt1 view which allows you to go back to the testrpt0 report.

Currently, you must drill down from one report ID to another report ID, but the breadcrumb link will not work if you try to drilldown to the same report. This feature makes sense when you are viewing the same report but drilldown through hierarchy.

Customizing the Reporting Engine

This topic describes how to customize the Reporting Engine. The examples use Struts and Tiles for the presentation framework, but the same techniques can be used for any other Web presentation framework.

You might want to customize the Reporting Engine to add the following features:

- Write your own Report XML
- Modify report templates
- Extending Reporting Engine through Reporting API

Write Your Own Report XML

The first step in creating your own report is to create your own report XML. Each report XML is project-specific. The best way to start is to use existing report as a base for your modifications

CAUTION: The Reporting Engine has a DTD, but is not used to validate the report XML. Therefore, make sure you do not to miss required attributes or XML elements.

You can create one report XML, which includes all the reports for your project, or you can create one XML file for each report. Remember to register all your report XML files in the reportList.properties file, and to give each XML file a unique name.

After creating your own report XML you can test it through the default template. Name your report ID with a prefix of telco_std, which will cause it to be loaded into the standard billing report list of Oracle Self-Service E-Billling

CAUTION: Make sure that each report has a unique name across all the reports in all report XMLs, or else a latter one will overwrite the previous one.

Customize the Report Template

After you have created a report XML and familiarize yourself with how the report engine renders the report, you might want to customize the report template to generate the look and feel of your project.

A set of templates are provided with the report product. To customize them, make a copy of each template, put it into a new template directory, and change your report XML to point to the new directory.

You can add new objects into the report context (and thereby, the Velocity context) using the IReportActionCallback interface. But do not to overwrite the existing context variables. One technique is to use a special prefix (a underscore character, for example) for your custom context variables.

The CSS for the reporting HTML is defined in a file called skin.css. You can modify this file to change the CSS of the report UI.

Write Your Own Action Classes and ReportForm

Write your own Action class and action form for your reports. Use the ReportActionHelper class to take care of common issues such as sorting and paging.

When writing your own action class, you must call the ReportActionHelper.execute() method. See "Integration with Struts and Tiles" on page 127 for details about how to invoke this method.

When defining your own Struts ActionForm, you can make the form map-based, which allows you to pass any parameter into the Reporting Engine without explicitly adding a set of get and set methods. The only downside to this method is that a map-based property cannot be passed into JavaScript for client side validation.

For example, you can define two map methods: public Object getParameter(String name) and void setParameter(String name, Object value). To use these parameters in an HTML form or URL, use a notion similar to the following:

```
"parameter(contractNumber)=123456"
```

This expression passes the contract number to struts, which calls setParameter() on your ActionForm to put the contractNumber into the map. This parameter can either be used as an SQL data source input binding or used in template.

To retrieve the parameter as an inputBinding, use:

```
<inputBinding object="form" property="parameter(contractNumber)" />
```

To retrieve the parameter from the template, use:

```
$form.getParameter("contractNumber").
```

Packaging

You can package your Struts action classes as usual at the WAR level. For struts forms, if you are not using batch report, then you can package them at the WAR level, but if you do use batch report, the forms must be accessible by non-Web components such as the Common Center batch report job. In that case, you must package your report forms at the EAR level. For example, make them part of the reporting-ext.1.2.1.jar file.

You must register your report XML files in the reportList.properties file, and put the report XML files in the *EDX_HOME*/config/rpt directory, where *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billling.

However, it is possible to put the report XML files under the other sub-directories of *EDX_HOME*.

Hiding Report Columns and Manipulating IReport

After you call IReportActionHelper and get back an IReport object, you can manipulate the object before forwarding it to report.jsp.

For example, to hide some columns based on certain conditions, get the IReportConfig object from IReport, find the ITransformerColumnConfig of the corresponding columns, and set the Hidden attribute based on your conditions.

Unlimited Paging

By default, the Reporting Engine retrieves 1000 rows from the data source. You can configure the number of rows the Reporting Engine retrieves (maxRows or fetchSize) in the report XML file.

The following sample code shoes how to configure the size in the sample report.xml file:

```
<transformer id="tr1" …>

<paging fetchSize="2000"/>

</transformer>
```

To retrieve all the rows from the data source without impeding performance, you can use unlimited paging. *Unlimited paging* enables the Reporting Engine to get the result set in batches and allows users to page across multiple batches. A *fetch* is one batch.

Unlimited paging retrieves result set rows in multiple fetches on demand when the user requests them. The user pages through the result set like regular paging. If the requested page is not in the current fetch, the Reporting Engine gets the next fetch from the data source. However, all the intricacies of checking whether the requested page is in the current fetch, if not getting next the fetch, are hidden from the end user.

Fetch size is the number of result set rows in one fetch. You can configure fetch size and page size in report XML. The following sample XML demonstrates how you can enable unlimited paging and to define the fetch size.

```
<transformer id="tr1" pageSize="20"  …>

<paging unlimited="true" fetchSize="5000"/>

</transformer>
```

The Reporting Engine supports unlimited paging for SQL data source and object data source. If unlimited paging is enabled, sorting and calculator is not supported because it is necessary to sort and apply calculator operations for all the result set across all the fetches rather than current fetch.

Unlimited Paging for SQL Data Source

For the SQL data source you define the query as usual. The Reporting Engine embeds this query with in `select count(*)` to get the size of the total result set.

Unlimited Paging for Object Data Source

For the object data source to get the result set in batches, the data source provides the Reporting Engine with a call back method which retrieves the data from start position to end position. For this purpose, the Reporting Engine expects an object which implements the call back interface in report context rather than result set object. That means, for regular paging, you put result set list or array of objects in the report context and for unlimited paging, you put an object which implements call back interface. This call back interface is called IReportObjectResultSet and has the following methods:

```
public Object getResultSet (ReportObjectSearchCriteria objectSearchCriteria);
```

```
public int getResultSetSize();
```

ReportObjectSearchCriteria object has the start position and end position of a fetch.

The object you put in the result set must implement getResultSet and getResultSetSize().

The getResultSet(ReportObjectSearchCriteria objectSearchCriteria) method returns result set rows from start position to end position defined in objectSearchCriteria.

The getResultSetSize() method returns size of the complete result set that data source returns. If you do not know the result set size, you can return IReportObjectResultSet.unknownResultSetSize.

Reloading Report XML and Templates without Restarting the Server

If you change the report XML, you can use following URL to reload it, where *localhost:7001* is the name and port number for your local host:

http://*localhost:7001*/ebilling/reporting/reloadReportConfig.jsp

When you change the Velocity templates, the Velocity engine loads the templates automatically. However, because of browser caching issues, restart the server or cleanup the browser cache.

If you are putting the resource bundle files under *EDX_HOME* and load them through `<localizer>`, then the resource bundle can also be reloaded with preceding URL.

The URL will not work in a clustered environment because it only refreshes the cache in one JVM.

Customizing Threshold Values for Batch Reporting

Oracle Self-Service E-Billling sets batch threshold values for Account Billing Overview and the Service Billing Overview reports by default. To implement this feature for other reports, you must specify the report threshold value, which determines the number of result set lines above which a report must process in batch mode instead of as an online download. Each report type uses this threshold value as follows:

- **CSV.** The report threshold value is the maximum number of output lines.
- **PDF.** A percentage of the batch threshold value (the default is 10%).
- **XML.** A percentage of the batch threshold value (the default is 20%).

For example, if the CSV report threshold is set to 3,000 result set lines, then a PDF threshold value set at 10% must process in batch mode when it has 300 or more result set lines

Table 42 shows the report XML file name and the report ID, found in the XML file, that you use to set batch report thresholds.

Table 42. Report XML File Names and IDs

Report Name in UI	Report XML File Name	Report ID in XML
Account Billing Overview	telco_billing_account.xml	telco_std_r1
Account Billing Trend	telco_billing_account.xml	telco_std_r9
Statement Billing Overview	telco_billing_account.xml	telco_std_r5
Service Billing Overview	telco_billing_contract.xml	telco_std_r3
Service Billing Trend	telco_billing_contract.xml	telco_std_r11
Service Details	telco_billing_contract.xml	telco_std_r13
Total Cost by Plan	telco_billing_contract.xml	telco_std_r6
Find Calls	telco_billing_contract.xml	telco_find_call
Highest Spending Services	telco_topX.xml	telco_topX_r13
Highest Spending Services by Service Agreement	telco_topX.xml	telco_HighestSpending ServicesBySA_topX
Most Expensive Calls	telco_topX.xml	telco_topX_r7
Longest Calls	telco_topX.xml	telco_topX_r12
Most Frequently Called Numbers	telco_topX.xml	telco_topX_r10
Most Frequently Called Numbers by Service Agreement	telco_topX.xml	telco_FreqNumberBySA_topX
Most Frequently Called Numbers by Service Agreement Detail	telco_topX.xml	telco_FreqNumberBySADetail _topX

Table 42. Report XML File Names and IDs

Report Name in UI	Report XML File Name	Report ID in XML
Most Frequently Called Destinations	telco_topX.xml	telco_topX_r15
Most Frequently Called Destinations by Service Agreement	telco_topX.xml	telco_FreqDestBySA_topX
Most Frequently Called Destinations by Service Agreement Detail	telco_topX.xml	telco_FreqDestBySADetail_topX
Most Frequently Called Countries	telco_topX.xml	telco_topX_r16
Most Frequently Called Countries by Service Agreement	telco_topX.xml	telco_FreqCountryBySA_topX
Most Frequently Called Countries by Service Agreement Detail	telco_topX.xml	telco_FreqCountryBySADetail_topX

To specify a batch reporting threshold for a report

1. Add the following code before the <dataSource> tag in the report xml file, specifying the threshold value and the XML report ID:

```
<batchCriteria threshold="10" reportId="batchCriteria_accounts">

            <query></query>

            <inputBindings name="input bindings name"></inputBindings>

        </batchCriteria>
```

2. You can also set the stopThreshold attribute. When you set the stopThreshold attribute, if the testing report result meets the stopThreshold value, the report engine withholds the report and displays the following message:

   The scope of the report you have requested is too large; reduce the number of periods, change your filter criteria, or change your hierarchy position.

Customizing Charts

Some Oracle Self-Service E-Billling reports use Data Visualization Tools (DVT) for charts. You can customize the format of DVT charts, specifying size, fonts, color, and so on for each chart type. The properties for each report are stored in individual property files. The properties in each file apply to all report charts of that type.

You can also create additional property files with alternate formats to associate with particular reports. You can associate only one property file with a chart type at any time, however.

For details about which reports can be presented as charts, see *Application Guide for Oracle Self-Service E-Billing (Business Edition)* or *Application Guide for Oracle Self-Service E-Billing (Consumer Edition)*.

For details on which properties are configurable for each type of chart available, see "Configurable Chart Properties" on page 145.

To customize the DVT charts

1 To customize the formatting properties for a particular DVT chart, edit the property file associated with the particular chart type. The following chart property files can be found in `EDX_HOME/config/chart` directory:

DVT Chart Type	Property File
Vertical bar chart	vertical_bar_chart.properties
Horizontal bar chart	horiz_bar_chart.properties
Vertical stack bar chart	vertical_stack_bar_chart.properties
Horizontal stack bar chart	horiz_stack_bar_chart.properties
Pie chart	pie_chart.properties
Line chart	line_chart.properties

2 You can create an alternate property file for a chart type and associate it with the chart type (replacing the default property file for the chart). Edit the chart section of XML in the `EDX_HOME/config/rpt/`*`filename`*`.xml` file, where *filename* is the name of the report. Replace the name of the properties file with the new one in the style statement as shown in the following table. (Specify only one property file for each chart type at a time.)

DVT Chart Type	Chart Type as Indicated in the Report XML	Corresponding XML Style Statement Where You Specify the Chart Property File Name
Vertical bar chart	`type="BAR_VERT_CLUST"`	`style="config/chart/vertical_bar_chart.properties"`
Horizontal bar chart	`type="BAR_HORIZ_CLUST"`	`style="config/chart/horiz_bar_chart.properties"`
Vertical stack bar chart	`type="BAR_VERT_STACK"`	`style="config/chart/vertical_stack_bar_chart.properties"`
Horizontal stack bar chart	`type="BAR_HORIZ_STACK"`	`style="config/chart/horiz_stack_bar_chart.properties"`
Pie chart	`type="PIE"`	`style="config/chart/pie_chart.properties"`
Line chart	`type="LINE"`	`style="config/chart/line_chart.properties"`

Configurable Chart Properties

You can configure many properties of each Oracle DVT chart type available with Oracle Self-Service E-Billling.

For instructions on updating the properties for a chart, see "Customizing Charts" on page 143.

Bar Chart and Stack Bar Chart Properties

Table 43 shows the configurable properties for the following types of charts:

- Vertical bar chart
- Horizontal bar chart
- Vertical stack bar chart
- Horizontal stack bar chart

Table 43. Configurable Properties Horizontal and Vertical Stack Bar Charts

Property Description	Property Name	Values or Units
Chart width	width	Pixels
Dashboard width	dashboardWidth	Pixels
Chart height	height	Pixels
Dashboard height	dashboardHeight	Pixels
Three-dimensional effect	3D	True or False
Gradient effect	gradient	True or False
Background color	backgroundColor	Transparent or a hex color code
Legend display location	legend	Auto, Top, Bottom, Left, Right, or None
Legend background color	legendBGColor	Transparent or a hex color code
Legend border color	legendBorderColor	Transparent or a hex color code
Legend font color	legendFontColor	Hex color code
Legend font type	legendFont	Style name
Legend font size	legendFontSize	Number
Legend dashboard font size	dashboardLegendFontSize	Number
Bold on legend	legendBold	True or False
Italic on legend	legendItalic	True or False

Table 43. Configurable Properties Horizontal and Vertical Stack Bar Charts

Property Description	Property Name	Values or Units
Underline on legend	legendUnderline	True or False
Legend alignment	legendAlignment	Right, Left, or Center
Series colors	seriesColors	String of hex color codes separated by a comma
Color by group	colorByGroup	True or False
Color of plot background	plotBGColor	Transparent or a hex color code
Color of plot border	plotBorderColor	Transparent or a hex color code
Alignment of data labels	dataLabels	Above, Center, or None
Number of digits to display to the right of the decimal point	dataDecimalDigit	Whole number
Maximum bar width	maxBarWidth	Pixels
Color of title font	titleFontColor	Hex color code
Font style of title	titleFont	Style name
Font size of title	titleFontSize	Points
Font size of dashboard title	dashboardTitleFontSize	Points
Bold on title	titleBold	True or False
Italics on title	titleItalic	True or False
Underline on title	titleUnderline	True or False
Alignment of title	titleAlignment	Left, Right, or Center
Color of X axis title	xAxisTitleColor	Hex color code
Font style on X axis title	xAxisTitleFont	Style name
Font size of X axis title	xAxisTitleFontSize	Points
Font size of X axis dashboard title	dashboardXAxisTitleFontSize	Points
Bold on X axis title	xAxisTitleBold	True or False
Italics on X axis title	xAxisTitleItalic	True or False
Underline on X axis title	xAxisTitleUnderline	True or False
Alignment of X axis title	xAxisTitleAlignment	Left, Right, or Center
Color of X axis label	xAxisLabelColor	Hex color code
Font on X axis label	xAxisLabelFont	Style name
Font size of X axis label	xAxisLabelFontSize	Points

Table 43. Configurable Properties Horizontal and Vertical Stack Bar Charts

Property Description	Property Name	Values or Units
Font size of X axis dashboard label	dashboardXAxisLabelFontSize	Points
Bold on X axis label	xAxisLabelBold	True or False
Italics on X axis label	xAxisLabelItalic	True or False
Underline on X axis label	xAxisLabelUnderline	True or False
Alignment of X axis label	xAxisLabelAlignment	Left, Right, or Center
Color of X axis line	xAxisLineColor	Hex color code
Display X axis grid	xGrid=false	True or False
Color of X axis grid	xGridColor	Hex color code
Color of Y axis title	y1AxisTitleColor	Hex color code
Font of Y axis title	y1AxisTitleFont	Style name
Font size of Y axis title	y1AxisTitleFontSize	Number of points
Font size of Y axis dashboard title	dashboardY1AxisTitleFontSize	Number of points
Bold on Y axis title	y1AxisTitleBold	True or False
Italics on Y axis title	y1AxisTitleItalic	True or False
Underline on Y axis title	y1AxisTitleUnderline	True or False
Alignment of Y axis title	y1AxisTitleAlignment	Left, Right, or Center
Color of Y axis label	y1AxisLabelColor	Hex color code
Font on Y axis label	y1AxisLabelFont	Style name
Font size of Y axis label	y1AxisLabelFontSize	Points
Font size of Y axis dashboard label	dashboardY1AxisLabelFontSize	Points
Bold on Y axis label	y1AxisLabelBold	True or False
Italics on Y axis label	y1AxisLabelItalic	True or False
Underline on Y axis label	y1AxisLabelUnderline	True or False
Alignment of Y axis label	y1AxisLabelAlignment	Left, Right, or Center
Color of Y axis line	y1AxisLineColor	Hex color code
Show Y axis Grid	y1Grid	True or False
Color of Y axis grid	y1GridColor	Hex color code
Number of markers per row	nMarkersPerRow	Whole number

Pie Chart Properties

Table 44 shows the configurable properties for pie charts.

Table 44. Configurable Properties for Pie Charts

Property Description	Property Name	Values or Units
Three-dimensional effect	3D	True or False
Gradient effect	gradient	True or False
Width of chart	width	Pixels
Height of chart	height	Pixels
Width of dashboard	dashboardWidth	Pixels
Height of dashboard	dashboardHeight	Pixels
Background color	backgroundColor	Transparent or a hex color code
Legend display location	legend	Auto, Top, Bottom, Left, Right, or None
Legend background color	legendBGColor	Transparent or a hex color code
Legend border color	legendBorderColor	Transparent or a hex color code
Legend font color	legendFontColor	Hex color code
Legend font type	legendFont	Style name
Legend font size	legendFontSize	Number
Legend dashboard font size	dashboardLegendFontSize	Number
Bold on legend	legendBold	True or False
Italic on legend	legendItalic	True or False
Underline on legend	legendUnderline	True or False
Legend alignment	legendAlignment	Left, Right, or Center
Number of markers per row	nMarkersPerRow	Whole number
Series colors	seriesColors	String of hex color codes separated by a comma
Color by group	colorByGroup	True or False
Color of plot background	plotBGColor	Transparent or a hex color code
Color of plot border	plotBorderColor	Transparent or a hex color code

Table 44. Configurable Properties for Pie Charts

Property Description	Property Name	Values or Units
Alignment of data labels	dataLabels	Above, Center, or None
Number of digits to display to the right of the decimal point	dataDecimalDigit	Whole number
Color of title font	titleFontColor	Hex color code
Font style of title	titleFont	Style name
Font size of title	titleFontSize	Points
Font size of dashboard title	dashboardTitleFontSize	Points
Bold on title	titleBold	True or False
Italics on title	titleItalic	True or False
Underline on title	titleUnderline	True or False
Alignment of title	titleAlignment	Left, Right, or Center
Type of pie slice label	sliceLabelType	■ **Percent.** The percentage value. ■ **Series.** The name of the series. ■ **Value.** The value of each slice. ■ **Series_percent.** The series name and percentage of the slice.
Position of pie slice label	sliceLabelPosition	Inside, Outside_without_feeler, Outside_with_feeler, or None.
Color of pie slice label	sliceLabelColor	Hex color code
Font style of pie slice label	sliceLabelFont	Style name
Font size of pie slice	sliceLabelFontSize	Number
Bold on pie slice label	sliceLabelBold	True or False
Italics on pie slice label	sliceLabelItalic	True or False
Underline on pie slice label	sliceLabelUnderline	True or False

Line Chart Properties

Table 45 shows the configurable properties for line charts.

Table 45. Configurable Properties for Line Charts

Property Description	Property Name	Values or Units
Width of chart	width	Pixels
Width of dashboard	dashboardWidth	Pixels
Height of chart	height	Pixels
Height of dashboard	dashboardHeight	Pixels
Three-dimensional effect	3D	True or False
Gradient effect	gradient	True or False
Background color	backgroundColor	Transparent or a hex color code
Legend display location	legend	Auto, Top, Bottom, Left, Right, or None
Legend background color	legendBGColor	Transparent or a hex color code
Legend border color	legendBorderColor	Transparent or a hex color code
Legend font color	legendFontColor	Hex color code
Legend font type	legendFont	Style name
Legend font size	legendFontSize	Number
Legend dashboard font size	dashboardLegendFontSize	Number
Bold on legend	legendBold	True or False
Italic on legend	legendItalic	True or False
Underline on legend	legendUnderline	True or False
Legend alignment	legendAlignment	Left, Right, or Center
Series colors	seriesColors	String of hex color codes separated by a comma
Color by group	colorByGroup	True or False
Color of plot background	plotBGColor	Transparent or a hex color code
Color of plot border	plotBorderColor	Transparent or a hex color code
Alignment of data labels	dataLabels	Above, Center, or None

Table 45. Configurable Properties for Line Charts

Property Description	Property Name	Values or Units
Number of digits to display to the right of the decimal point	dataDecimalDigit	Whole number
Color of title font	titleFontColor	Hex color code
Font style of title	titleFont	Style name
Font size of title	titleFontSize	Points
Font size of dashboard title	dashboardTitleFontSize	Points
Bold on title	titleBold	True or False
Italics on title	titleItalic	True or False
Underline on title	titleUnderline	True or False
Alignment of title	titleAlignment	Left, Right, or Center
Color of X axis title	xAxisTitleColor	Hex color code
Font style on X axis title	xAxisTitleFont	Style name
Font size of X axis title	xAxisTitleFontSize	Points
Font size of X axis dashboard title	dashboardXAxisTitleFontSize	Points
Bold on X axis title	xAxisTitleBold	True or False
Italics on X axis title	xAxisTitleItalic	True or False
Underline on X axis title	xAxisTitleUnderline	True or False
Alignment of X axis title	xAxisTitleAlignment	Left, Right, or Center
Color of X axis label	xAxisLabelColor	Hex color code
Font on X axis label	xAxisLabelFont	Style name
Font size of X axis label	xAxisLabelFontSize	Points
Font size of X axis dashboard label	dashboardXAxisLabelFontSize	Points
Bold on X axis label	xAxisLabelBold	True or False
Italics on X axis label	xAxisLabelItalic	True or False
Underline on X axis label	xAxisLabelUnderline	True or False
Alignment of X axis label	xAxisLabelAlignment	Left, Right, or Center
Color of X axis line	xAxisLineColor	Hex color code
Display X axis grid	xGrid=false	True or False
Color of X axis grid	xGridColor	Hex color code
Color of Y axis title	y1AxisTitleColor	Hex color code

Table 45. Configurable Properties for Line Charts

Property Description	Property Name	Values or Units
Font of Y axis title	y1AxisTitleFont	Style name
Font size of Y axis title	y1AxisTitleFontSize	Number of points
Font size of Y axis dashboard title	dashboardY1AxisTitleFontSize	Number of points
Bold on Y axis title	y1AxisTitleBold	True or False
Italics on Y axis title	y1AxisTitleItalic	True or False
Underline on Y axis title	y1AxisTitleUnderline	True or False
Alignment of Y axis title	y1AxisTitleAlignment	Left, Right, or Center
Color of Y axis label	y1AxisLabelColor	Hex color code
Font on Y axis label	y1AxisLabelFont	Style name
Font size of Y axis label	y1AxisLabelFontSize	Points
Font size of Y axis dashboard label	dashboardY1AxisLabelFontSize	Points
Bold on Y axis label	y1AxisLabelBold	True or False
Italics on Y axis label	y1AxisLabelItalic	True or False
Underline on Y axis label	y1AxisLabelUnderline	True or False
Alignment of Y axis label	y1AxisLabelAlignment	Left, Right, or Center
Color of Y axis line	y1AxisLineColor	Hex color code
Show Y axis Grid	y1Grid	True or False
Color of Y axis grid	y1GridColor	Hex color code

Customizing the Statement Summary Chart

You can customize the vertical bar chart shown on the Statement Summary page in the Billing and Payment application. The properties for this chart are maintained in the statement.properties file. For details about the configurable properties for the Statement Summary chart, see "Bar Chart and Stack Bar Chart Properties" on page 145.

NOTE: The dashboard height and width properties are not used with the Statement Summary chart.

To customize the Statement Summary vertical bar chart

- Edit the statement.properties file, found in the *EDX_HOME*/config/chart directory (use back slashes (\) on Windows).

Reporting on User Audit Data

Oracle Self-Service E-Billling audits some enrollment user actions performed in the Billing and Payment application.

Oracle Self-Service E-Billling audits the following actions that occur when creating users:

- A B2B administrator creates another B2B user’s account.
- A B2C user creates his or her own account.
- The default CSR administrator creates another CSR administrator account.
- A CSR administrator creates a CSR user’s account.
- A CSR administrator creates another B2B administrator’s account.
- A CSR user (administrator or CSR) impersonates a B2B administrator creating another B2B user’s account.

Oracle Self-Service E-Billling audits the following actions that occur when enrolling users:

- An end user (B2B or B2C) enrolls.
- A CSR user (administrator or CSR) enrolls.

Oracle Self-Service E-Billling audits the following actions that occur when updating user profiles:

- An end user (B2B or B2C) updates his or her own user access information.
- An end user (B2B or B2C) updates his or her own notification settings.
- A B2B user (administrator or manager) updates another B2B user’s user access information.
- A CSR user (administrator or CSR) updates his or her own user access information.
- A CSR administrator updates another CSR user’s user access information.
- A CSR user (administrator or CSR) updates another B2B user’s user access information.
- A CSR user (administrator or CSR) updates the following by impersonation:
  - His or her own notification settings (B2B or B2C).
  - His or her own user access information (B2B or B2C).
  - A B2B user (administrator or manager) updates another B2B user’s user access information.
- A B2B, B2C, or CSR user enrolls to complete the reactivation process after clicking the URL in an email notification.
- A B2B, B2C, or CSR user resets his or her forgotten password.
- A migrated B2B, B2C, or CSR user creates a new HIPPA-compliant password after clicking the URL in the email notification.
- A B2B, B2C, or CSR user updates his or her own expired password.

Oracle Self-Service E-Billling audits the following actions that occur when deleting users:

- A B2B administrator user deletes another B2B user’s account.
- A CSR user (administrator or CSR) deletes an end user’s account.

- A CSR administrator deletes another CSR user's account.
- A CSR user (administrator or CSR) impersonates a B2B administrator user deleting another B2B user's account.

Oracle Self-Service E-Billling audits the following actions that occur when logging in and out:

- An end user (B2B or B2C) logs in.
- A CSR user (administrator or CSR) logs in.
- A CSR user (administrator or CSR) impersonates a B2B or B2C user logging in or out.
- A B2B, B2C, or CSR user fails to log in.
- A CSR administrator reactivates a locked-out account.

You can report on the audit data for each user role, including the user who performed the action, the date and time, IP address, and various attributes. For details about payment audit data, see "About Payment Auditing" on page 246. For information about database auditing, see "Auditing Database Administration Activity" on page 23.

You can create customized reports on the audited user enrollment data. Oracle Self-Service E-Billling stores audit data for user enrollment activities in the EDX_UMF_USER_AUDIT table. Table 46 describes the EDX_UMF_USER_AUDIT table:

Table 46. EDX_UMF_USER_AUDIT Table

Column Name	Description
ID	A unique ID assigned to the each occurrence of a user enrollment event, generated automatically by sequence.
USER_ID	The ID of the user who performed the action, either on his or her own account or on another user's account.
ACTION	The ID indicating the type of user action. (Action type IDs are defined in the EDX_UMF_USER_ACTION_TYPE table.)
ACTION_DATE	The date of the user action.
TARGET_USER_ID	The ID of the target user. If the user performed the action on his or her own account (the target user is the same as the USER_ID), the value in this column is null.
USER_ROLE	The role of the user who performed the action.
NOTES	The reason for locking an account: Incorrect Login, Reset Password, Security Question, or Account Expired. (User account reactivation only.)
ATTRIBUTES	The changed attribute.
IP_ADDRESS	The IP address of the user who accessed the Billing and Payment application.

Table 47 describesdescribesdescribes user action type table, EDX_UMF_USER_ACTION_TYPE:

Table 47. Definition of the EDX_UMF_USER_ACTION_TYPE Table

Column Name	Description
ID	ID associated with the user action type.
TYPE	User action type.
RESOURCE_KEY	Key for the language resource bundle.

Table 48 shows the ID associated with each type of user action. These associations are stored in the EDX_UMF_USER_ACTION_TYPE Table:

Table 48. ID of User Action Types Stored in the iEDX_UMF_USER_ACTION_TYPE Table

ID	User Action Type
1	Reactivate
2	Impersonate-login
3	Impersonate-logout
4	Login
5	Logout
6	Update user profile
7	Update notifications
9	Reset password
10	Create user
13	Enroll user
16	Delete user
17	Login failure

Table 49 shows the user action type recorded in the EDX_UMF_USER_AUDIT table for each user enrollment activity in Oracle Self-Service E-Billling.

Table 49. User Action Types Used for Each User Enrollment Activity

User Action Type ID and Description	Associated User Enrollment Activities	Valid Attributes
1 - Reactivate	A CSR administrator reactivates a locked-out account.	Password
2-Impersonate-login	A CSR user (administrator or CSR) impersonates a B2B or B2C user logging in.	Null
3-Impersonate-logout	A CSR user (administrator or CSR) impersonates a B2B or B2C user logging out.	Null
4-Login	A B2B, B2C, or CSR user logs in.	Null
5-Logout	A B2B, B2C, or CSR user logs in.	Null
6 - Update user profile	■ A B2B, B2C, or CSR user updates his or her own user access information.	One of the following: ■ One or more of the following: First name, last name, and email address ■ Password ■ Security question and answer
	■ A B2B user (administrator or manager) updates another B2B user's access information. ■ A CSR administrator updates another CSR user's access information. ■ A CSR user (administrator or CSR) updates a B2B or B2C user's access information by impersonation.	One of the following: ■ One or more of the following: First name, last name, and email address ■ Role
	■ A migrated B2B, B2C, or CSR user creates a new HIPPA-compliant password.	Null
	■ A B2B, B2C, or CSR user updates his or her own expired password.	Password
7 - Update notifications	■ An end user (B2B or B2C) updates his or her own notification settings. ■ A CSR user (administrator or CSR) updates a B2B or B2C user's notifications by impersonation.	Notifications

Table 49. User Action Types Used for Each User Enrollment Activity

User Action Type ID and Description	Associated User Enrollment Activities	Valid Attributes
9 - Reset password	A B2B, B2C, or CSR user resets his or her forgotten password.	Password
10 - Create user	■ A B2B administrator creates another B2B user's account. ■ A B2C user creates his or her own account. ■ The default CSR administrator creates another CSR administrator account. ■ A CSR administrator creates another CSR user's account. ■ A CSR administrator creates a B2B administrator's account. ■ A CSR administrator or user creates a B2B user's account while impersonating a B2B administrator.	Null
13 - Enroll user	■ A B2B or B2C end user enrolls. ■ A CSR user (administrator or CSR) enrolls. ■ A migrated B2B, B2C, or CSR user creates a new HIPPA-compliant password.	Null
	■ A B2B, B2C, or CSR user enrolls to complete the reactivation process after clicking the URL in an email notification.	Security Question and Answer
16 - Delete user	■ A B2B administrator user deletes another B2B user's account. ■ A CSR user (administrator or CSR) deletes a B2B or B2C end user's account by impersonation. ■ A CSR administrator deletes another CSR user's account.	Null
17 - Login failure	A B2B, B2C, or CSR user login fails.	Null

Reporting on System Administrator Audit Data

Oracle Self-Service E-Billling audits the following system administrator actions performed in the Command Center application:

- Creating new jobs
- Updating jobs
- Removing jobs
- Scheduling jobs
- Running jobs
- Creating a new administrator
- Enrolling a new administrator
- Updating an administrator's information
- Adding, updating, and deleting payment settings
- Logging into and out of the Command Center

You can create customized reports on the audited administrator data. Oracle Self-Service E-Billling stores the audit data for these system administrator activities in the administrator activity table, ADMIN_ACTIVITY.

Table 50 describes the ADMIN_ACTIVITY table:

Table 50. Definition of the ADMIN_ACTIVITY Table

Column Name	Description
ACTIVITY_ID	Unique ID assigned to the each occurrence of a Command Center activity, generated automatically by sequence.
ACTIVITY_CODE	The name of the activity performed by the system administrator. See Table 51 on page 159 for a list of valid activity codes.
PRODUCT_CODE	The product name for the Command Center application. The default value is ESTATEMENT.
DDN_REFERENCE	The DDN reference number.
LOGIN_ID	The ID of the system administrator who logged into the Command Center to perform the activity.
ACTIVITY_START_TIME	The time the activity started.
ACTIVITY_END_TIME	The time the activity finished.
FLEX_FIELD1	For logging into and out of the Command Center: Whether the activity was successful (Yes or No). For creating, updating, removing, running, or scheduling a job: The name of the job.
FLEX_FIELD2	The IP address where the administrator logged in and performed the Command Center activity.
DATE_CREATED	Date when the audit record was created.
CREATED_BY	The ID of the system administrator who performed the activity.

Table 50. Definition of the ADMIN_ACTIVITY Table

Column Name	Description
UPDATE_DATE	This field is not currently used.
UPDATED_BY	This field is not currently used.

Table 51 shows the valid activity codes that can be stored in the ACTIVITY_CODE column in the ADMIN_ACTIVITY:

Table 51. Valid Activity Codes

Valid Activity Code	Description
LOGIN	An administrator logs in.
LOGOUT	An administrator logs out.
CREATE JOB	An administrator creates a new job.
UPDATE JOB	An administrator updates job information.
DELETE JOB	An administrator deletes a job.
CREATE USER	The default administrator creates a new administrator user.
ENROLL USER	A newly created administrator user enrolls his or her own information.
UPDATE USER	An administrator updates his or her own information.
CREATE PAYMENT SETTINGS	An administrator creates payment settings.
UPDATE PAYMENT SETTINGS	An administrator updates payment settings.
DELETE PAYMENT SETTINGS	An administrator deletes payment settings.

7 About Payment Processing

This chapter describes how Oracle Self-Service E-Billling processes certain payment activities. It includes the following topics:

- About Check Processing on page 161
- About Credit Card Processing on page 168
- About Recurring Payments on page 173

About Check Processing

This topic describes how Oracle Self-Service E-Billling supports check payments through the ACH gateway.

Adding a Check Account

This topic is one example of adding a check account. You might use this feature differently, depending on your business model.

The following actions describe the process to enroll a new user with Oracle Self-Service E-Billling Payment who specifies a check account at enrollment time.

To add a check account for a new user

1. A new customer enrolls for check payment services by completing an enrollment form in the user interface. Oracle Self-Service E-Billling saves the information in the payment_accounts table with an enrollment status of pnd_active.
2. The pmtSubmitEnroll job runs to submit the enrollment information to the payment gateway. It changes the enrollment status to pnd_wait. If the check cannot be submitted, its status is changed to Failed.

   For ACH only, pmtSubmitEnroll sends customer enrollment information, which is contained in a zero amount check called a prenote, to an ACH payment gateway for verification. To send a prenote, the pmtSubmitEnroll job creates a zero amount check with status of prenote_scheduled, and immediately inserts the check into the check_payments table with a status of prenote_processed. This means that the status prenote_scheduled is transitory, and so is not visible in the check_payments table. A summary report is created, which can be viewed from the Command Center.
3. After receiving the customer enrollment information, the ACH payment gateway responds with a return file only if there are errors in the customer enrollment information. If there are no errors, ACH does not send a return file, or any other form of acknowledgement.

4 The pmtConfirmEnrollment job runs. This job updates the status of the customer enrollment status to active if there are no problems after a specified number of days (by default, three days).

If the payment enrollment information is not correct, the pmtConfirmEnrollment job updates the customer enrollment status to bad_active. An exception report is created, which can be viewed from the Command Center.

5 The customer might optionally receive an email about enrollment status from the pmtNotifyEnroll job.

Check Account Enrollment Status Workflow

Figure 6 shows the status changes that a new check account goes through for enrollment, depending on customer actions and the pmtSubmitEnroll and pmtConfirmEnroll jobs. The status is kept in the account_status field in the payment_accounts table.

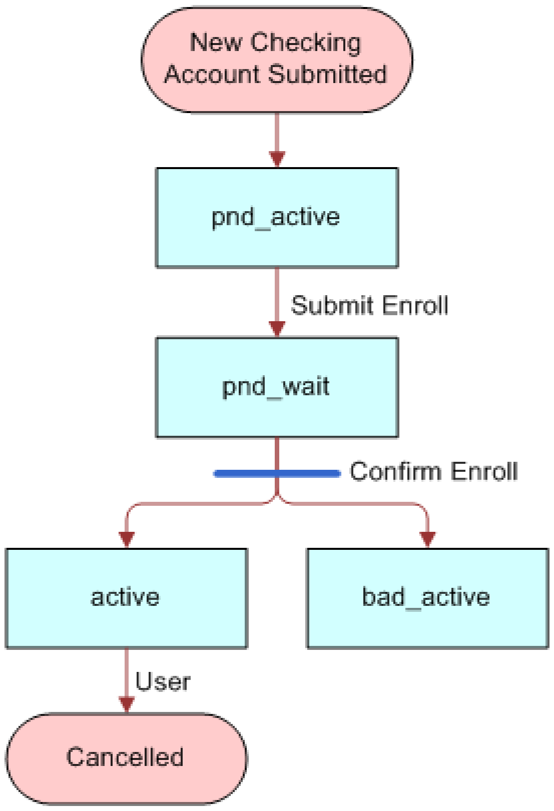


Figure 6. Check Account Enrollment Status Workflow

Table 52 describes each new check account status.

Table 52. New Check Account Status

Enrollment Status	**Description**
pnd_active	A new check account is enrolled, pending approval.
pnd_wait	The check account has been sent to the bank for verification.
active	The check account has been activated for payment.
bad_active	The check account failed to be activated.

Check Payment Transaction Workflow

Check Payment Transaction Workflow, shown in Figure 6, processes the typical ACH check payment transaction cycle (excluding transfers between the ODFI, ACH operator and RDFI).

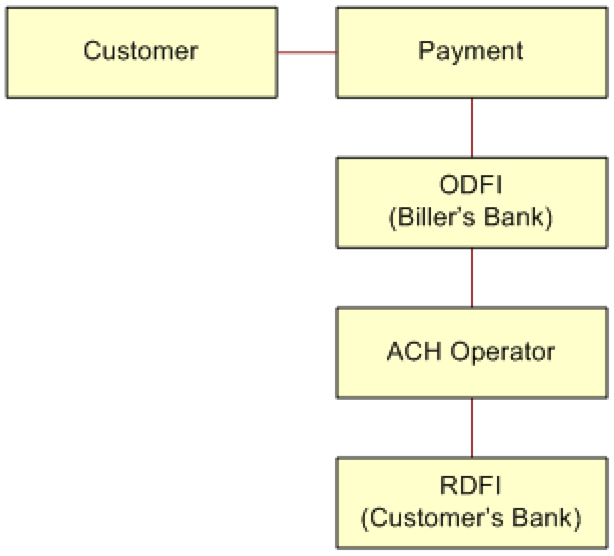


Figure 7. Check Payment Transaction Workflow

Workflow Description. This workflow performs the following actions:

1 **A customer logs in and schedules a new payment**. This step inserts a check into the database with a status of scheduled.

If the customer later cancels the payment, the check status is changed to cancelled, but the payment remains in the database for the customer to view as a cancelled payment.

2 **The pmtCheckSubmit job runs**. This step selects all the checks that are due for payment, creates a batch file of selected checks, and sends the batch file to the payment gateway (ODFI). It also changes the status of each selected check to Processed in the Oracle Self-Service E-Billling Payment database.

If the check cannot be submitted, the status is changed to Failed. A summary report log file is generated, which can be viewed from Command Center.

3 **The payment gateway (ODFI) processes the received check payment through the ACH operator to the RDFI**. In this step, if there is an error clearing the check, ACH creates a file containing a code that indicates why the check was returned, and sends the file to Oracle Self-Service E-Billling.

4 **The pmtCheckUpdate job runs**. This step changes the status of the check from processed to paid if there is no return code, and five business days (default) have passed.

If the payment gateway returns the check, the pmtCheckUpdate job updates the check's status to returned, and saves the reason code in the txn_err_msg field of the check_payments table. An exception report is generated to summarize the information in the returned file, which can be viewed from Command Center.

If there is an error other than returned, pmtCheckUpdate changes the check status to failed.

5 **If configured, the pmtPaymentReminder job sends email to the customer about the status of the check payment.**

Check Payment Status Workflow

Check Payment Status Workflow updates the check payment status at different stages of check payment processing.

Figure 7 shows the states that a check can be in, and the jobs that change the state.

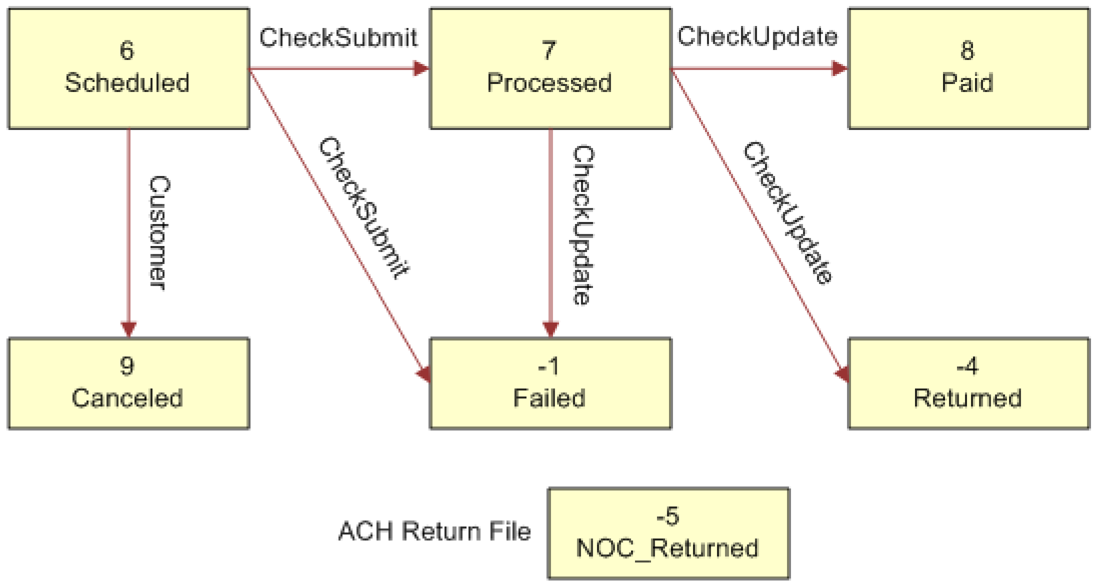


Figure 8. Check Payment Status Workflow

Table 53 lists the statuses that can occur during a check payment transaction cycle. The values in parentheses () are the values saved in the Oracle Self-Service E-Billling Payment database.

Table 53. Check Payment Transaction Status

Transaction Status	Description
Scheduled(6)	A customer scheduled a new check payment.
Processed(7)	Oracle Self-Service E-Billling Payment processed a check and sent it to the ACH gateway.
Paid(8)	ACH paid or cleared a check.
Cancelled(9)	The customer cancelled a check.
Failed(-1)	ACH failed to pay a check failed for a reason other than returned.
Returned(-4)	ACH returned a check.
noc_returned(-5)	This customer’s payment account information must be changed.

Credit Reversals

Oracle Self-Service E-Billling supports check credit reversals.

Automated Clearing House (ACH)

This topic describes the codes and other data used with Automated Clearing House (ACH) Network fund transfers. Additional information about ACH and change codes are available at:

http://www.nacha.org

Supported SEC Codes

For ACH, Oracle Self-Service E-Billling supports the following SEC Codes (Standard Entry Class Codes):

- **Web.** Internet initiated entry (default for Oracle Self-Service E-Billling).

  Debit entries are originated (either single or recurring) from a customer's account using web based authorization.

- **PPD.** Pre Arranged Payment and Deposit Entry. Under PPD the following types are included:
  - Direct Deposit: The credit application transfers funds into the customer's account.
  - Preauthorized Bill Payment: This is a debit application, where billers transfer electronic bill payment entries through the ACH network.

- **CTX.** Corporate Trade Exchange

  Supports multiple Addenda record based on ANSI ASC X12 standards. Can be used either with the credit or debit application.

ACH Change Codes (NOC)

Table 54 lists some of the ACH change codes (also known as NOC codes) that might appear in the returns file after running the pmtCheckUpdate job if previously valid payment information is now incorrect or out-of-date.

Table 54. ACH Change Code

Code	ACH Change Code Description
C01	Incorrect DFI Account Number.
C02	Incorrect Routing Number.
C03	Incorrect Routing Number and Incorrect DFI Account Number.
C05	Incorrect Transaction Code.
C06	Incorrect DFI Account Number and Incorrect Transaction Code.
C07	Incorrect Routing Number, Incorrect DFI Account Number, and Incorrect Transaction Code.

ACH Return Codes

Table 55 lists some of the ACH return codes that might appear in the returns file after running the pmtCheckUpdate job.

Table 55. ACH Return Codes

Code	ACH Return Code Description
R01	Insufficient Funds.
R02	Account Closed.
R03	No Account or Unable to Locate the Account.
R04	Invalid Account Number.
R05	Reserved.
R06	Returned at the request of ODFI.
R07	Authorization Revoked by Customer (adjustment entries).
R08	Payment Stopped or Stop Payment on Item.
R09	Uncollected Funds.
R10	Customer Advises Not Authorized; Item Is Ineligible, Notice Not Provided, Signatures Not Genuine, or Item Altered (adjustment entries).
R11	Check Truncation Entry Return (Specify) or State Law Affecting Acceptance of PPD Debit Entry Constituting Notice of Presentment or PPD Accounts Receivable Truncated Check Debit Entry.
R12	Branch Sold to Another DFI.
R14	Representative Payee Deceased or Unable to Continue in that Capacity.
R15	Beneficiary or Account Holder (Other Than a Representative Payee) Deceased.
R16	Account Frozen.
R17	File Record Edit Criteria (Specify).
R20	Non-Transaction Account.
R21	Invalid Company Identification.
R22	Invalid Individual ID Number.
R23	Credit Entry Refused by Receiver.
R24	Duplicate Entry.
R29	Corporate Customer Advises Not Authorized.
R31	Permissible Return Entry (CCD and CTX only).
R33	Return of XCK Entry.

Additional information about these and additional ACH return codes are available from the following URL:

http://www.nacha.org/

NOC Transactions

When a prenote is returned with a NOC, TXN_MESSAGE is populated with NOC information formatted as *NOC_CODE*::*NEW_ADDENDA_INFO*::*OLD_ADDENDA_INFO*, where

- *NOC_CODE* is the three-character code returned.
- *NEW_ADDENDA_INFO* is the NOC information returned from ACH, which can include the corrected account number, routing and account type.
- *OLD_ADDENDA_INFO* is the existing addenda information.

ACH Effective Date

The Skip non-business days for batch effective entry date field on the Payment Settings page for an ACH check payment gateway controls how the effective entry date is calculated when the ACH batch file is created by pmtCheckSubmit.

If the field is set to Yes, then non-business days are not taken into consideration. The effective entry date is set to the payment date that the customer specified when scheduling the payment.

If the field is set to No, then non-business days are skipped, and the effective entry date is the next business day following the computed date. Payment checks the scheduled payment date to see if it is on or before the end of today. If it is, the computed date is the customer-scheduled date plus one. If it is not, then the computed date is the customer-scheduled date.

Non-business days are weekend days, plus the U.S. Federal holidays.

ACH Settlement Date

The ACH settlement date is not written to the ACH batch file by pmtCheckSubmit. That date is added by the ACH Operator when the payment is settled.

ACH Addenda Records

Payment supports ACH addenda records, which means you can append a list of addenda records after an entry detail record in an ACH file. Addenda records are biller-specific, so customization is required to support this feature. Theoretically, you can put any information into an addenda record, such as the invoices of a payment. To add addenda records, you must write a plug-in for the pmtCheckSubmit job. For more information about supporting ACH addenda records, contact My Oracle Support.

About Credit Card Processing

Credit card payments are supported for immediate or future (scheduled) payments. Credit card payments require two steps: authorization and settlement. Authorization verifies the customer account and puts a hold on the account for the amount of the payment. Settlement occurs when the payment is made. Oracle Self-Service E-Billling Payment performs authorization and settlement in one transaction using the credit card gateway for credit card payments.

Credit card payments require an agreement with a credit card gateway to process credit card transactions. A cartridge for PayPal Payflow Pro is provided with Oracle Self-Service E-Billling, which requires signing up with PayPal Payflow Pro payment services. For help with cartridges, contact your Oracle sales representative to request assistance from Oracle's Professional Services. In addition, other cartridges can be created to support other payment processors.

Credit Card Payment Status

Table 56 lists the statuses that can occur during a credit card payment transaction cycle. The values in parentheses () are the values saved in the payment database.

Table 56. Credit Card Payment Status

Transaction Status	Description
Scheduled (6)	A customer has scheduled a new credit card payment.
Settled (8)	The credit card payment was authorized and settled successfully.
Failed-authorized (-4)	A credit card payment failed during authorization.
Cancelled (9)	A credit card payment was cancelled by the customer.
Failed (-1)	A credit card payment failed because of network problems. This state occurs only for instant payments. For scheduled payments or recurring payments, the state stays scheduled if there is a network problem, so that it will be tried again. It is not necessary for Oracle Self-Service E-Billling to retry an instant payment; the user sees the error message and can optionally retry the payment.

Credit Card Payment Transactions

Figure 9 shows the entities involved in a credit card payment transaction.

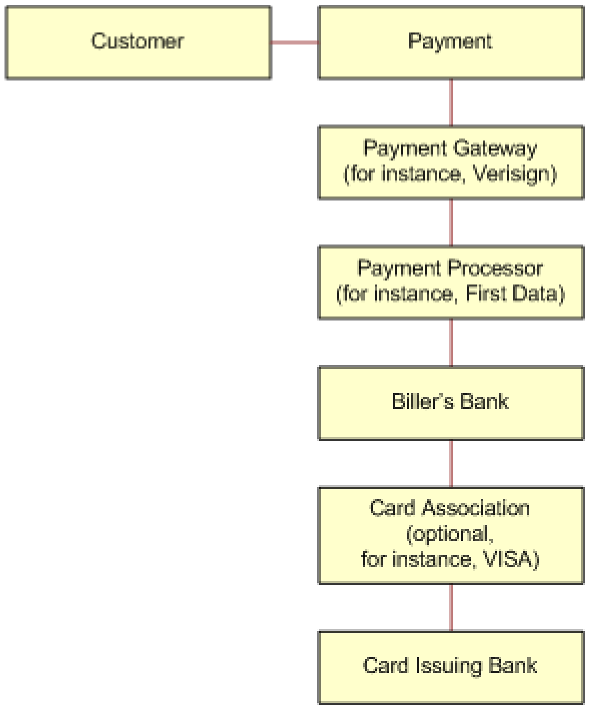

Customer
Payment
Payment Gateway (for instance, Verisign)
Payment Processor (for instance, First Data)
Biller's Bank
Card Association (optional, for instance, VISA)
Card Issuing Bank

Figure 9. Credit Card Payment Transactions

Because credit card is processing is real-time and not batch-based, the life cycle of credit card is simpler than check processing. Credit card processing typically goes through the following steps:

1 A user enters a credit card number and other card-related information.

2 The card information is sent to the card-issuing bank for authorization. Authorization only guarantees that the money is available at the time of authorization.

3 The merchant issues a settlement request to issuing bank so that the money can be transferred, usually after fulfillment (sending out ordered goods). For bill payments, the biller does not send out ordered goods, so authorization and synchronization are combined into one operation; a credit card payment is settled at the same time it is authorized.

Instant Credit Card Payments

The following code shows the states for an instant credit card payment. For instant payments, there is no scheduled state:

```
Credit Card Payment ‡ settled
                      |-> failed-authorize
                      |-> Failed
```

Instant credit card payments process is as follows:

1. A user submits an instant credit card payment from the UI.
2. Oracle Self-Service E-Billling sends the payment to credit card cartridge in real time.
3. If the card is authorize and settled, the credit card state is set to settled.
4. If the card failed to authorize, the state is set to failed_authorize.
5. If there is a network problem, the state is set to failed.
6. The card is inserted into creditcard_payments table.
7. The result of the transaction is presented to the user.
8. The pmtPaymentReminder job runs and (optionally) sends email to users who have made an instant payment.

Scheduled Credit Card Payments

The following code shows the states for a scheduled credit card payment:

```
Credit Card Payment ‡ Scheduled ‡ Cancelled
          |
          | pmtCreditCardSubmit job
          |‡ settled
          |‡ failed-authorize
          |‡ scheduled
```

Scheduled credit card payments process is as follows:

1. A credit card payment is scheduled by the customer through the user interface, and the payment is marked as Scheduled in the creditcard_payments table.

   Before the scheduled credit card payment is processed by pmtCreditCardSubmit, the user can modify or cancel it.

2. When the pmtCreditCardSubmit job runs, it selects all credit card payments that are scheduled to be paid at the time the job runs, opens a connection to the credit card payment gateway, and starts making payments. The Number of days before a credit card's pay date for it to be submitted field on the pmtCreditCardSubmit job determines how many days ahead to look when selecting payments to be made.

   If IPayPalCreditCardSubmitPlugIn has been implemented in Payment Settings, this job modifies the credit card payments that are scheduled to be paid, or takes other actions related to the selected credit card payments. Functions in the plug-in are called before and after credit card payment processing. For more information about the pmtCreditCardSubmit job and its plug-in, see *Administration Guide for Oracle Self-Service E-Billing*. For help with configuring job plug-ins, contact your Oracle sales representative to request assistance from Oracle's Professional Services.

3. The credit card gateway sends the transactions to the credit card processor. The credit card processor either authorizes and settles the credit card payment, or rejects it. The results are returned to the credit card gateway, which forwards the results to the pmtCreditCardSubmit job.
4. The pmtCreditCardSubmit job changes the status of the credit card payment in the database depending on the transaction status returned by the credit card processor, and optionally sends email to the customer about the status of the payment.

   If the card is authorized and settled, the credit card state is set to settled.

   If the card fails to authorize, the state is set to failed_authorize.

   If there is a network problem, the state remains scheduled so it will process the next time pmtCreditCardSubmit runs.
5. The pmtPaymentReminder job runs and (optionally) sends email to users about the status of their scheduled payment.

Credit Reversals

Oracle Self-Service E-Billling supports credit reversals.

User Options

The user interface to Oracle Self-Service E-Billling Payment can offer a variety of credit card payment options. Some of those options require that fields be configured in Payment Settings for a credit card payment gateway.

Using PayPal Payflow Pro as a Payment Gateway

A cartridge for PayPal Payflow Pro is provided with Oracle Self-Service E-Billling. Before configuring a PayPal Payflow Pro credit card payment gateway, you must obtain a digital certificate through PayPal Payflow Pro.

You must also configure your application server to support a PayPal Payflow Pro payment gateway. For more information about setting up a payment gateway, see *Installation Guide for Oracle Self-Service E-Billing*.

Address Verification Service

Address Verification Service (AVS) reduces the risk of fraudulent transactions by verifying that the credit card holder's billing address matches the one on file at the card issuer. The address is optional and does not affect whether the payment is accepted or rejected. However, using an address might get a lower rate from the card issuer.

A merchant (also known as the biller) submits the AVS request through the payment process directly to the specific credit card association (for example, PayPal Payflow Pro) for address comparison. If AVS is turned on by the System Administrator, address information is passed into PayPal Payflow Pro as part of the PayPal Payflow Pro request. PayPal Payflow Pro then contacts the credit card issuing bank and passes along the address information.

The credit card issuing bank verifies the credit card address information on record matches the address information from PayPal Payflow Pro. The credit card issuing bank then replies back to PayPal Payflow Pro whether information matched (address and zip code are checked during AVS). Y means yes, N means no, and X means a match cannot be determined. PayPal Payflow Pro then accepts or rejects (voids) the transaction based on the filter set through Oracle Self-Service E-Billling Payment (for both street address and zip code).

There is also a filter option to set the international AVS code to determine if the AVS response was international, U.S. or could not be determined. Some credit card issuing banks require city and state verification as well. Oracle Self-Service E-Billling Payment does not handle these by default, but the pmtCreditCardSubmit job has a plug-in to allow custom code to pass in the AVS values.

If Oracle Self-Service E-Billling Payment does not send the address information to PayPal Payflow Pro, or the system administrator did not turn on AVS, and the AVS check level is set to Full, the transaction fails. If the card issuer address is sent to the payment gateway but the address does not match the information on the gateway, then the gateway can send an AVS code. If an AVS code is received, Oracle Self-Service E-Billling Payment logs the AVS code in the audit tables.

Turning AVS On or Off by Transaction

PayPal Payflow Pro supports turning AVS on or off by transaction. However, the lower capability Payflow Link can perform this function. You also must set up the AVS level as part of your PayPal Payflow Pro agreement. When setting up the account with PayPal Payflow Pro, the merchant must specify the level of AVS check: full, medium or light. For additional information on setting up PayPal Payflow Pro, see the PayPal Payflow Pro documentation.

When Oracle Self-Service E-Billling passes the address information, PayPal Payflow Pro accepts or rejects the transaction based on the AVS check level. Note that the AVS check level is specified once during merchant account setup and applies to all transactions for that merchant. During setup, the customer (merchant) also must specify to PayPal Payflow Pro that he or she uses AVS (through Oracle Self-Service E-Billling) for transactions.

About Recurring Payments

Oracle Self-Service E-Billling provides two types of recurring payments for check and credit card:

- **A recurring payment.** A recurring payment allows a customer to schedule a payment amount that is fixed, for the entire amount due from a bill, or for the minimum amount due from a bill. The payment can be scheduled to be paid on a certain date of the week, month or quarter.
- **An automatic payment.** An automatic payment allows a customer to schedule a payment of a fixed amount, for the entire amount due from a bill, or for the minimum amount due from a bill, to be made a certain number of days before due date. Automatic payments of the entire amount due can also be made, if the amount due is less than a specified amount.

Both recurring and automatic payments are designated as recurring payments by the NACHA 2009 specification. NACHA 2009 defines a payment as recurring when the account manager (Oracle Self-Service E-Billling) keeps the account information (in a database).

Recurring payments can be modified or cancelled at any time before the payment is scheduled.

Recurring payment allows a customer to make payments automatically, based on the amount and pay date. There are five kinds of recurring payments:

- (Minimum) amount due and before due date, for example, pay the entire amount due two days before the due date.
- (Minimum) amount due and fixed pay date, for example, pay minimal amount due on day 31 of each month.
- Fixed amount and before the due date, for example, pay $100 one day before the due date.
- Fixed amount and fixed pay date, for example, pay $100 on the first day of each month.
- (Minimum) amount due up to a fixed amount, and send email if over that fixed amount.

Amount defines how much the recurring payment is going to pay for each payment. The amount can be fixed, amount due or minimum amount due. If the amount is (minimum) amount due, then it must be indexed by the Composer. The name and format of the (minimum) amount due must be specified in the Payment Settings topic of the Command Center.

Pay date defines when each payment is going to be cleared (money transfers). Pay date can be fixed or before due. If it is before due, then the due date must be indexed by the Composer. The name and format of the due date must be specified in the Payment Settings topic of the Command Center.

For monthly payments, if day 29, 30, or 31 is selected, and that day does not exist for a particular month, the pay date defaults to the last day of that month. For example, specifying day 31 of each month ensures that payments are made at the last day of each month.

For weekly payments, the week starts on Sunday. For example, day 1 of each week means Sunday.

The *effective period* defines when a recurring payment starts and ends. A payment is made if its pay date is within the effective period (inclusive). If the pay date is after the end date of the effective period, the recurring payment is deactivated. By default, a recurring payment only starts tomorrow. This is done so that all bills that arrive up to and including today are considered paid, so recurring payment must not pay these bills a second time.

There is also a script that can be run after installation that prevents a bill from being paid twice. For more information about that script, see *Installation Guide for Oracle Self-Service E-Billing*.

After an end-customer creates a recurring payment, that customer is not permitted to change the payment amount from fixed to (minimum) amount due, or to change the pay date from fixed to before due date, or conversely. When a recurring payment starts (which is when the first recurring payment has been made), the start date of the recurring payment cannot be modified.

CAUTION: Recurring payment supports only one customer account for each biller. Recurring payment does not support multiple customer accounts with a single biller.

The next topic provides examples for the first four recurring payment types. The topic after that explains how to test those payment types.

Recurring Payment Transaction Cycle

Recurring payment information is saved into the recurring_payments table.

Recurring payments can support only one customer account for a biller. Recurring payments do not support multiple customer accounts with a single biller.

The parameter, When to Synchronize Recurring Payment with Statements, belongs to the pmtRecurringPayment job.

By default, Oracle Self-Service E-Billling uses the latest available bill when submitting the payment to the payment gateway. You can configure each payment gateway to only synchronize once, which reduces processing. The setting, Whenever Job Runs, can be changed to, Only After the Current Bill is Scheduled, which causes Oracle Self-Service E-Billling to synchronize only once when the bill is scheduled.

The pmtRecurringPayment job retrieves bills, makes payments (check or credit card), and sends email notifications for recurring payments. The job performs two actions:

- Retrieves the latest bill for a recurring payment that a customer set up through the UI. This process is called *synchronization*. A recurring payment can only be synchronized with the Command Center database if it is associated with a bill and the amount to pay is the minimum (amount) due or the pay date is before the due date. A recurring payment with fixed amount and fixed date will not be synchronized with the Command Center database, which means there is no bill information associated with this recurring payment.
- Schedules payments (inserts a payment with status of scheduled in the check_payments or creditcard_payments table so that the payments will be processed. This process is called *scheduling*. A payment is scheduled three days before the pay date (by default). The number of days can be changed by changing the Number of days before pay date to schedule the payment field in the job configuration. This delay allows the customer to modify or cancel this payment before the payment is processed by the pmtCheckSubmit or pmtCreditCardSubmit jobs.

Table 57 shows the columns that are updated in the recurring_payments table by the pmtRecurringPayment job.

Table 57. Columns Updated in recurring_payments table by the pmtRecurringPayment Job

Column Name in the recurring_payments Table	Description
bill_scheduled	Y or N: determines whether the current bill associated with the recurring payment has been scheduled (inserted) into check_payments or creditcard_payments. It is always N for a fixed amount and fixed pay date.
Status	Active or Inactive: This status is calculated internally. It indicates whether the recurring payment has ended, because either the pay date is after the end date, or the number of payments has reached the maximum allowed.
last_process_time	The last synchronization time. To improve performance, only bills whose doc date falls between last_process_time and the current job running time (inclusive) are synchronized. By default, last_process_time is set to the start_date of the effective period when the recurring payment is created, which means all bills whose doc dates are before start_date will not be synchronized.
last_pay_date	The pay date of last payment made. It is set to 01/01/1970 if the recurring payment has not started yet.
next_pay_date	The pay date of next payment. It is calculated based on start_date, last_pay_date and pay_interval.
bill_id	A foreign key reference to a row in the payment_bill_summaries table. Use bill_id to retrieve the latest bill information paid by the recurring payment. It might be null if there is no such bill.
curr_num_payments	Current number of payments made.

TIP: There is no payment inserted into check_payments or creditcard_payments table when a recurring payment is created by the user. Payments are inserted by the pmtRecurringPayment job.

Tables Affected by Recurring Payments

The recurring_payments table only contains the setup information for the recurring payment, which an end-user enters using a Web interface. This table does not save bill summary or payment information. The amount field in the recurring_payments table records the amount when you do one of the following:

- Specify the recurring payment to pay fixed amount
- Pay if less than this amount
- Pay up to this amount

Oracle Self-Service E-Billling pulls bill summary information from the Command Center tables and saves it into the payment_bill_summaries table. The pmtRecurringPayment job populates the payment_bill_summaries and bill_id of the recurring_payments tables.

Payment information is scheduled into the check_payments (for check) or creditcard_payments (for credit card) tables. The recurring_payments table is updated with the payment_id.

Example of Scheduling Amount Due and Before Due Date

This topic shows an example of how a recurring payment processes for amount due, before the due date. You could use this feature differently, depending on your business model.

To schedule Amount Due and Before Due Date

1 On date 04/09/2009, a customer with account number acct1111 creates a recurring payment. The amount is amount due, the pay date is one day before due date, the start date is 04/10/2009, and the end date is 06/10/2009.

Column Name in the recurring_payments Table	Value
payer_account_number	acct1111
bill_scheduled	Y
status	active
last_process_time	04/10/2009; Same as start date.
last_pay_date	01/01/1970; Not paid yet.
next_pay_date	01/01/3000; This future date ensures there is no due date available yet.
bill_id	null
max_num_payments	2147483647. This large number means the recurring payment will only be deactivated when the pay date is after the end date.

2 The pmtRecurringPayment job runs on 04/10/2009 23:59:00PM. The job searches the recurring_payments table to find all recurring payments whose bill_scheduled is Y and status is Active. It finds the example recurring payment and then asks Command Center to return all bills whose account number is acct1111 and whose STATEMENT_LOAD_DATE is between 04/10/2009 (last_process_time) and 04/10/2009 23:59:00PM (job run time). Two bills - bill2 and bill3 - are returned. pmtRecurringPayment then finds the bill with latest due date bill3. bill2 is ignored because only the latest bill is paid.

3 After finding the latest bill from Command Center, pmtRecurringPayment checks whether the due date of this bill is after the due date of the bill used in the last payment (last bill information can be retrieved from payment_bill_summaries using the bill_id). If not, that means this is an old bill and must not be paid. In this case, because there is no last payment, bill3 is paid.

4 bill3 is inserted into the payment_bill_summaries table and the recurring_payment table is recalculated as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	N, means this bill has not been paid or scheduled
status	active, because next_pay_date is within the effective period
last_process_time	04/10/2009 23:59:00PM, changes to job run time
last_pay_date	01/01/1970, unchanged
next_pay_date	05/14/2009, one day before the due date, 05/15/2009
bill_id	bill3

5 If the pmtRecurringPayment job runs between 04/11/2009 and 05/10/2009, nothing happens to this recurring payment because synchronization and scheduling do not happen. The table remains unchanged.

6 On 05/11/2009 11:59:00PM, three days before next_pay_date, pmtRecurringPayment runs again. The recurring payment mentioned previously will not be synchronized, because its bill_scheduled is N. However, it will be scheduled. pmtRecurringPayment finds all recurring payments whose bill_scheduled is N, status is Active and next_pay_date is equal to or before 05/14/2009 (05/11/2009 + 3 days). The previously mentioned recurring payment is picked up and a payment is inserted into the check_payments or creditcard_payments table. The amount of the payment is $100.00, and the pay date is 05/14/2009. After this, the recurring payment table is changed to:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	Y, means this bill has been paid
status	Active because next_pay_date is within the effective period
last_process_time	04/10/2009 23:59:00PM, unchanged because there was no synchronization
last_pay_date	05/14/2009, change to check's pay date
next_pay_date	05/14/2009, unchanged
bill_id	bill3
payment_id	points to the new payment_id inserted into the check_payments or creditcard_payments table

The customer can now view the payment from Future Payments in the example interface. He or she can update or cancel the scheduled payment if desired.

7 On 05/12/2009 23:59:00PM, pmtRecurringPayment runs again and finds bills whose doc date is between 04/10/2009 11:59:00PM and 05/12/2009 23:59:00PM. No bills exist, and the last process time is updated to 05/12/2009 23:59:00PM. Everything else remains the same.

8 On 05/13/2009, the ETL Load job runs again and inserts a new bill, bill4. ACCOUNT_NUM is obtained from the EDX_RPT_ACCOUNT_DIM table from OLAP. Everything else is from the EDX_RPT_STATMENT_FACT table. The table details are a combination of data from EDX_RPT_ACCOUNT_DIM, EDX_RPT_ACCOUNT_FACT, and the EDX_RPT_STATEMENT_FACT tables.

Z_PRIMARY-ACCOUNT_NUM	Z_DOC_ID-STATEMENT_NUMBER	Z_DOC_DATE-STATEMENT_LOAD_DATE	AmountDue	DueDate
acct1111	bill1	03/10/2009	100.01	04/15/2009
acct1111	bill2	04/10/2009	50.00	04/25/2009
acct1111	bill3	04/10/2009	100.00	05/15/2009
acct1111	bill4	05/13/2009	80.00	06/15/2009

9 On 05/13/2009 23:59:00PM, the pmtRecurringPayment job runs again. It contacts Command Center and retrieves bills whose doc date are between 05/12/2009 23:59:00PM and 05/13/2009 23:59:00PM. bill4 is retrieved and the recurring_payments table is updated like this:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	N means this bill has not been paid
status	Inactive because next_pay_date is beyond the effective period
last_process_time	05/15/2009 23:59:00PM, changes to job run time
last_pay_date	05/14/2009, unchanged
next_pay_date	06/14/2009, one day before due date, 06/15/2009
bill_id	bill4

After synchronization, the recurring payment is deactivated, and is never synchronized or scheduled again.

Example of Scheduling Amount Due And Fixed Pay Date

This topic shows an example of how a recurring payment processes for the amount sue scheduled on a fixed pay date. You could use this feature differently, depending on your business model.

To schedule Amount Due and Fixed Pay Date

1 On 04/09/2009, a customer with account number acct1111 creates a recurring payment. The amount is amount due, the pay date is day 31 of each month, the start date is 04/10/2009,and the recurring payment stops after 10 payments.

Column Name	Value
payer_account_number	acct1111
bill_scheduled	Y
status	active
last_process_time	04/10/2009
last_pay_date	01/01/1970
next_pay_date	4/30/2009; the first available pay date after 04/10/2009 (because there is no April 31).
bill_id	null
end_date	01/01/3000; The end date is so far in the future that the recurring payment will only be deactivated when the number of payments reaches maximum allowed.
curr_num_payments	0; no payments yet.

The Bill table has the following values:

Z_PRIMARY-ACCOUNT_NUM	Z_DOC_ID-STATEMENT_NUMBER	Z_DOC_DATE-STATEMENT_LOAD_DATE	AmountDue	DueDate
acct1111	bill1	03/10/2009	100.01	04/15/2009
acct1111	bill2	04/10/2009	50.00	04/25/2009
acct1111	bill3	04/10/2009	100.00	05/15/2009

Even though the pay date is not related to the due date, DueDate must still be indexed because it is used to decide which bill is the latest.

2 The pmtRecurringPayment job runs on 04/10/2009 23:59:00PM. bill3 is found in the index table and inserted into the payment_bill_summaries table. The recurring_payments table is recalculated as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	N, this bill has not been paid.
status	Active, curr_num_payments is less than max_num_payments.

Column Name	Value
last_process_time	01/01/1970; unchanged.
last_pay_date	01/01/1970; unchanged.
next_pay_date	04/30/2009; there is no April 31.
bill_id	bill3
curr_num_payments	0

3 On 04/27/2009, three days before next_pay_date, pmtRecurringPayment runs again. There is no synchronization (bill_scheduled is N), but a payment is inserted into the check_payments or creditcard_payments table. The amount of the check is $100.00 and its pay date is 04/30/2009. The recurring payment table is changed as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	Y, means this bill has been paid.
status	Active, curr_num_payments is less than max_num_payments.
last_process_time	04/10/2009 23:59:00PM: not changed because there has been no synchronization.
last_pay_date	04/30/2009; changed to next_pay_date.
next_pay_date	05/31/2009; changed to next available pay date.
bill_id	bill3
payment_id	Points to the new payment_id inserted into the check_payments or creditcard_payments table.
curr_num_payments	1

4 Repeat steps 2, 3 and 4 until curr_num_payments reaches 10. At step 4 of the tenth payment, the status changes to Inactive.

If no bills arrive for a month, then next_pay_date is automatically moved to next month. For example, if there is no bill for April, then the next_pay_date is automatically moved from 04/30/2009 to 05/31/2009 when the current job run time is May 1.

Example of Scheduling Fixed Amount and Before Due Date

This topic shows an example of how a recurring payment processes for a fixed amount scheduled before the due date. You could use this feature differently, depending on your business model.

To schedule Fixed Amount and Before Due Date

1 On 04/09/2009, a customer with account number as acct1111 creates a recurring payment from the UI. The amount is $50, the pay date is one day before the due date, the start date is 04/10/2009 and the recurring payment stops after 10 payments.

Column Name	Value
payer_account_number	acct1111
bill_scheduled	Y
status	active
last_process_time	04/10/2009
last_pay_date	01/01/1970
next_pay_date	01/01/300
bill_id	null
end_date	01/01/3000; the end date is so far in the future that the recurring payment will only be deactivated when the number of payments reaches the maximum allowed.
curr_num_payments	0; no payment yet.

Index table entries are as follows:

Z_PRIMARY-ACCOUNT_NUM	Z_DOC_ID-STATEMENT_NUMBER	Z_DOC_DATE-STATEMENT_LOAD_DATE	DueDate
acct1111	bill1	03/10/2009	04/15/2009
acct1111	bill2	04/10/2009	04/25/2009
acct1111	bill3	04/10/2009	05/15/2009

Amount due is not required for this case.

2 The pmtRecurringPayment job runs on 04/10/2009 23:59:00PM, after running the ETL load and after the new bill has been inserted. In this case, bill3 is found in the index table and inserted into the payment_bill_summaries table. bill3 details are retrieved from the OLAP database tables and inserted into the payment_bill_summaries table. The recurring_payments table is recalculated as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	N; this bill has not been paid.
status	Active, curr_num_payments is less than max_num_payments.

Column Name	Value
last_process_time	04/10/2009 23:59:00PM; changes to job run time.
last_pay_date	01/01/1970; unchanged.
next_pay_date	05/14/2009; one day before due date, 05/15/2009.
bill_id	bill3
curr_num_payments	0

3 On 05/11/2009, three days before next_pay_date, pmtRecurringPayment runs again. There is no synchronization (because bill_scheduled is N), but a payment is inserted into the check_payments or creditcard_payments table. The amount of the payment is $50.00 and its pay date is 05/14/2009. The recurring_payments table is changed as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	Y means this bill has been paid.
status	Active, next_pay_date is not after end_date.
last_process_time	04/10/2009 23:59:00PM; unchanged, because there was no synchronization.
last_pay_date	05/11/2009; changed to next_pay_date.
next_pay_date	05/11/2009; unchanged, the next bill is not known.
bill_id	bill3
payment_id	Points to the new payment_id inserted into the check_payments or creditcard_payments table.
curr_num_payments	1

Repeat steps 2, 3 and 4 until next_pay_date is after end_date, when status changes to inactive.

Example of Scheduling Fixed Amount and Fixed Pay Date

This topic shows an example of how a recurring payment processes for a fixed amount scheduled on a fixed pay date. You could use this feature differently, depending on your business model.

To schedule Fixed Amount and Fixed Pay Date

1 On 04/09/2009, a customer with account number acct1111 creates a recurring payment. The amount is $50 and the pay date is day 1 of each month. The recurring payment starts at 04/10/2009 and ends at 06/10/2009. The columns in the recurring_payments table are updated as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	N
status	active
last_process_time	04/10/2009
last_pay_date	01/01/1970
next_pay_date	05/01/2009
bill_id	null
end_date	06/10/2009
curr_num_payments	0; no payment yet.

2 On 04/28/2009, three days before next_pay_date, pmtRecurringPayment runs again. There is no synchronization (bill_scheduled is always N) but a payment is inserted into the check_payments or creditcard_payments table. The amount of the check is $50.00 and its pay date is 05/01/2009. The columns in the recurring_payments table are updated as follows:

Column Name	Value
payer_account_number	acct1111
bill_scheduled	N; this bill has been paid.
status	Active, next_pay_date is not after end_date.
last_process_time	04/10/2009; unchanged, because there was no synchronization.
last_pay_date	05/01/2009; changed to next_pay_date.
next_pay_date	06/01/2009; changed to the next available pay date.
bill_id	null
payment_id	Points to the new payment_id inserted into the check_payments or creditcard_payments table
curr_num_payments	1

Repeat step 2 until next_pay_date is after end_date. Then the status changes to Inactive.

Payment Job Status Monitoring

When a payment job is done, an email can be sent to the administrator about the status of the email. You can enable this feature in the Payment Settings.

Payment Job Plug-In

Some Oracle Self-Service E-Billling payment jobs support plug-ins to extend core Oracle Self-Service E-Billling payment functionality.

1. If the setting When to Synchronize Recurring Payment with statements is Whenever job runs, then the pmtRecurringPayment job synchronizes the bill with the Command Center every time pmtRecurringPayment runs. If the setting is Only after the current bill is scheduled, a rebill is not synchronized unless it is time to schedule the payment.
2. The pmtRecurringPayment job runs again. If the setting Synchronize with Statements Every Time the Job Runs is Yes, and a rebill arrives, the pmtRecurringPayment job synchronizes the bill with the statements.
   - If the payment for this bill has already been scheduled, then the job cancels the scheduled payment, and schedules a payment for the updated amount.
   - If the status of the bill is Processed, then the rebill is ignored.

Each time the pmtRecurringPayment job runs, it checks the bills that are newer than the last time the pmtRecurringPayment job ran. To determine whether a bill is newer, it checks the due date. If the due date is the same as the previous bill, then the bill is considered newer if the doc_date database field or the IVN number is newer, and the bill's payment status is processed. The last process time of that recurring payment is updated.

8 Customizing Payment

This chapter covers the tasks required to customize the Payment module in Oracle Self-Service E-Billing. It includes the following topics:

Architecture of Oracle Self-Service E-Billing Payment

Oracle Self-Service E-Billing Payment is based on J2EE. It uses Servlets and JSPs for the presentation layer and uses Enterprise JavaBeans (EJB) for the business logic layer. It offers the following sets of functions:

- **Enrollment functions.** To enroll users for both viewing bills and paying bills (payment). Examples of user information include account numbers and email addresses, and examples of payment account information include bank account numbers and credit card accounts.
- **Payment functions.** To make payments, set up payment reminders and recurring payments, and so on.
- **Administration functions.** To set up payment jobs, view payment reports and configure Payment Settings.

Figure 10 shows an overview of the J2EE architecture of Oracle Self-Service E-Billing.

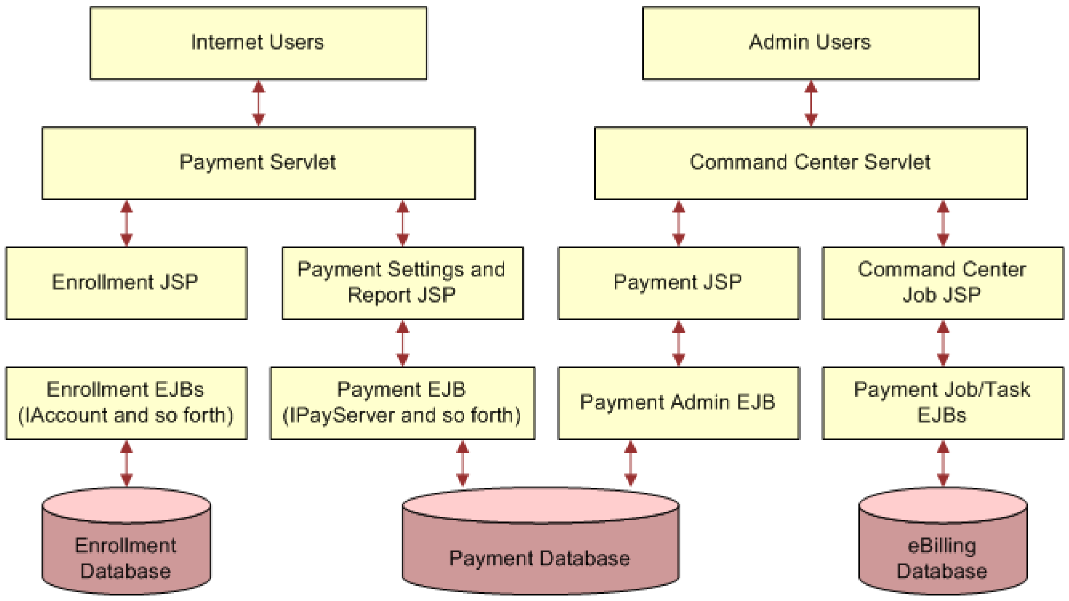


Figure 10. J2EE Architecture of Oracle Self-Service E-Billing

In this architecture, the servlet is responsible for user authentication. After authentication, the servlet forwards the request to JSP pages, which do the bulk of the work. The Oracle Self-Service E-Billing Payment user JSP pages can be categorized into two groups:

- Enrollment JSP pages are responsible for Oracle Self-Service E-Billing Payment user enrollment
- Oracle Self-Service E-Billing JSP pages are responsible for core Oracle Self-Service E-Billing Payment functionality: scheduling payment, setting up recurring payment, and so on.

All Oracle Self-Service E-Billing Payment database access is performed through EJB objects. The JSPs and servlets do not access the database directly. The Oracle Self-Service E-Billing payment batch jobs run from the Command Center. For a list and description of Oracle Self-Service E-Billing Payment jobs, see *Administration Guide for Oracle Self-Service E-Billing*.

This chapter assumes that you have installed and configured Oracle Self-Service E-Billing. It also assumes you understand:

- XML structure and syntax
- J2EE: JSP, HTML, Struts and Tiles

Primary Payment JavaBeans

This topic describes the primary JavaBeans used for payments in the Billing and Payment, and Command Center EAR files.

Table 58 describes the PayServer JavaBean features.

Table 58. PayServer JavaBean Features

JavaBean Feature	Description
Function	Payserver is the main EJB JavaBean that the Billing and Payment application uses to access the Oracle Self-Service E-Billing database.
Remote Interface	Com.edocs.payment.remote.IPayServer
Home Interface	Com.edocs.payment.remote.IPayServerHome
JavaBean Type	Stateless
Jar File	ejb-Payment-payserver.jar

Table 59 describes the PayAdmin Server JavaBean features.

Table 59. PayAdmin Server JavaBean Features

JavaBean Feature	Description
Function	PayAdmin Server is the main EJB that Command Center uses to configure payment settings and display payment reports.
Remote Interface	Com.edocs.payment.remote.IPayAdminServer
Home Interface	Com.edocs.payment.remote.IPayAdminServerHome
JavaBean Type	State-less
Jar File	ejb-Payment-admin.jar

Table 60 describes the IPaymentAccount Manager JavaBean features.

Table 60. IPaymentAccount Manager JavaBean Features

JavaBean Feature	Description
Function	IPaymentAccount Manager is the main EJB used by the Billing and Payment application to access payment account information in the Oracle Self-Service E-Billing database.
Remote Interface	Com.edocs.payment.remote.PaymentAccountManager
Home Interface	Com.edocs.payment.remote.IPaymentAccountManagerHome
JavaBean Type	Stateful
Jar File	ejb-Payment-acctmgr.jar

Table 61 describes the CreditCardSubmit JavaBean features.

Table 61. CreditCardSubmit JavaBean Features

JavaBean Feature	Description
Function	The CreditCardSubmit JavaBean performs the CreditCardSubmitTask.
Remote Interface	Com.edocs.pwc.tasks.ITask
Home Interface	Com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-ccsubmit.jar

Table 62 describes the ChkSubmit JavaBean features.

Table 62. ChkSubmit JavaBean Features

JavaBean Feature	Description
Function	The ChkSubmit JavaBean performs the CheckSubmitTask.
Remote Interface	com.edocs.pwc.tasks.ITask
Home Interface	com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-chksubmit.jar

Table 63 describes the ChkUpdate JavaBean features.

Table 63. ChkUpdate JavaBean Features

JavaBean Feature	Description
Function	The ChkUpdate JavaBean performs the pmtCheckUpdate task.
Remote Interface	com.edocs.pwc.tasks.ITask
Home Interface	com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-chkupdate.jar

Table 64 describes the ConfirmEnroll JavaBean features.

Table 64. ConfirmEnroll JavaBean Features

JavaBean Feature	Description
Function	The ConfirmEnroll JavaBean performs the pmtConfirmEnroll task.
Remote Interface	Com.edocs.pwc.tasks.ITask
Home Interface	Com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-confirm-enroll.jar

Table 65 describes the NotifyEnroll JavaBean features.

Table 65. NotifyEnroll JavaBean Features

JavaBean Feature	Description
Function	The NotifyEnroll JavaBean performs the Notify enroll task.
Remote Interface	Com.edocs.pwc.tasks.ITask
Home Interface	Com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-notify-enroll.jar

Table 66 describes the RecurPayment JavaBean features.

Table 66. RecurPayment JavaBean Features

JavaBean Feature	Description
Function	The RecurPayment JavaBean performs the Recurring payment task.
Remote Interface	Com.edocs.pwc.tasks.ITask
Home Interface	Com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-recur-payment.jar

Table 67 describes the PaymentReminder JavaBean.

Table 67. PaymentReminder JavaBean Features

JavaBean Feature	Description
Function	The PaymentReminder JavaBean performs the payment reminder task.
Remote Interface	Com.edocs.pwc.tasks.ITask
Home Interface	Com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-reminder.jar

Table 68 describes the SubmitEnroll JavaBean features.

Table 68. SubmitEnroll JavaBean Features

JavaBean Feature	Description
Function	The SubmitEnroll JavaBean performs the submit enroll task.
Remote Interface	Com.edocs.pwc.tasks.ITask
Home Interface	Com.edocs.pwc.tasks.ITaskHome
JavaBean Type	Stateful
Jar File	ejb-Payment-submit-enroll.jar

About Recurring Payment Processing

The recurring payment feature is very complex, involving a great deal of business logic. This topic discusses the recurring payment processing in detail.

Recurring payments consist of actions at the front-end (UI) and back end (Command Center jobs). The UI allows a user to insert, update, and delete a recurring payment, and the back end pmtRecurPayment job makes the payment.

The changes to the information in the recurring_payments table are described in Table 69, showing how payment works.

Table 69. Changes in the Recurring Payments Table

Column Name	Comment
AMOUNT_TYPE and AMOUNT	These two columns record how the payment amount is generated. They are only updated through the UI and are used by back-end jobs to calculate how much to pay. The valid values of AMOUNT_TYPE are: ■ **Fixed amount.** Pay a fixed amount and the amount value is specified by AMOUNT column. ■ **Amount due.** Pay amount due on the bill and, AMOUNT column is not used (null). ■ **Minimal due.** Pay minimum amount due on the bill and AMOUNT column is not used (null). ■ **Less due.** Pay the amount due if it is less than the value of the AMOUNT column; otherwise, pay nothing and send email notification. ■ **Upto amount.** Pay the amount due if it is less than the value of the AMOUNT column; otherwise, pay the value of AMOUNT and send email notification.
PAY_INTERVAL DAY_OF_PAY_INTERVAL MONTH_OF_PAY_INTERVAL	These three columns record how the payment date is generated. They are only updated through the UI, and are used by back-end jobs to calculate when to pay. Valid PAY_INTERVAL values are: ■ **Weekly.** User-specified to make payments weekly. The day of week is specified by DAY_OF_PAY_INTERVAL. The MONTH_OF_PAY_INTERVAL is irrelevant. ■ **Monthly**. User-specified to make payments monthly. The day of month is specified by DAY_OF_PAY_INTERVAL. The MONTH_OF_PAY_INTERVAL is irrelevant. ■ **Quarterly.** User-specified to make payments quarterly. The day of month is specified by DAY_OF_PAY_INTERVAL. The month of quarter is specified by MONTH_OF_PAY_INTERVAL (one of 1,2 or 3).

Table 69. Changes in the Recurring Payments Table

Column Name	Comment
START_DATE END_DATE CURR_NUM_PAYMENTS MAX_NUM_PAYMENTS STATUS	These columns determine when to start the recurring payment and when to stop it. START_DATE, END_DATE and MAX_NUM_PAYMENTS can only be updated through the UI. START_DATE is required, but you set only one of the END_DATE (end by that date) or MAX_NUM_PAYMENTS (end when this number of payments is made). The recurring payment STATUS is active when it is created and it has not reached either END_DATE or MAX_NUM_PAYMENTS. When one of them is reached, the STATUS is changed to inactive and the recurring payment will never take effect again. If END_DATE is chosen, NEXT_PAY_DATE (the pay date for the next bill to be paid) is greater than or equal to START_DATE and less then or equal to END_DATE, the bill will be paid. The STATUS is set to inactive if NEXT_PAY_DATE > END_DATE. If MAX_NUM_PAYMENTS is chosen, the STATUS is changed to inactive when CURR_NUM_PAYMENTS reaches MAX_NUM_PAYMENTS.
LAST_PAY_DATE	This is the pay date of last bill. It is set to 01/07/1970 when recurring payment is created to indicate that there is valid information.
NEXT_PAY_DATE	This is the pay date of next bill. When the recurring payment job runs, it schedules a payment with a pay date of NEXT_PAY_DATE. Note, NEXT_PAY_DATE is calculated based on LAST_PAY_DATE and PAY_INTERNAL.

Recurring Payment UI

This topic discusses the actions of the recurring payment UI.

The UI sets up a recurring payment: the UI allows you to insert/update/delete a recurring payment and get back the list of recurring payments.

Figure 11 illustrates the objects involved in the process.

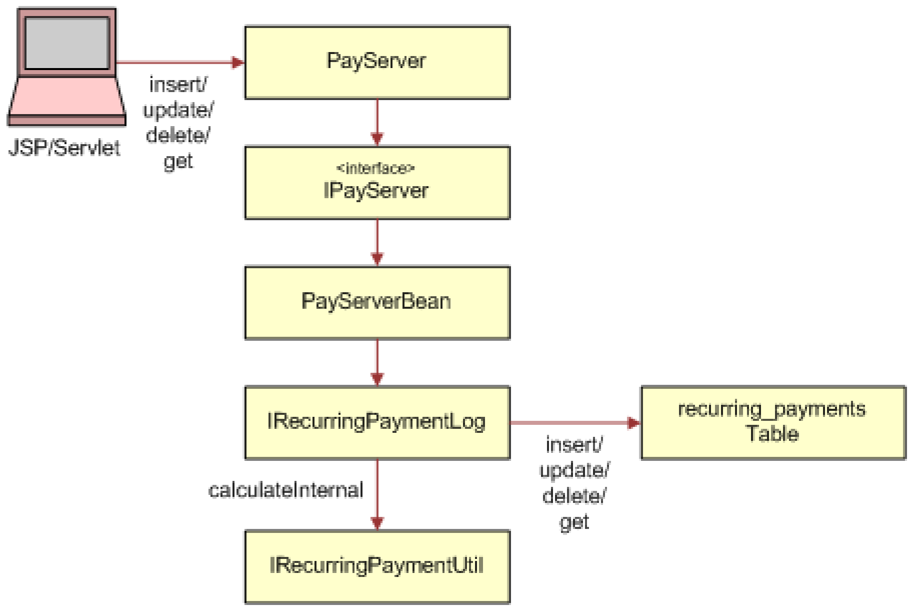


Figure 11. Objects Involved in the Recurring Payment User Interface

Retrieving and deleting recurring payments from the database is straightforward, so the next topics discuss what happens when a recurring payment is inserted or updated.

Insert Recurring Payment From UI

Figure 12 demonstrates what happens when a recurring payment is inserted into database using the UI.

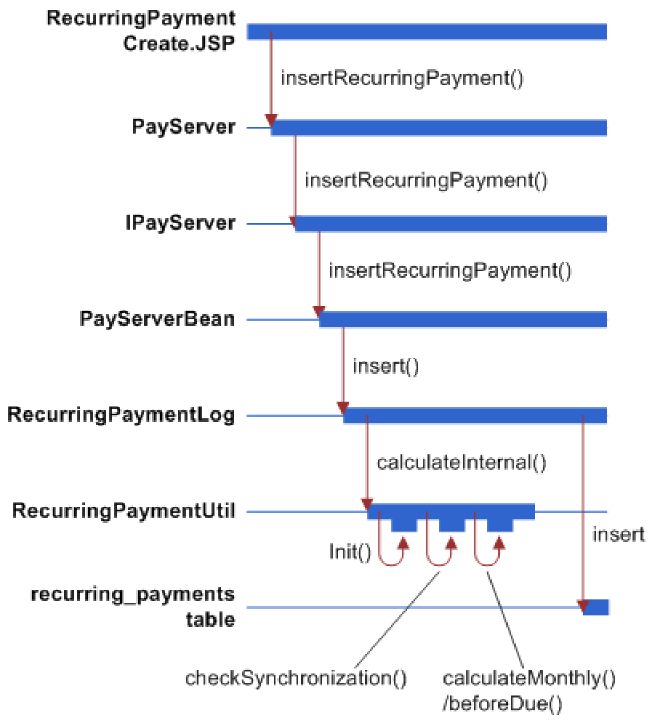


Figure 12. Recurring Payment Inserted into the Database Using the User Interface

The next topic explains RecurringPaymentUtil.calculateInternal(). This method calculates the next_pay_date and status of the recurring payment before it is being inserted into database.

This method calculates the internal states of recurring payment differently for insert and update. For the insert operation, this method performs the following tasks:

1. Call init() method: this method sets some of the recurring payment fields.
    - If the user chooses to end recurring payment by maximum number of payments, set end_date to 01/01/3000 00:00:00.
    - If the user chooses to end recurring payments by a fixed date, set max_num_payments it java.lang.Integer.MAX_VALUE.
    - Set last_pay_date to 01/01/1970 00:00:00; this means no bill has been paid.
    - Set bill_scheduled to Y if the recurring payment is fixed amount and fixed date. Note, in this case, the flag is always true because whenever a payment is made, the next payment is calculated. It has the same effect as making the next bill available immediately.

- Set last_process_time to start_date, which by default must be tomorrow or later. This means that any bills indexed through today (inclusive) will not be picked up by recurring payment. The recurring payment UI checks whether there are unpaid bills when a recurring payment is setup, and reminds the user to make a one-time payment to pay the outstanding bill.

2. Call the checkSynchronization method: Checks whether any required information is missing from recurring payment before inserting it into the database.

3. Check whether the recurring payment has expired by checking the current number of payments against maximum number of payments. Note, this check always return false for insert case.

4. Calculate the next_pay_date by calling one of calculateMonthly(), calculateQuarterly(), calculateWeekly() or calculateBeforeDue() depending on whether pay_interval is “monthly”, “quarterly” or “weekly” or “before_due” respectively.

   - Call calculateMonthly() when pay_interval is “monthly”

     This method calculates the next pay date, which is based on last_pay_date, start_date and day_of_pay_internal. Because last_pay_date is 01/01/1970, the next_pay_date is the nearest date with day_of_pay_internal after the start_date. If date_of_pay_internal is 29, 30 or 31 and there is no such date in that month, the last day of that month is used. After next_pay_date is calculated, it is checked against the end_date. If next_pay_date passes the end_date, the status of the recurring payment is set to “inactive”.

     The following table displays some examples of how next_pay_date is calculated:

Day_of_pay_interval	Start_date	Next_pay_date
1	Sep 10	October 1
10	Sep 10	September 10
15	Sep 10	October 15
31	Sep 10	September 30

   - Call calculateQuarterly() when pay_interval is “quarterly”: works similar to “monthly”
   - Call calculateWeekly() when pay_interval is “weekly”: works similar to “weekly”.
   - Call calculateBeforeDue() when pay_interval is “before due”: because there is no bill yet (bill due date is null), the recurring payment status is set to active and the next_pay_date is set to 01/01/3000.

Update Recurring Payment From the UI

This topic assumes that the UI prevents a user from updating a recurring payment from fixed date to before due date or conversely. If the UI is changed to allow a user to do so, the behavior of recurring payment is not tested.

Figure 13 demonstrates what happens when a recurring payment is updated using the UI into the database.

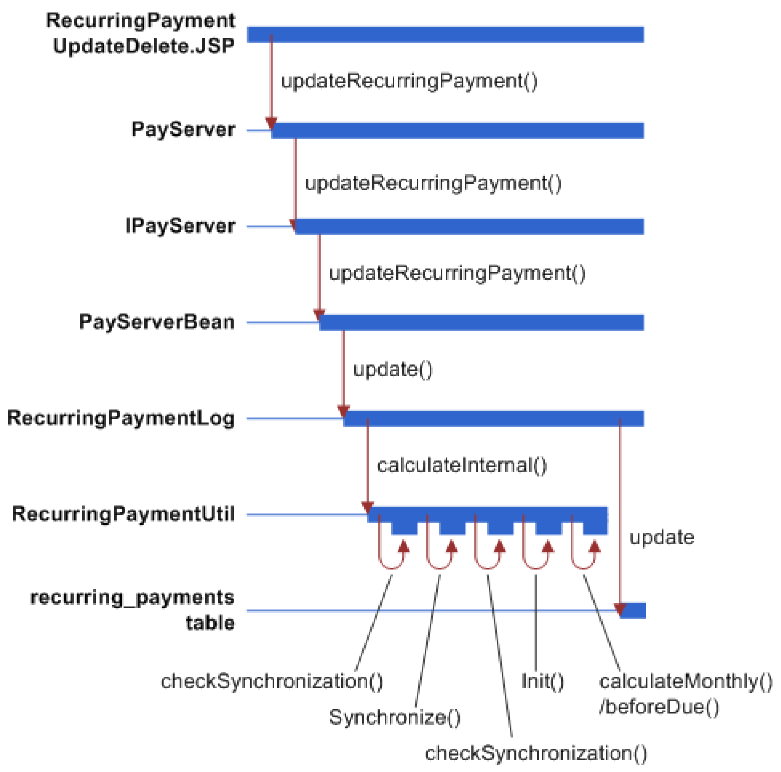


Figure 13. Recurring Payment Inserted into the Database Using the Database

The RecurringPaymentUtil.calculateInternal() method calculates the next_pay_date and the status of the recurring payment before it is inserted into database. Note that this method is also used for update by the back-end job.

The following example shows how this method processes starting with iRecurringPaymentLog.update():

1. Call IRecurringPaymentLog.update()
2. Call RecurringPaymentUtil.calculateInternal()
3. Call checkSynchronization() method to check whether the information required for recurring payment is present.
4. If checkSynchronization() throws an exception indicating missed information, then:
   - Call synchronize() method to read the missed information from the database and populate the missing information into the recurring payment object.
   - Call checkSynchronization() again to make sure the required information has been populated.

- Call init() method: unlike the insert operation, this method checks whether the recurring payment has started or not by checking the last_pay_date (01/01/1970 means not started yet) and then sets the last process time to the start_date of the recurring payment if the recurring payment has not been started. The last process time will not be updated if recurring payment has been started.

5 Check whether the recurring payment has expired by checking the current number of payments against maximum number of payments. If true, set the recurring payment as inactive and return.

6 Calculate next_pay_date and recurring payment status by calling one of calculateMonthly(), calculateQuarterly() or calculateWeekly() based on pay_interval of monthly, quarterly or weekly.

   a Call calculateMonthly() when pay_interval is monthly, to calculate the next pay date.

   b If the last_pay_date is 01/01/1970, then the next_pay_date is calculated based on the start_date and day_of_pay_interval. It is set to the nearest date with day_of_pay_interval as day of month after the start_date. This is the same as the insert case.

   c If the last_pay_date is not 01/01/1970, that means that recurring payment has started, so the next_pay_date is calculated based on the last_pay_date and day_of_pay_interval. It is set to the date one month after the last_pay_date. The calculation does not depend on the current date. For example, if the recurring payment job runs today on October 1, the last_pay_date is Aug 30 and day_of_pay_interval is 30, the next_pay_date will be Sep 30 (not October 30) even though this date is in the past. In the case of fixed date and pay amount due, this can pose a problem if there is no bill for a certain month: the pay date will be in the past. To fix the problem, the recurring payment job will move the last_pay_date ahead by one month if there is no bill for that month. See following discussion for more details about the recurring payment job.

   d If day_of_pay_interval is 29, 30 or 31 and there is no such date in that month, the last day of that month is used.

   After next_pay_date is calculated, it is checked against the end_date and if it passes the end_date, the status of the recurring payment is set to inactive.

- Call calculateQuarterly() when pay_interval is quarterly, it works similarly to monthly.
- Call calculateWeekly() when pay_interval is weekly, it works similar to weekly.
- Call calculateBeforeDue() when pay_interval is before_due:

  First check whether the recurring payment has been synchronized (bill due date not null) and if so, set status to active and next pays date to 01/01/3000 and return.

  Calculate the proposed next pay date by current bill due date and day_of_internal.

  If the proposed next_pay_date is before start_date, set the status of recurring payment to active and next_pay_date to 01/01/3000 and return: the bill will not be paid in this case because it falls outside the effective period of the recurring payment.

  If the proposed next pay date is after end_date, set the status of recurring payment to inactive and set the next_pay_date to 01/01/3000 and return.

  Otherwise, set the status of the recurring payment to active and set its next_pay_date to the proposed next pay date.

Recurring Payment – Back-End Job

The pmtRecurringPayment job gets bills from the Command Center and then schedules payments. The first process is called synchronization and the second process is called scheduling.

Recurring Payment Synchronization

During the synchronization process, the job retrieves a list of recurring payments to be synchronized, and then tries to get the bills for the recurring payments from the Command Center. Figure 14 illustrates this process.

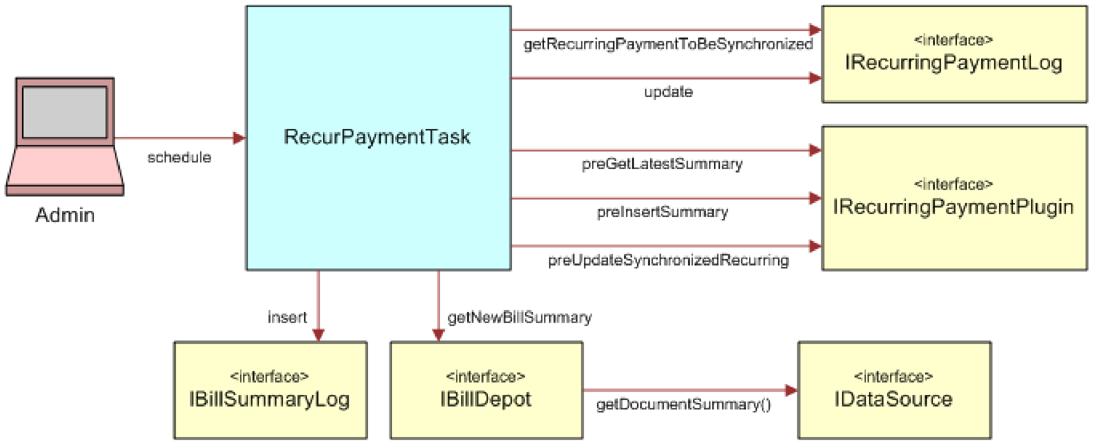


Figure 14. The Recurrent Payment Synchronization Process

Figure 15 shows the synchronization.

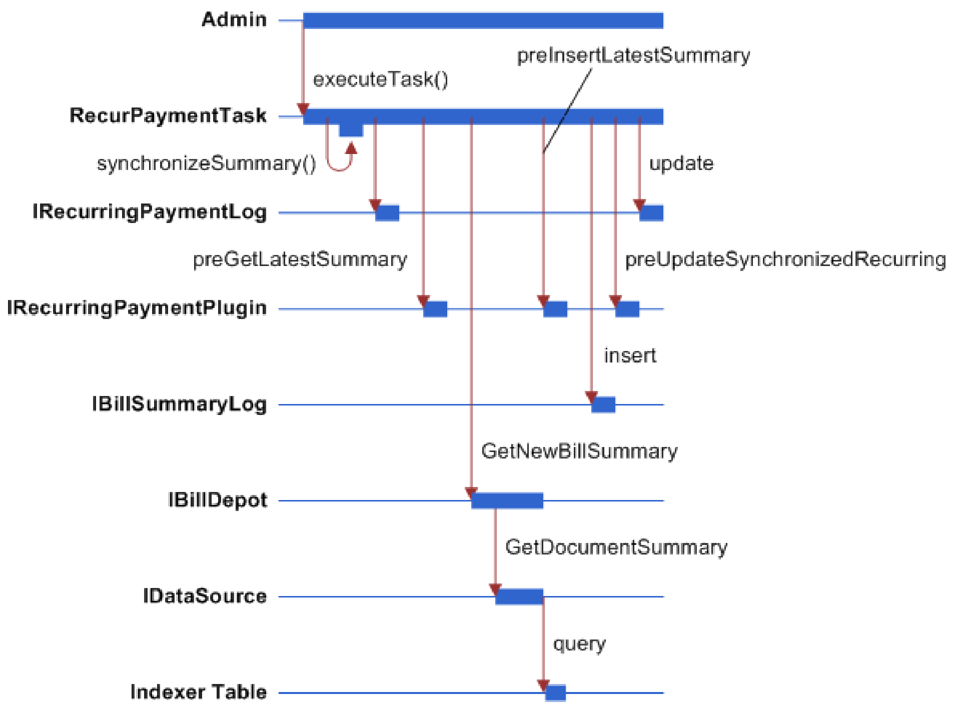


Figure 15. Recurring Payment Synchronization

Synchronization follows these steps:

1 RecurPaymentTask.executeTask() is called when the job runs, which calls RecurringPaymentTask.synchronizeSummary().

2 RecurringPaymentTask.synchronizeSummary() is called. This method does the real work of synchronization and following are the actions taken in this method.

3 IRecurringPaymentLog.getRecurringPaymentsToBeSynchronized() is called to get a list of recurring payments to be synchronized. The query result is affected by the recurring payment job configuration parameter When to Synchronize Recurring Payment with Oracle. When this configuration is Whenever job runs, all the recurring payments are retrieved from the recurring_payments table with payee_id as the job DDN and status as active. If Only after current bill is scheduled is selected, then all payments with the payee_id as job DDN and status as active" and bill_scheduled as Y are retrieved from the recurring_payments table.

4 For each recurring payment, IRecurringPaymentPlugIn.preGetLatestSummary() is called. This method allows the recurring payment plug-in code to decide whether to retrieve bills for a particular recurring payment based on biller-specific business rules.

5 Call RecurPaymentTask.updateRecuringPaymentOnly() if the plug-in rejects this recurring payment by returning PRE_GET_LATEST_SUMMARY_REJECT. This method performs these functions:

   - Update last_process_time to the current time.

- If the recurring payment pay date is fixed date (monthly/quarterly/weekly) and pay amount is based on (minimum) amount due, and no bill arrives for this pay period (bill_scheduled is Y and current time is after the current next_pay_date), the last_pay_date is updated to current next_pay_date. This ensures that if no bill arrives for this pay period; the next bill will be paid on the correct date.
- Call IRecurringPayment.update(): this method calculates the next_pay_date based on the current last_pay_date.

6 Call IBillDepot.getNewBillSummary(). This interface is implemented by com.edocs.payment.imported.eadirect.BillDepot. The BillDepot class retrieves the latest bill summary for the specified account.

- BillDepot.getNewBillSummary() is called, which then calls BillDepot.getSummary().
- BillDepot.getSummary() is called. This method calls IDataSource.getDocumentSummary() to get all the bills indexed for this account between the last_process_time of the recurring payment and the current job run time.
- The returned bills are in the format of name value pairs with value of string. They are interpreted to retrieve due date, amount due and/or minimum amount due.
  - For each bill, if minimum amount due is not null, call BillDepot.preParseMinAmountDue() to give a child class of BillDepot (through the plug-in) a chance to manipulate the minimum amount due string before it is parsed, then it parses min amount due.
  - If the bill’s amount due is not null, call BillDepot.preParseAmountDue() to give child class of BillDepot (through the plug-in) a chance to manipulate the amount due string before it is parsed, then it parses the amount due. If the amount due fails to parse, the bill is ignored.
  - If the bill has no amount due, or its amount due is set to null by preParseAmountDue(), or the amount due failed parsing, then the bill is ignored.
  - If the bill’s due date is not null, call BillDepot.preParseDueDate() to give child class of BillDepot (through the plug-in) a chance to manipulate the due date string before it is parsed, then it parses the due date.
  - If the bill has no due date, or its due date is set to null by preParseAmountDue(), or the due date failed parsing, then the bill is ignored.
- All the successfully parsed bills are compared with the bill summary associated with the current recurring payment, if the summary is not null. The following business rules are used to decide which bill is the latest one:

  The due dates of the bill summaries retrieved are compared and the one with latest due date is chosen.

  For rebill, multiple bills with the same due date can be retrieved. In this case, a rebill is chosen based on the following rules: the one with latest doc date and in case of the same doc date, the one with the larger IVN number. This assumes that a rebill is indexed after its original bill. A rebill will be ignored if its original bill has been paid (the bill_scheduled flag of recurring payment is Y).
- BillDepot.Summary() returns the latest bill if there is one found, otherwise, it returns null.

Recurring Payment Scheduling

Recurring payment scheduling processes as follows:

1. Call RecurPaymentTask.isValidBillSummar() to validate the latest retrieved bill summary. The latest bill summary could be ignored if it has no bill due date, or if the recurring payment is based on minimum amount due but the bill summary has no minimum amount due, or the recurring payment is based on amount due but the bill summary has no amount due.
2. Now you have a valid bill summary. If the payment to the previous bill summary is still in scheduled status, it does the following:
   - Call RecurPaymentTask.cancelScheduledPayment() to cancel this payment. The reason to cancel it is that the new bill summary just retrieved must include the balance of this scheduled bill, cancel the payment so that it will not pay the same bill twice.
   - Call RecurPaymentTask.modifyLastPayDate(): If a recurring payment has a fixed pay date, but the amount is based on amount due or minimum amount due, it is necessary to back date the last pay date because the previous bill payment has been cancelled. Failing to do so will cause the current new bill being paid in next pay interval, not the current one. For example, assume that current bill cycle is October, the previous bill was retrieved on October 10 and is scheduled to pay on October 15. As a result, the last_pay_date and next_pay_date of the recurring payment are updated to October 15 and November 15, respectively. On October 11, a new bill is retrieved and the payment is scheduled. If Oracle Self-Service E-Billing does not back up the last_pay_date, the new bill will be scheduled to pay on November 15. But in this case, it is necessary to pay the bill on October 15 because it is still in the October billing cycle. To fulfill this goal, go back date the last_pay_date to Sep 15 so the next_pay_date will be calculated as October 15, which will be used as the pay date for the new bill.
3. Call RecurPaymentTask.insertNewBillAndUpdateRecurring(), which inserts the retrieved new bill and updates recurring payment accordingly.
   - Call IRecurringPaymentPlugIn.preInsertLatestSummary() before inserting the bill summary in the payment_bill_summaries table.
   - If PRE_INSERT_LATEST_SUMMARY_REJECT is returned from the plug-in, call RecurPaymenTask.updateRecurringPaymentOnly() and return. See step 5 for details about what this method does.
   - Call IBillSummaryLog.insert() to insert this new bill summary.
   - If IBillSummaryLog.insert() throws DuplicateKeyException indicating that this bill is already in the database, so call RecurPaymenTask.updateRecurringPaymentOnly(). See step 5 for details about what this method does.
   - Set the bill_scheduled flag to N if the payment amount is not negative, or Y if it is negative. This means that no credit/reversal will be issued from recurring payment; the credit appears as part of the next bill.
   - Set the bill_id of the recurring payment to the one of the new bill summary.
   - Call IRecurringPaymentPlugIn.preUpdateSynchronizedRecurring().
   - If PRE_UPDATE_SYNCHRONIZED_RECURRING_REJECT is returned from the plug-in, call RecurPaymenTask.updateRecurringPaymentOnly() and return. See step 5 for details about what this method does.

- Call IRecurringPaymentLog.update() to update the recurring payment. The following table lists the information being updated:

Column	Value
last_pay_date	In the case where the pay date is fixed, but amount is based on amount due, last_pay_date could be moved one pay_interval back if a scheduled payment is cancelled because a new bill arrives. Otherwise, last_pay_date will stay the same.
next_pay_date	Next_pay_date will be updated in RecurringPaymentUtil.calculateInternal(). In the case of fixed pay date, it will be updated based on last_pay_date; in case of before due, it will be updated based on the due date of the new bill. See "Update Recurring Payment From the UI" on page 197 for more information.
status	Because next_pay_date is changed, the status could be changed to inactive if next_pay_date falls after end_date.
bill_id	It is set to the bill_id (doc ID) of the bill being inserted into the payment_bill_summaries table.
bill_scheduled	The bill_scheduled flag is set to N if the payment amount is not negative, Y if it is negative.
last_process_time	Set to the current time.

Recurring Payment Scheduling Workflow

Recurring Payment Scheduling Workflow schedules recurring payments for processing with the pmtRecurringPayment job. During scheduling processing, the pmtRecurringPayment job retrieves a list of recurring payments to be scheduled, and then schedules them, as shown in Figure 16.

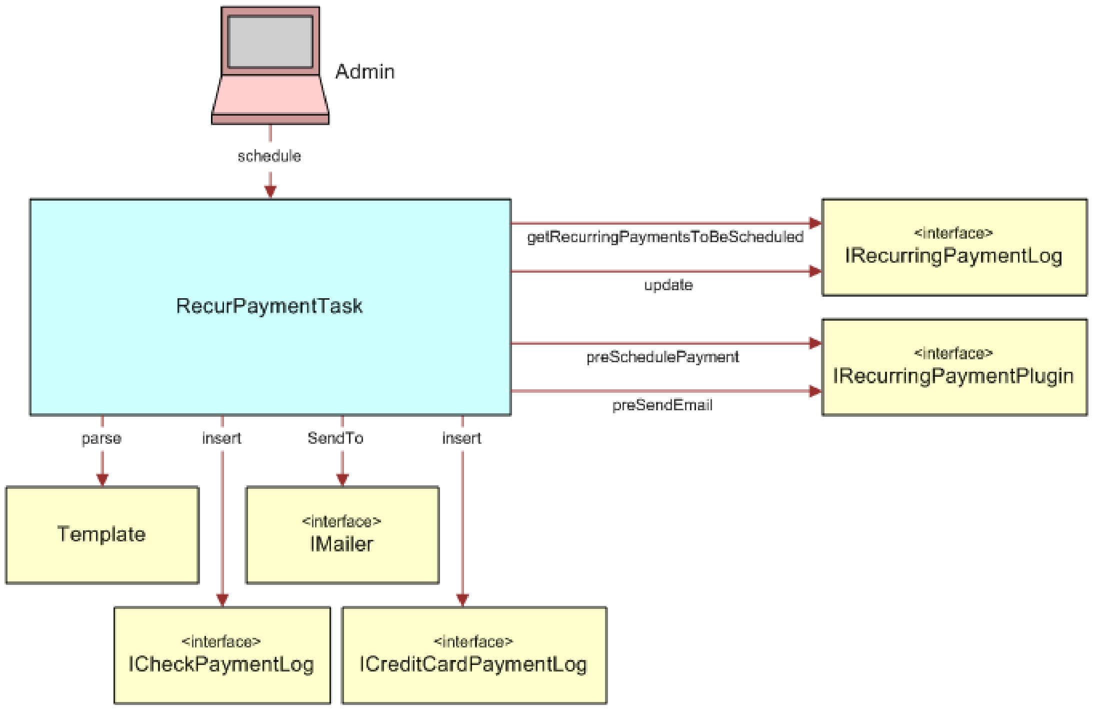


Figure 16. Recurring Payment Scheduling Workflow

Figure 17 shows the action sequence:

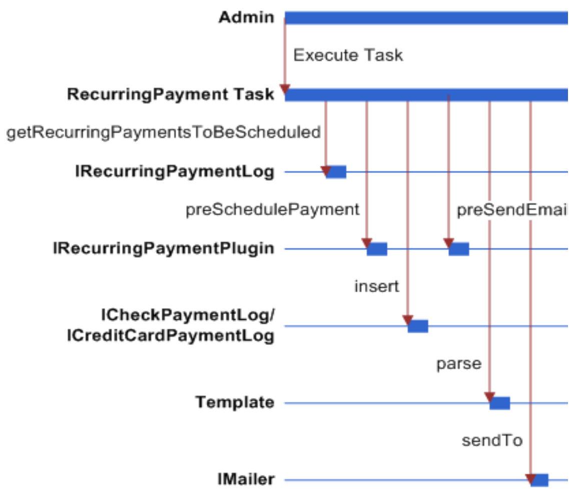


Figure 17. Recurring Payment Scheduling Action Sequence

Workflow Description. This workflow performs the following actions:

1. **RecurPaymentTask.execute().**
2. **RecurringPaymentTask.schedulePayments().** This step performs the scheduling work.
3. **IRecurringPaymentLog.getRecurringPaymentsToBeScheduled().** This step gets a list of recurring payments to be scheduled. The result is affected by the recurring payment job configuration parameter Number of days before pay date to schedule the payment, which is a number, N. The SQL query finds all the recurring payments where the payee_id is the job's DDN reference, bill_scheduled is N and next_pay_date is less than or equal to today plus N.
4. **IPayUserAccountAccessor.getPaymentAccount().** This step gets the current payment account information associated with this recurring payment. A sanity check is done on the retrieved payment account and different actions can be take based on the result:
   - If no payment account has been retrieved, which means it has been deleted from database, then the current recurring payment setup will be de-activated (IRecurringPaymentLog.update() is called to update status to inactive) and no payment is scheduled.
   - If the payment account is a check account, its status is cancelled, and the job configuration parameter "Cancel recurring payment if payment account is canceled?" is true, then the current recurring payment setup is de- activated (IRecurringPaymentLog.update() is called to update status to inactive) and no payment is scheduled.
   - If the payment account is a credit card account, it has expired, and the job configuration parameter Cancel recurring payment if payment account is canceled? is true, then the current recurring payment setup is de-activated (IRecurringPaymentLog.update() is called to update status to inactive) and no payment is scheduled.
5. **RecurPaymentTask.createPaymentTransaction()**. This step creates a new payment transaction (either a check or a credit card) with status as scheduled and pay date and amount as specified by recurring payment setup.
6. **IRecurringPaymentPlugin.preSchedulePayment()**. This step gives PS a change to customize the payment transaction before it is inserted into the database. If this method returns PRE_SCHEDUE_PAYMENT_REJECT, the payment will not be scheduled, and the program return to process next recurring payment. If not, the program will go to the next step to schedule the payment.
7. **ICheckPaymentLog.insert()**. This step inserts a check or ICreditCardPaymentLog.insert() to insert a credit card if the amount of the payment is not negative (it will never be negative because the bill_scheduled will not be N if amount is negative. See job synchronization part for detail). The following table lists part of the payment information inserted into the payment tables:

Column	Value
status	6
pay_date	The next_pay_date (calculated during synchronization process) of the current recurring payment. Because recurring payment will be updated after this insert operation, this value is the same value as last_pay_date of the updated recurring payment.

Column	Value
Amount	This value is decided by amount_type and the amount of the recurring payment. It is calculated when RecurPaymentTask.createPaymentTransaction() is called. It must be the same as the amount column of the recurring payment if amount_type is Fixed. It must be the same as the amount_due or min_amount_due of the bill associated with current recurring payment if amount_type is amount due or minimal due, respectively. If amount_type is “less due”, the payment amount is the amount due of the bill if amount due is less than or equal to the amount column value of the recurring payment. Otherwise, the payment amount value is 0. If amount_type is “upto amount”, then the payment amount is the amount due of the bill if amount due is less than or equal to the amount column value of the recurring payment. Otherwise, the payment amount is the amount column value of the recurring payment.
bill_id	Same as the one from recurring_payment.
Pid	Same as the one from recurring_payment.
payer_id	Same as the one from recurring_payment.
payer_acct_number	Same as the one from recurring_payment.

8 **IRecurringPaymentLog.update()**. This step updates the recurring payment. The following information of the recurring payment will be updated:

Column	Value
Curr_num_payments	Increased by 1
Bill_scheduled	N if pay date is on fixed date (monthly, quarterly or weekly) and pay amount is fixed amount, otherwise Y.
Last_pay_date	The last_pay_date is set to the current next_pay_date of the recurring payment.
Next_pay_date	After last_pay_date is set to the current next_pay_date, the next_pay_date is calculated again by RecurringPaymentUtil.calculateInternal(). If the payment is using a fixed pay date (weekly, quarterly or weekly), then next_pay_date is calculated and moved to the next pay date in the next pay interval. In case of before due date, the next pay date will be calculated based on the current due date (whose bill has been paid), so this next_pay_date has no meaning until the next bill is synchronized.
Status	Status is recalculated and will be changed to inactive if next_pay_date is after end_date, or curr_num_payments is greater than max_num_payments.

9. **IRecurringPaymentPlugIn.preSendEmail()**. This step lets the plug-in customize the email being sent out. The email is not sent out if this method returns PRE_SEND_EMAIL_REJECT.
10. **Template.parse()**. This step parses the email template and generates the content of email.
11. **PaymentMailer.send()**. This step sends email.

Recurring Payment FAQ

This topic answers a few common questions about recurring payment.

- **Why is my current bill not paid by recurring payment after I set up my recurring payment?**

  The recurring payment start date can only start from tomorrow, so the last_process_date is set to start from tomorrow. This means all the bills indexed before today will not be processed by the recurring payment. The reason is that, currently, there is no reliable way for recurring payment to know whether the current bill has been paid or not. The user might have paid it through a one-time payment or through paper check. To avoid paying the bill twice, recurring payment will only start processing bills indexed since tomorrow.

  When a recurring payment is created, the JSP page checks whether there are any indexed bills for the account. If so, Oracle Self-Service E-Billing retrieves the latest bill for the account. Oracle Self-Service E-Billing also checks whether the latest bill has been paid by checking its doc ID against the bill_id of Oracle Self-Service E-Billing Payment tables. If there is no match, it is reasonable to assume that the bill has not been paid, so Oracle Self-Service E-Billing prompts the user to make a one-time payment to pay that bill.

- **What assumptions does recurring payment make?**

  Recurring payment assumes that the bill balances are accumulative; that is, the bill of this billing cycle includes the balance of the bill from previous billing cycle, and the later bill has a due date after that of the previous bill (the only situation where the same due date can happen is for a rebill).

  Recurring payment also assumes that each bill has a date indicating the chronological order of bills; this is usually the date when the bill arrives. For example, in the case of the Command Center, doc date can be used to indicate the chronological order of arriving bills. In the case of external billing software, other dates such as statement date can be used for this purpose. When recurring payment synchronizes with the Command Center or other billing software, it must retrieve the latest bill issued between the last_process_time and current time. This chronological date of bills (doc date or statement date) is used to guarantee that functionality.

- **Can recurring payment work with billing software other than the Command Center?**

  Yes. Recurring payment assumes nothing specific to the Command Center and the only action to take is to reimplement the IBillDepot API. The billing software must meet assumptions stated in item 2.

- **Must the bills have due dates?**

  Yes, if the recurring payment is not fixed date and fixed amount. The due date is used to decide which bill is the latest one to pay. For the Command Center, you must index the due date or some date equivalent to use as the due date.

- **What is the Rebill feature? How do I enable it?**

  The Rebill feature lets you issue the same bill multiple times during one billing cycle to handle adjustments. All the rebills must have the same due dates. To decide which rebill is the latest bill to pay, the current IBillDepot implementation considers the latest one to be the one with latest doc date. If there is more than one bill with same doc date, the bill with highest IVN number is chosen. Note, this implementation assumes that a later rebill is always indexed after a previous rebill, and no rebills will be put together in one data file. This would cause the rebills to have same doc date and IVN number. If you want to consider other factor such as amount for making the decision, you must reimplement IBillDepot.

  The Rebill feature is enabled by job configuration parameter When to synchronize with Oracle. To use the Rebill feature, you must choose Whenever the job runs. If you do not use the Rebill feature, you can choose either Whenever the job runs, or Only after current bill is scheduled.

  Technically, there is not much difference between a regular bill and rebill. The major difference is the logic required to decide which rebill is the latest bill, which goes beyond checking bill due date. You can think about non rebill as a special case of rebill; rebill allows the same bill to appear more than once in a single billing period, but non rebill appears only once. The code and programming logic does not distinguish between these two cases.

- **When rebill is not involved, is there any difference between the job configuration options for the job configuration parameter When to Synchronize with Oracle?**

  It must not affect functionality, and you can choose either of them. Consider the following two possibilities:

  - Performance can deteriorate if you choose Whenever the job runs because instead of waiting until current bill is scheduled, the job will try to synchronize with the Command Center for each recurring payment. This can be especially true if you are talking with billing software other than the Command Center that might have a slow connection.

  - A scheduled payment can be cancelled because of an unexpected early-arrival of next bill. Because the user only wants to pay the latest bill, the scheduled payment will be canceled and the new bill will be scheduled.

- **Why and when can a scheduled payment be cancelled by recurring payment job?**

  The cancellation of a scheduled payment can only happen when the job configuration When to synchronize with Oracle is set to Whenever job runs.

  It can happen because of two reasons:

  - The first case is: (for rebill) after the original bill is scheduled, but before it is processed, the rebill arrives. In this case, the original payment will be cancelled, and the rebill will be scheduled.

  - Second, the bill of this billing cycle is still scheduled, but before it is processed, the bill of next billing cycle arrives (early). In this case, this bill's payment is cancelled and the next bill is scheduled.

  In case of fixed pay date and pay amount due, if a scheduled payment is cancelled, move the last_pay_date and next_pay_date back by the pay_interval before the next bill is scheduled. This ensures that the next bill is paid with the same pay date as the previous bill.

- **In the case of fixed pay date and pay amount due, what happens if there is no bill for this billing cycle?**

  Recurring payment can never be triggered for a billing cycle if there is no bill, or if the bill's balance is negative (recurring payment does not issue credit). For example, a user sets to pay the bill's amount due on the 15th of each month, and current month is Oct. The next_pay_date will be set to October 15. However, if no bill arrives before October 15, then after October 15, the next_pay_date will be changed to November 15 to ensure that the bill arrives it will be paid in the next pay period. Otherwise, the user might end up paying the November bill with the October pay date.

- **Will recurring payment make a pay if the balance is negative?**

  No. Instead, recurring payment assumes that this credit will roll into the balance of next bill. However, a zero dollar payment will be made if the balance is zero.

- **Can I set up a recurring payment to pay from multiple payment accounts?**

  No, you can only pay from one payment account for each recurring payment.

- **Why does the default recurring payment update UI limit some options after the recurring payment is started?** For example, it is not possible to switch from pay on fixed date to pay before due.

  The logic to calculate next pay date becomes extremely complicated, so it is disallowed. If a custom UI does allow such an update, the behavior is undefined.

- **What happens if my credit card account expires?**

  The recurring payment does not schedule a payment. It is then de-activated and an email is sent to the users to indicate that they must update their credit card account information. In this case, the user must log in to cancel the inactive recurring payment and create a new one.

- **Why was my bill not scheduled?**

  This is the most often asked question, but there can be many causes. Follow these steps to debug this problem. To start, review the recurring payment logic steps described previously.

  First, check whether this is a false alarm. A bill can be synchronized, but yet scheduled. Also check the next_pay_date to see whether it reflects the correct pay date for the bill.

  If the bill is not even synchronized, check whether it has been indexed;

  If indexed, check whether it falls into the synchronization period. Only bills whose doc date fall between last_process_time and the current time will be considered.

  Check whether this bill has valid information. For example, whether its due date, amount due are valid parse-able strings. A bill with invalid bill information or with negative balance will not be paid.

  Even though this is a valid bill, it might not still be paid because its due date is before the due date of the current bill associated with the recurring payment.

  Custom plug-ins might be a factor. The custom code might not have been thoroughly tested, so check the plug-in the code carefully. Especially if the custom plug-in is manipulating the bill's due date or amount due or recurring payment information directly.

  The bill cannot be scheduled because the payment account has been cancelled or deleted or de-activated.

- **Will a single recurring payment failure fail the whole recurring payment job?**

  No. If it does, it is a bug. If this happens, create a service request (SR) on My Oracle Support. Alternatively, you can phone Oracle Global Customer Support directly to create a service request or get a status update on your current SR. Support phone numbers are listed on My Oracle Support.

- **What is bill ID?**

  A unique ID used to identify each bill. In the Command Center, it is the doc ID.

- **What is last process time? What is it used for?**

  The time when the last recurring payment job ran. It is used to ensure that a bill is only retrieved once from the Command Center. Oracle Self-Service E-Billing Payment only retrieves bills indexed between the last process time and the current time. That is, bills whose doc date is greater than or equal to the last process time and less than or equal to the current time. The last process time only contains date information (because the doc date only contains date information).

- **What happens if a bill is indexed twice?**

  This is similar to rebill. The two bills have the same due dates, but the second indexing produces a later doc date, or a larger IVN, if they are indexed in the same day.

  If When to synchronize with is set to Whenever job runs, this is a true rebill case, and will be treated as a rebill.

  If When to synchronize with Oracle is set to After current bill is scheduled, the second indexed bill will be ignored during next round of synchronization.

About Payment Plug-Ins

The Oracle Self-Service E-Billing Payment plug-in is a callback, which allows you to add code to extend the functionality of Oracle Self-Service E-Billing. The following payment plug-ins are included with Oracle Self-Service E-Billing:

- IAchCheckSubmitPlugIn for the ACH cartridge when submitting checks to ACH.
- IPayPalCreditCardSubmitPlugIn for the PayPal Payflow Pro cartridge when submitting credit cards to PayPal Payflow Pro.
- IPaymentReminderPlugIn for the job pmtPaymentReminder
- IRecurringPaymentPlugIn for the job pmtRecurPayment

Each plug-in comes with a default implementation. It is recommended that you derive your plug-in from the default implementation to ensure that future updates to the plug-in will not break your code. The plug-ins and sample code are provided in Sample Plugin Code on page 87.

ACH Check Submit Plug-In

The ACH cartridge supports a plug-in to modify ACH file generation. When the pmtCheckSubmit job runs for ACH, it calls the methods of the implementation of IAchCheckSubmitPlugIn (defined in Payment Settings) during numerous events. The default implementation is AchCheckSubmitPlugIn, which does nothing.

Figure 18 shows the workflow for the pmtCheckSubmit job plug-in.

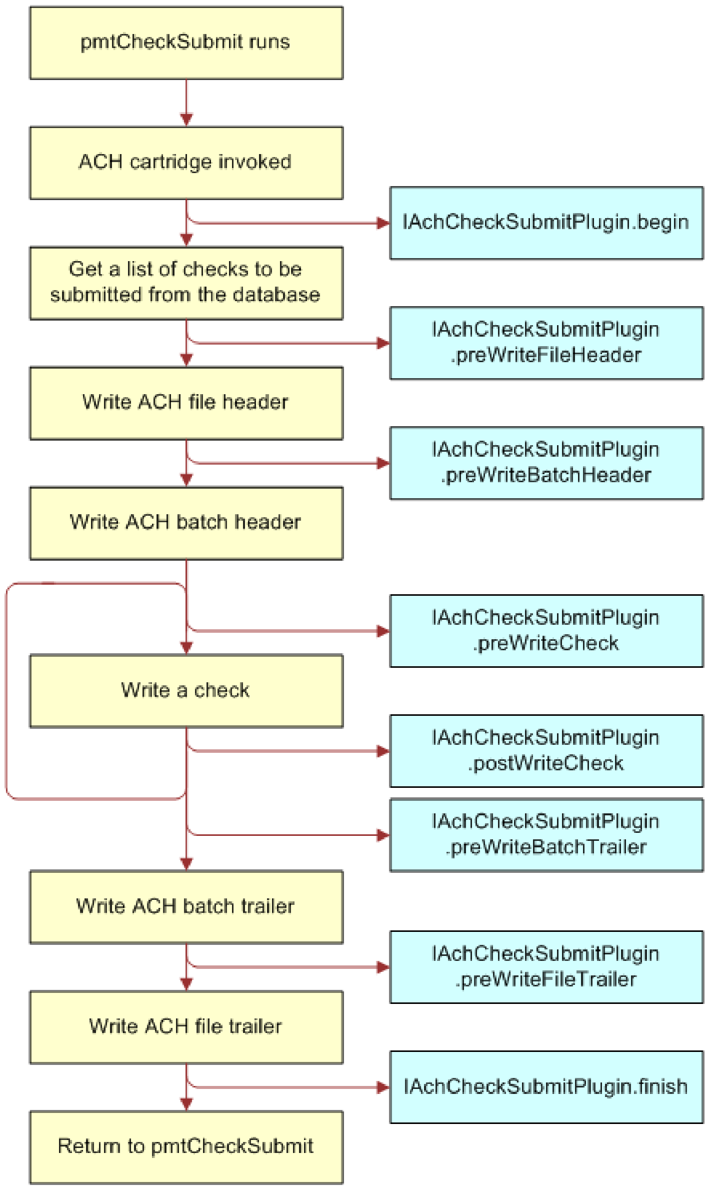

pmtCheckSubmit runs
ACH cartridge invoked
IAchCheckSubmitPlugin.begin
Get a list of checks to be submitted from the database
IAchCheckSubmitPlugin .preWriteFileHeader
Write ACH file header
IAchCheckSubmitPlugin .preWriteBatchHeader
Write ACH batch header
IAchCheckSubmitPlugin .preWriteCheck
Write a check
IAchCheckSubmitPlugin .postWriteCheck
IAchCheckSubmitPlugin .preWriteBatchTrailer
Write ACH batch trailer
IAchCheckSubmitPlugin .preWriteFileTrailer
Write ACH file trailer
IAchCheckSubmitPlugin.finish
Return to pmtCheckSubmit

Figure 18. pmtCheckSubmit Job Plug-in Workflow

Writing a Plug-in

You can use the pmtCheckSubmit plug-in to change the default name of the ACH file, create a remittance file in addition to the standard ACH file, deny a check or change the default information put into the ACH file. Create your own implementation to accomplish these tasks. See "Accessing Oracle Self-Service E-Billing Javadoc" on page 31 for information about writing an implementation of IAchCheckSubmitPlugIn.

To create your own implementation

1. Derive your implementation from the default implementation AchCheckSubmitPlugIn.
2. Overwrite the methods whose behavior you want to change.
3. When compiling, include Payment_common.jar and Payment_client.jar into your java classpath.
4. Package this class into Payment_custom.jar of each EAR file. See "Packaging Oracle Self-Service E-Billing Payment Custom Code" on page 245 for information about redeploying EAR files.
5. Change the Payment Settings to point to your new class.

Using a Plug-in to Write ACH Addenda Records

You can use the pmtCheckSubmit plug-in to write addenda records for ACH. The implementation called AddendaCheckSubmitPlugIn gets the invoice information of a payment and writes them out as addenda records. For more information about this class or its implementation details, see "Accessing Oracle Self-Service E-Billing Javadoc" on page 31, and then follow the steps in "Writing a Plug-in" on page 213 to write your own implementation.

PayPal Payflow Pro Credit Card Payment Plug-in

Unlike the ACH plug-in, the PayPal Payflow Pro credit card plug-in CreditCardSubmit is invoked from both the front end (when an instant credit card is made) and the back end (when credit card submit job runs). This plug-in allows you to audit the credit card payment, deny it, or even changes the HTTP request sent to PayPal Payflow Pro HTTP server. Check the API IPayPalCreditCardSubmitPlugIn for details.

Figure 19 shows the workflow of the plug-in when an instant credit card payment is submitted:

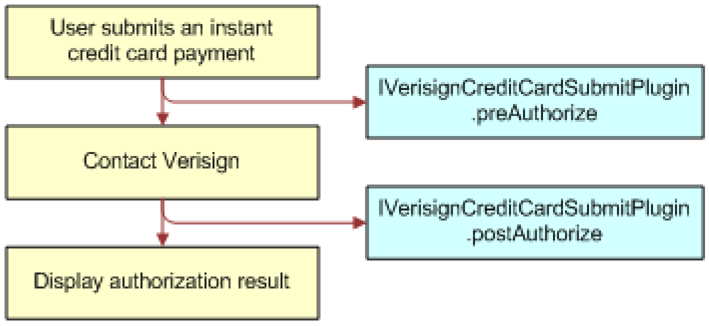

User submits an instant credit card payment
IVerisignCreditCardSubmitPlugin .preAuthorize
Contact Verisign
IVerisignCreditCardSubmitPlugin .postAuthorize
Display authorization result

Figure 19. Workflow of the Plug-in when an Instant Credit Card Payment is Submitted

Figure 20 shows the workflow of the plug-in when the pmtCreditCardSubmit job runs for PayPal Payflow Pro:

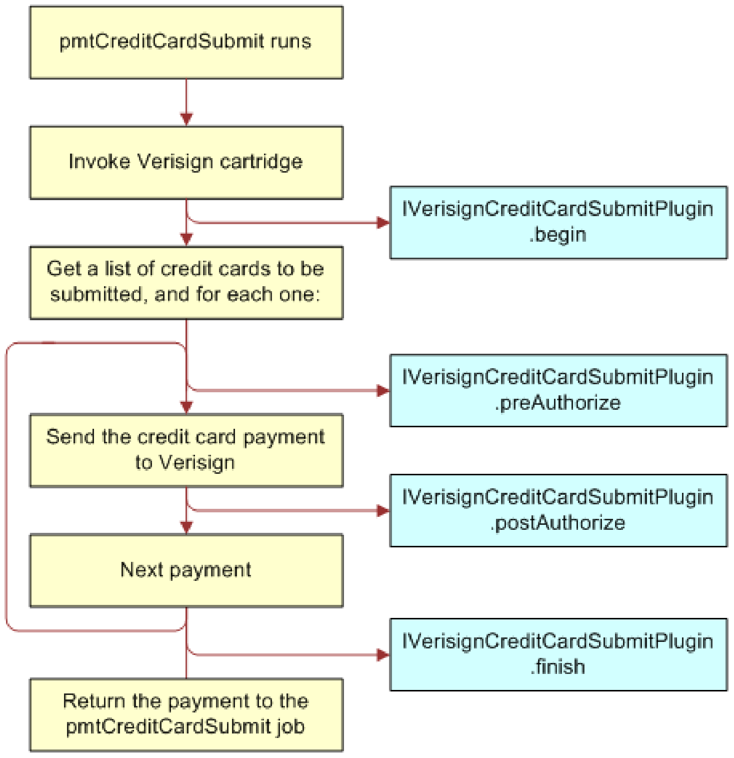

pmtCreditCardSubmit runs
Invoke Verisign cartridge
IVerisignCreditCardSubmitPlugin .begin
Get a list of credit cards to be submitted, and for each one:
IVerisignCreditCardSubmitPlugin .preAuthorize
Send the credit card payment to Verisign
IVerisignCreditCardSubmitPlugin .postAuthorize
Next payment
IVerisignCreditCardSubmitPlugin .finish
Return the payment to the pmtCreditCardSubmit job

Figure 20. Workflow of the Plug-in when the pmtCreditCardSubmit Job Runs for PayPal Payflow Pro

Writing a Credit Card Plug-in

The default implementation of IPayPalCreditCardSubmitPlugIn, PayPalCreditCardSubmitPlugIn, does nothing. You must write an implementation.

To write your own implementation

1 Derive your implementation from PayPalCreditCardSubmitPlugIn.

2 Overwrite the methods for which you want to change the default behavior.

3 When compiling, include Payment_common.jar and Payment_client.jar in your java class path.

4 Package this class into Payment_custom.jar of each EAR file.

5 Change the Payment Settings of that DDN to use the new plug-in implementation.

Payment Reminder Plug-in

The payment reminder plug-in is invoked when the pmtPaymentReminder job runs. pmtPaymentReminder performs the following functions:

- Regular payment reminders
- Check status notification
- Credit card status notification

There are corresponding plug-ins for the preceding tasks. Refer to com.edocs.payment.tasks.reminder.IPaymentReminderPlugIn for details.

Figure 21 shows the workflow for the plug-in of the pmtPaymentReminder job:

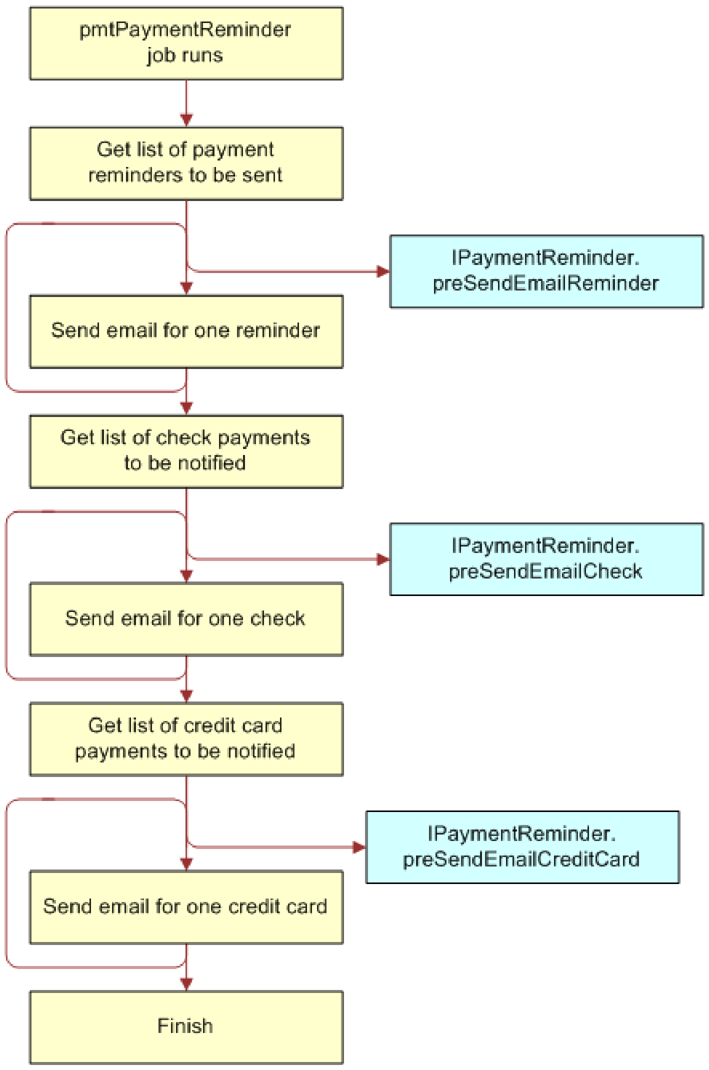

pmtPaymentReminder job runs
Get list of payment reminders to be sent
IPaymentReminder. preSendEmailReminder
Send email for one reminder
Get list of check payments to be notified
IPaymentReminder. preSendEmailCheck
Send email for one check
Get list of credit card payments to be notified
IPaymentReminder. preSendEmailCreditCard
Send email for one credit card
Finish

Figure 21. pmtPaymentReminder Job Plug-in Workflow

Creating a pmtPaymentReminder Plug-in

The default plug-in implementation, com.edocs.payment.tasks.reminder.PaymentReminderPlugIn, does nothing.

To implement your own plug-in

1. Derive your implementation class from PaymentReminderPlugIn.
2. Overwrite the methods for you want to change behavior.
3. When compiling, include Payment_common.jar and Payment_client.jar in your javac class path.
4. Package this class into Payment_custom.jar of each EAR file.
5. Update the pmtPaymentReminder job configuration to use the new class.

Recurring Payment Plug-in

The recurring payment plug-in is called when the pmtRecurPayment job runs. You can use this plug-in to prevent a recurring payment from being scheduled based on business rules. Or, you can extract some indexed fields from the index table and put them into the payment being scheduled. The implementations: com.edocs.tasks.payment.recur_payment.RecurringPaymentPlugIn, is the default one and it does nothing.

The file SampleRecurringPlugin.java provides an example implementation. Figure 22 shows the workflow of recurring payment and how the plug-in works.

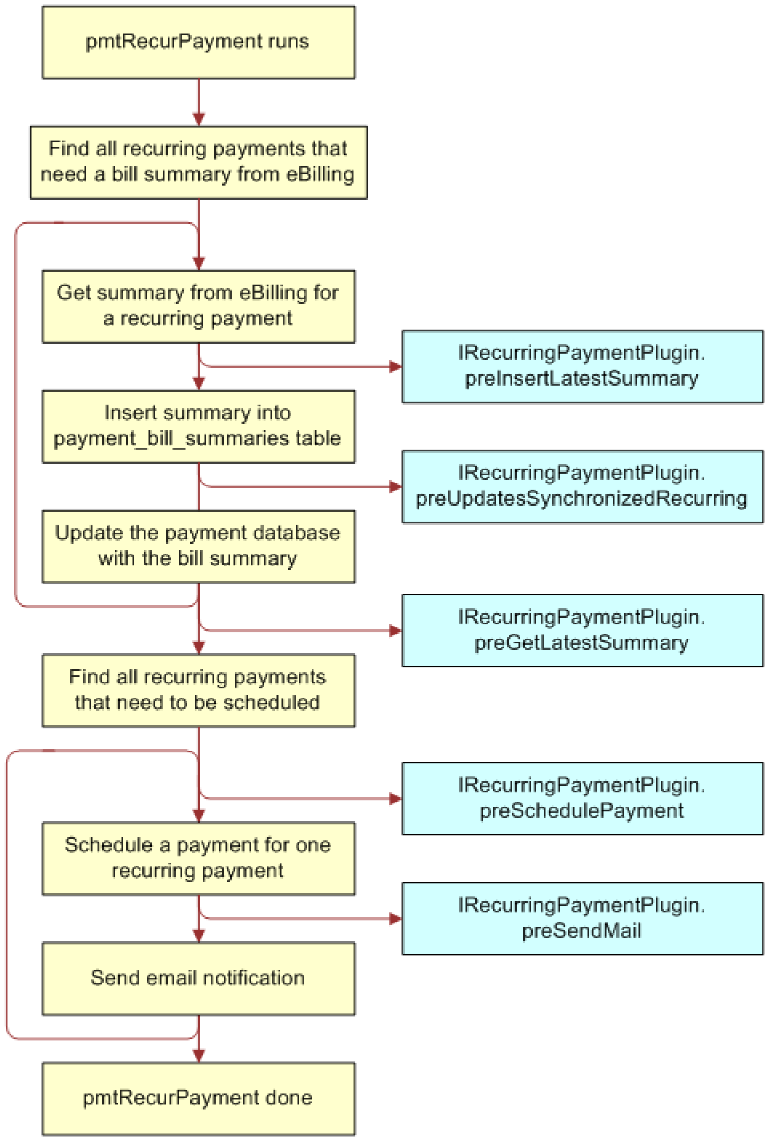

pmtRecurPayment runs
Find all recurring payments that need a bill summary from eBilling
Get summary from eBilling for a recurring payment
IRecurringPaymentPlugin. preInsertLatestSummary
Insert summary into payment_bill_summaries table
IRecurringPaymentPlugin. preUpdatesSynchronizedRecurring
Update the payment database with the bill summary
IRecurringPaymentPlugin. preGetLatestSummary
Find all recurring payments that need to be scheduled
IRecurringPaymentPlugin. preSchedulePayment
Schedule a payment for one recurring payment
IRecurringPaymentPlugin. preSendMail
Send email notification
pmtRecurPayment done

Figure 22. Recurring Payment Workflow

Writing a Plug-in

The default plug-in implementation, com.edocs.payment.tasks.recur_payment. RecurringPaymentPlugIn, does nothing.

To implement your own plug-in

1 Derive your implementation class from RecurringPaymentPlugIn.

2 Overwrite the method that you want to change behavior of.

3 When compiling, include Payment_common.jar and Payment_client.jar in your javac class path.

4 Package this class into Payment_custom.jar of each EAR file.

5 Update the pmtRecurPayment job configuration to use the new class.

Populating Index Fields in Payment Flexible Fields

The plug-in com.edocs.paymenttasks.recur_payment.SampleRecurringPlugIn demonstrates how to use a plug-in to populate the flexible fields of the Oracle Self-Service E-Billing Payment database (ICheck or ICreditCard) with the indexed information from the indexer table.

Customizing Oracle Self-Service E-Billing Payment Template Files

Oracle Self-Service E-Billing provides a template engine to generate text messages, such as email, ACH files, and A/R files. This topic describes how to use Oracle Self-Service E-Billing Payment templates to customize those text messages.

Oracle Self-Service E-Billing Payment Template Engine

The Oracle Self-Service E-Billing Payment templates provide a generic template mechanism based on Java reflection. The template engine generates custom text output based on the templates. Similar to JSP, the template engine replaces the special placeholders inserted into the text file with the values of Java objects. For more detailed API documentation, see "Accessing Oracle Self-Service E-Billing Javadoc" on page 31 for details on accessing *Oracle Self-Service E-Billing* Javadoc.

The Template engine hosts a pool of objects in its context in the form of a hash table. You can refer to the variables in that context by their names. For example, there is a Check object whose name is check. You can refer to that object as %check%. This means replace %check% with the string returned from check.toString(). This is true for all Java objects except java.util.Date, where getTime() is called and inserts a long value that is the number of milliseconds since January 1, 1970, 00:00:00 GMT. If a method returns void, then nothing prints.

The content of the message consists of text plus resolved placeholders. Placeholders are Java variables, which are Payment hosted objects including their attributes and methods.

Enclose all template variables with two percent signs (%%). To escape %, use %%. For example, %%40 means %40.

In addition to referring to variables, you can also access an object's public fields and methods. The valid reference is `%name.field%`, `%name.method(param1, param2, ...)%`, where each parameter to a method can be `name`, `name.field`, or `name.method(param1, param2, ,,,)`. The number of parameters is unlimited and an arbitrary level of method nesting is allowed (nesting means that a method's return value is used as a parameter when calling another method). For example, suppose there are two objects in contexts: buf which is a StringBuffer, and str which is a String. The following references are valid: `%buf%`, `%buf.append(str)%`, `%buf.append(str.toString())%`.

A static field or method can be accessed directly without instantiating an object. For example, java.lang.Integer has a static field called MIN_VALUE and a static method called parseInt. You can refer to them as %java.lang.Integer.MIN_VALUE% or %java.lang.Integer.parseInt("12.34")%.

All variables must be preset by calling putToContext on the Template class. Some variables are already set by Oracle Self-Service E-Billing Payment which you can use directly. But you can also put your own variables into the context:

```
%template.putToContext("buf", new java.lang.StringBuffer())%
```

This means to put a new StringBuffer object called buf into the template context. You can then refer to this object by its name:

```
%buf.append("abc")%
```

This appends "abc" to the end of the StringBuffer's value.

The current Oracle Self-Service E-Billing Payment engine has some limitations. It cannot do math operations, such as x plus y. You must call a Java method to do math operations. Another limitation is that it does not allow you to concatenate method calls, for example: %variable.method().method()%. You must write your own Java method to do method concatenation.

Included with the Oracle Self-Service E-Billing Payment package, there are a few utility classes to help you overcome the weakness of Oracle Self-Service E-Billing Payment Template Engine. These classes are:

```
com.edocs.payment.util.DecimalUtil
com.edocs.payment.util.DateUtil
com.edocs.payment.util.StringUtil.
```

One useful method in StringUtil is concat, which you declare and use as follows:

```
public static String concat(String s1, String s2, String s3)
%com.edocs.payment.util.StringUtil.concat(s1,s2,s3)%
```

Remember, you cannot do %s1.concat(s2).concat(s3)% inside a template, instead, you must call this function from template:

```
%com.edocs.payment.util.StringUtil.concat(s1,s2,s3)%.
```

Another useful method is format() from DateUtil class. This method helps format a Date object into different display formats. For example: %com.edocs.payment.util.DateUtil.format("MMM dd, yyyy", check.getPayDate())% formats a check's pay date to display as "Jan 01, 2000." For a complete list of possible date formats, please check the JDK document about java.text.SimpleDateFormat.

When writing customized Java code, it is strongly recommended that you use static methods as frequently as possible, so you can call them directly from a template without creating an instance of that object first. For example, by default, the individual ID field of an ACH entry detail field is populated with the customer's account number using %check.getPayerAcctNumber()%. The returned result is 16 bytes long, but the account number is 15 bytes, so you must truncate the retrieved value.

To create a java class to do truncation and enable it in the Oracle Self-Service E-Billing Payment Template Engine

1 Write a Java class:

```
package com.edocs.ps;
public class MyUtil {
  public static String truncate(String s){
     return s.substring(1);
  }
}
```

2 Compile the class and put it into Payment_custom.jar of each EAR file, then redeploy the EAR files.

3 Refer to this class in a template as follows:

```
%com.edocs.ps.MyUtil.truncate(check.getPayerAcctNumber())%
```

Customizing Email Templates

The Payment module uses template files to generate customized text that will be sent in a notification email. The email templates can be customized for you by Oracle Professional Services, or you can customize them yourself.

Table 70 describes the email notification templates used in Payment.

Table 70. Email Notification Templates

Type of Notification	Name of the Task that Uses the Notification	Template File
Reminder to pay bills and the status of the checks	pmtPaymentReminder	paymentReminder.txt
Enrollment status	pmtNotifyEnroll	modifyEnroll.txt
Recurring payment was scheduled	pmtRecurPayment	recurringNotify.txt
Payment Command Center job status	All Payment jobs	notifyPaymentTask.txt
Credit card expiration	pmtCreditCardExpNotify	CCExpNotify.txt

For UNIX, the default path to the email template files is the *EDX_HOME*/payment/lib/payment_resources/ directory (the *EDX_HOME*\payment\lib\payment_resources directory on Windows). In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.

The email templates use a programming structure that works similar to JSP (but is not JSP). The template language includes a list of placeholders that refers to Java objects, which are hosted by Payment. It also includes some logic control directives such as IF and LOOP.

For more information about template classes, see "Accessing Oracle Self-Service E-Billing Javadoc" on page 31 for details on accessing *Oracle Self-Service E-Billing* Javadoc.

Oracle Self-Service E-Billing Payment Reminder Template

Oracle Self-Service E-Billing Payment reminder messages are generated based on PaymentReminder.txt, which resides in the *EDX_HOME*/payment/lib/payment_resources/ directory (the *EDX_HOME*\payment\lib\payment_resources directory on Windows). In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing.

This template is used for regular payment reminder and email notifications for processed, returned or failed payments.

Table 71 describes the Oracle Self-Service E-Billing Payment reminder template variables.

Table 71. Payment Reminder Template Variables

Variable	Type	Description
check	ICheck	The ICheck object being notified, valid only when isCheck is true.
creditcard	ICreditCard	The ICreditCard object being notified, valid only when isCCard is true.
isCCard	Boolean	True means this is for credit card status notification.
isCheck	Boolean	True means this is for check status notification.
isFailed	Boolean	True means the payment has failed to process (isFailedAuthorize).
isPaid	Boolean	True means the check has been paid or cleared.
isProcessed	Boolean	True means the check has been processed.
isReminded	Boolean	True means this is for regular payment reminders.
isReturned	Boolean	True means the check has been returned.
isSettled	Boolean	True means the credit card has been settled.
isSystemFailure	Boolean	True means there has been a system error. For example, a network failure.
reminder	IPaymentReminder	The IPaymentReminder object being reminded, valid only when isReminded is true.

Enrollment Notification Template

The enrollment notification template notifies customers about active and bad-active payment accounts and NOC returns. Enrollment reminder messages are generated based on enrollNotify.txt.

This template is used for ACH. The text between %<IF isACH>% and the corresponding %</IF>% is for ACH. If there are no payment gateways for ACH, you can remove that topic from the template file.

Each Oracle Self-Service E-Billing Payment account will be sent an individual email. Oracle Self-Service E-Billing Payment supports multiple payment accounts. If a customer has multiple payment accounts, there could be more than one email message sent for each customer.

Table 72 list the variables available for use in the Enrollment Notification email template. The variables described in Table 72 apply to all cases.

Table 72. Enrollment Notification Template Variables

Variable	Type	Description
checkAccount	ICheckAccount	The current check account being notified.
template	Template	The Payment Template Engine, which is used to declare new variables for the template.
config	IPaymentConfig	Payment setting information, as configured in the Command Center.

The variables described in Table 73 apply to ACH.

Table 73. ACH Variables

ACH Variable	Type	Description
isACH	Boolean	True indicates this is an ACH notification.
success	Boolean	Success means this account has been activated successfully.
errCode	String	ACH return code, if the transaction failed.

The variables described in Table 74 apply to ACH NOC returns.

Table 74. ACH NOC Return Variables

ACH NOC Variable	Type	Description
isNOC	Boolean	True indicates this is an NOC return.
isC01, isC02, isC03, isC05, isC06, isC07	Boolean	True indicates the returned NOC codes.
isAutoUpdate	Boolean	Returns the state of the com.edocs.payment.cassette.ach.autoUpdatNOC flag, which is configured on the Payment Settings page from the Command Center.
newPaymentAccount	String	New payment account number.
oldPaymentAccount	String	Old payment account number.
newRouting	String	New payment routing number.
oldRouting	String	Old payment routing number.
newPaymentType	String	New payment account type.
oldPaymentType	String	Old payment account type.

Recurring Payment Schedule Notification Template

When recurring payment schedules a payment, email notification messages are generated from the template file recurringNotify.txt.

Table 75 describes the recurring notification template variables.

Table 75. Recurring Notification Template Variables

Variable Name	Type	Description
recurringPayment	IRecurringPayment	Contains recurring payment information and current bill information paid by this recurring payment, when applicable. Bill information is null if the amount and pay date are both fixed.
isPaymentScheduled	Boolean	True if a payment has been scheduled.
isCheck	Boolean	True if the payment scheduled is a check.
isCCard	Boolean	True if the payment scheduled is a credit card.
payment	IPaymentTransaction	ICheck if isCheck is true or ICreditCard if isCCard is true. This is the payment being scheduled.
isPaymentNotScheduled	Boolean	True if the payment is not scheduled for some reason. Usually this is because a payment job plug-in rejected the payment based on a customer business rule.
isLessPayment	Boolean	True if the amount due is less than a certain amount, but the amount due is more than that. Notify the customer to pay manually.
isAlreadyPaid	Boolean	True when Oracle Self-Service E-Billing finds a DuplicateBillIdException during the insertion of a payment into database.
isLastRecurringPayment	Boolean	True if this is the last payment.
isRecurringPaymentCancelled	Boolean	True if the recurring payment is cancelled. For example, if the payment account is cancelled. See the job configuration for details.

Payment Notification Template

This template controls the format of email that are sent to the administrator by each job. The template file is notifyPaymentTask.txt.

pmtCreditCardExpNotify Variables

Table 76 describes the payment notification template variables related to pmtCreditCardExpNotifiy.

Table 76. pmtCreditCardExpNotifiy Variables

Variable	Value type	Description
CreditCardExpNotifyTask	String	Identifies the credit card expiration notification task.
isDone	Boolean (true or false)	Identifies the job had done.
jobName	String	Identifies the job name.
ccexpNotifyCount	int	Total number of notifications to be made.
ccexpNotifySuccessCount	int	Successful number of accounts.
ccexpNotifyFailureCount	int	Failed number of accounts.
goodCCAccountCount	int	Number of good credit card accounts (due to decryption).
badCCAccountCount	int	Number of bad credit card accounts (due to decryption).

pmtRecurringPayment Variables

Table 77 describes the recurring notification template variables for the synchronization task.

Table 77. Synchronization Task Variables

Recurring Synchronization Variable	Type	Description
skipSynchronization	Boolean (true or false)	True enables the skip synchronization option.
recurringPmtSyncTask	Boolean (true or false)	True identifies this as the recurring payment task.
isDone	Boolean (true or false)	True indicates that the job is done.
jobName	String	The job name.
syncCount	int	Total number of accounts to be synchronized.
syncSuccessCount	int	Successful number of synchronized accounts.
syncFailureCount	int	Number of failed of synchronized accounts.

Table 78 describes the recurring notification template variables for the scheduler task.

Table 78. scheduler Task Variables

Recurring Scheduler Variable	Type	Description
recurringPmtSchedulerTask	String	Identifies the scheduler task.
isDone	Boolean (true or false)	To identify the job had done.
jobName	String	To identify the job name.
scheduleCount	Int	Total number of accounts to be scheduled.
scheduleSuccessCount	Int	Successful number of scheduled accounts.
scheduleFailureCount	Int	Failed number of scheduled accounts.
CancelCount	Int	Cancelled number of scheduled accounts.
isDecryptFailed	Boolean value (true or false)	To identify whether there were any decryption failures.

pmtPaymentReminder Variables

Table 79 describes the pmtPaymentReminder variables.

Table 79. pmtPaymentReminder Task Variables

Reminder Variable	Type	Description
paymentReminderTask	String	Identifies the payment reminder task
isDone	Boolean (true or false)	Identifies the job is done
jobName	String	Identifies the job name
goodCheckPaymentsCount	Int	Number of successful check accounts
badCheckPaymentsCount	Int	Number of failed check accounts
goodCCPaymentsCount	Int	Number of successful credit card accounts
badCCPaymentsCount	Int	Number of failed credit card accounts

pmtCreditCardExpNotify Variables

Table 80 describes the pmtCreditCardExpNotify variables.

Table 80. pmtCreditCardExpNotify Variables

CCExpNotify Variable	Type	Description
CreditCardExpNotifyTask	String	Identifies the credit card expiration notification task
isDone	Boolean (true or false)	Identifies the job is done
jobName	String	Identifies the job name
ccexpNotifyCoun	int	Total number of notifications to be made
ccexpNotifySuccessCount	int	Number of successful accounts
ccexpNotifyFailureCount	int	Number of failed accounts
goodCCAccountCount	int	Number of good credit card accounts (due to successful decryption)
badCCAccountCount	int	Number of bad credit card accounts (due to unsuccessful decryption)

pmtCheckSubmit Variables

Table 81 describes the pmtCheckSubmit variables.

Table 81. pmtCheckSubmit Variables

Check Submit Variable	Type	Description
CheckSubmitTask	Boolean value (true or false)	Identifies the check submit task.
isDone	Boolean (true or false)	Identifies the job done.
jobName	String	Identifies the job name.
isHoliday	Boolean value (true or false)	Identifies a holiday.
dateUtil	DateUtil object	Format of the expiration date.
isDecryptFailed	Boolean value (true or false)	Identifies whether there were any decryption failures.

pmtSubmitEnroll

Table 82 describes the pmtSubmitEnroll variables.

Table 82. pmtSubmitEnroll Variables

Submit Enroll Variable	Type	Description
SubmitEnrollTask	String	Identifies the submit enroll task.
isDone	Boolean (true or false)	Identifies the job had done.
jobName	String	Identifies the job name.
sHoliday	Boolean value (true or false)	Identifies a holiday.
isDecryptFailed	Boolean value (true or false)	Identifies whether there were any decryption failures.

Credit Card Expiration Notification Template

When a credit card is about to expire, email notification messages are generated from the template file CCExpNotify.txt.

Table 83 describes the credit card expiration notification template variables.

Table 83. Credit Card Expiration Notification Template Variables

Variable	Value Type	Description
accExpired	Boolean value (true or false)	Identify whether the account is expired or not
account	ICreditCardAccount object	Object of ICreditCardAccount that has the information about the account

Customizing ACH Templates

The ACH records of interest are in File Header, Batch Header, Entry Detail for PPD, Addenda and return for PPD, Batch Trailer and File Trailer. ACH fields can be mandatory, required, or optional. The contents of mandatory fields are fixed and must not be customized. Required fields are usually defined by the receiving bank, and can be customized for different banks. Optional fields can be customized, also.

By default, secCode is set to WEB to be compliant with the ACH 2001 format. However, you can change the SEC code based on the requirements of a biller's bank by editing the batchHeader_template.xml file.

Table 84 describes some ACH fields. These fields can be customized upon a biller's request. The pmtCheckSubmit jobs running date is referred to as Today.

Table 84. ACH Fields

Field Name	Location	Description
Company Descriptive Date	8th field in batch header, optional	Default set to Today; the date pmtCheckSubmit is running.
Effective Entry Date	9th field in batch, required	The date when checks in the batches are to be cleared. This is a suggested date from ACH, but the date that checks are cleared can vary. All checks with the same pay date will be put into one batch. The effective entry date might not always be the pay date. The default setting for effective entry date is: If the pay date is tomorrow or earlier, then it is the earliest business date after today. If the pay date is after tomorrow, then it is the earliest business date after the pay date (including the pay date).
Individual ID	7th field in PPD entry detail, optional or required	By default set to the customer's account with the biller. Because this field is 15 bytes, the length of customer's account must not exceed 15 bytes. If the customer account is longer than 15 bytes, either the field will not be populated, or you must truncate this field using Java code or the Java classes provided by Oracle Self-Service E-Billing.
Individual Name	8th field in PPD entry detail. Required	By default set to the check's payment ID. Payment ID is the primary key on the check_payments table. It can be used to map a returned check back to the one in Oracle Self-Service E-Billing Payment database.

The templates for ACH are XML files, which describe the format of each ACH record, such as the start position, length, and so on. There are two sets of templates: one to generate ACH files, and another to parse ACH return files.

The first set of templates is used to generate the following ACH files:

- fileHeader_template.xml
- batchHeader_template.xml
- entryDetail_template.xml
- batchTrailer_template.xml
- Trailer_template.xml

When an ACH file is generated, check information is pulled from the database and then populated into the content of the XML files by replacing the template variables. The resulting XML file is transferred into an ACH file according to the format specified by the XML tags. The generic format of an XML tag is:

```
<amount pos="30" len="10" fmt="N" fract="2">%
```

where:

- *amount* is the name of the tag
- pos="*30*" is the start position
- len="*10*" is the length of the field
- fmt="*N*" is the format of the field
- fract="*2*" is the number of digits after the decimal point if the format (fmt) is N (numerical)

Table 85 through Table 89 list the template variables that are predefined in the Oracle Self-Service E-Billing Payment Template Engine. These variables are used to populate the content of the templates.

Table 85 describes the template variables that all templates use.

Table 85. Global Template Variables

Global Variable Name	Type	Description
template	com.edocs.util.template.Template	The template engine.
stringUtil	com.edocs.payment.util.StringUtil	Makes calling the static methods of StringUtil easier. Instead of using: %com.edocs.payment.util.StringUtil.concat(“a”,”b”,”c”)% use: %stringUtil.concat(“a”, “b”, “c”)%
decimalUtil	com.edocs.payment.util.DecimalUtil	Provides decimal number manipulations.
dateUtil	com.edocs.payment.util.DateUtil	Provides date manipulation methods Also a calendar, which includes all U.S. holidays.
batch	com.edocs.payment.IPaymentBatch	The payment summary report, which you can view through the Command Center.
config	com.edocs.payment.config.IPaymentConfig	Payment setting information.
attributeName	com.edocs.payment.config.AttributeName	Payment setting parameter names, Use it with the variable config to get payment setting information.

Table 86 describes the template variables that File Header uses.

Table 86. File Header Variables

Variable Name	Type	Description
fileCreateDate	java.util.Date	Creation date of the ACH file.
fileCreateTime	java.util.Date	Creation time of the ACH file.
fileIdModifier	java.lang.String	ACH file modifier, A to Z and 0 to 9.

Table 87 describes the template variables that Batch Header uses.

Table 87. Batch Header Variables

Variable Name	Type	Description
curPayDate	java.util.Date	The pay date of checks in the batch. All the checks in the same batch have the same pay date.
companyDescData	String	From Payment Settings.
companyDescDate	Date	Defaults to Today. To use another date, you must call a static Java method.
batchNumber	int	Starts from 1; identifies the batches in the ACH.
batchEffectiveEntryDate	Date	Identifies the batches in the ACH.

Table 88 describes the template variables that Entry Detail uses.

Table 88. Entry Detail Variables

Variable Name	Type	Description
check	com.edocs. payment.ICheck	All check payment information, including the trace number.
addenda Record Indicator	int	Indicates whether there is addenda record for entry detail. 0=No; 1=Yes.

Table 89 describes the template variables that Batch Trailer uses.

Table 89. Batch Trailer Variables

Variable Name	Type	Description
batchEntryHash	String	See the ACH documentation.
batchEntryAddendaCount	int	Number of entries in the batch.
batchDebitAmount	String	Total debit amount in the batch.
batchCreditAmount	String	Always zero.

Table 89. Batch Trailer Variables

Variable Name	Type	Description
blockCount	int	See the ACH documentation.
totalEntryHash	String	See the ACH documentation.
totalEntryAddendaCount	int	Total number of entries in the file.
totalDebitAmount	String	Total debit amount in the file.

Matching a Check in the ACH Return to the Database

Return files are parsed by the return templates:

- fileHeader_return_template.xml
- batchHeader_return_template.xml
- entryDetail_return_template.xml
- addenda_return_template.xml
- batchTrailer_return_template.xml
- fileTrailer_return_template.xml

The format of these files is similar to the format of the submit template. For example:

```
<individualName pos="55" len="22" fmt="AN" target="%check.setPaymentId(?)%"></
individualName>
```

This code retrieves the part of the text from positions 55 to 77, puts it into a variable called ? and then calls check.setPaymentId() to set payment_id for the check. The template executes the template statement specified by XML tag “target” only.

When a check is returned from the ACH network, Oracle Self-Service E-Billing Payment matches it to that check in the database and marks it as returned. ACH modifies several fields in the return file. Oracle Self-Service E-Billing Payment populates one or more unchanged fields with identification information to help in matching them with a check in the database. Consult the ACH documentation for information about which fields are not changed.

The return template retrieves the error return code from the addenda record and then tries to reconstruct the payment ID or gateway payment ID to match a check in the database. If Oracle Self-Service E-Billing Payment cannot populate the payment ID into the ACH file, it uses the gateway payment ID, which is a concatenation of a few check payment fields that can identify a check.

By default, Oracle Self-Service E-Billing Payment populates the payment_id of the check into the individual name field to create the ACH file. The following line in the entryDetail_template.xml file populates the payment ID into an individual name:

```
<individualName pos="55" len="22" fmt="AN">%check.getPaymentId()%</individualName>
```

The following line in the entryDetail_return_template.xml file extracts the payment ID:

```
< individualName pos="55" len="22" fmt="AN" target="%check.setPaymentId(?)%"></
individualName >
```

The following line in the addenda_return_template.xml file extracts the return error code:

```
<returnReasonCode pos="4" len="3" target="%check.setTxnErrMsg(?)%"></
returnReasonCode>
```

Payment then changes the status of the check to returned and updates this check in the database using its payment_id.

If the individual name is required for another task, for example, the check account name (which is the first 22 bytes), then follow these steps to use gateway payment ID.

To use the gateway payment ID

1. Modify the entryDetail_template.xml file to populate individual name with account name. Change:

    ```
    <individualName pos="55" len="22" fmt="AN">%check.getPaymentId()%</
    individualName>
    ```

    to:

    ```
    <individualName pos="55" len="22"
    fmt="AN">%stringUtil.substring(check.getAccountName(), 0, 22)%</individualName>
    ```

2. Modify the entryDetail_return_template.xml file so that payment ID will not be set for a returned check. Change:

    ```
    <individualName pos="55" len="22" fmt="AN" target='%check.setPaymenId(?)%'></
    individualName>
    ```

    to:

    ```
    <individualName pos="55" len="22" fmt="AN"></individualName>
    ```

3. Because payment ID cannot be used to match checks, use the gateway payment ID instead. Gateway payment ID is the ID generated by the template that submitted the ACH file to ACH. This template generates a unique ID based on the information submitted to ACH. This ID must contain information that will not be changed by ACH in the return file. The Oracle Self-Service E-Billing engine uses the gateway payment ID to find a match in the database.

In very rare circumstances, more than one match might be found. In that case, the match with the latest creation time is used. The following example discusses several ways to generate the gateway payment ID. Oracle Self-Service E-Billing Payment generates a trace number and puts that into the entry detail record. By default, the trace number starts at 0000000 and increases by one for each check until it reaches 9999999. After this point, the numbering restarts at 0000000. It is possible to get a duplicate trace number (after 10 million checks). However, because the Oracle Self-Service E-Billing Payment engine always chooses the payment with the latest date, the correct check will be matched. You can use both the trace number and individual ID (customer account number) to identify a payment and use them for the gateway payment ID.

Example 1: Unchanged ACH Trace Number

In the following example, it is assumed that the ACH or Bank will return both the original trace number and individual ID to Oracle Self-Service E-Billing:

1 At the beginning of the entryDetail_template.xml file, see the following code:

```
<ACH_6>
%<*>%
%check.setGatewayPaymentId(com.edocs.payment.util.StringUtil.concat(check.getPa
yerAcctNumber(), "_", check.getTxnNumber()))%

%</*>%
```

This statement is commented out in the template, using %<*>% and %</*>%. Removing the comment tags enables the statement.

The trace number is stored as txnNumber in the check object. This statement concatenates the customer account number, a "_", and trace number as the gateway payment ID. The `setGatewayPaymentId` method returns void, so nothing will print out. (If it did return a value, then that would print, which would ruin the format of the XML file.) After running pmtCheckSubmit, check the gateway payment ID in the check_payments table, which is the concatenation of the individual ID and the trace number that are written into the entry detail record.

2 Next, Payment retrieves the original trace number from the return file, and sets it as the gateway payment ID. In the addenda_return_template.xm, find this code:

```
<traceNumber pos="80" len="15" fmt="N"
target1='%check.setGatewayPaymentId(txnNumber)%'
target2='%check.setGatewayPaymentId(stringUtil.concat(payerAcctNumber, "_",
txnNumber))%'></traceNumber>
```

Rename target2= to target, which will reconstruct the gateway payment ID based on the returned customer account number and trace number. Template variable payerAcctNumber has been set in the entryDetail_return_template.xml file and txnNumber has been set before this line in the addenda_return_template.xml file by calling template.putToContext.

3 Now you are all set. Test this setting using a real return file and verify that the check's status has been updated to –4 in the check_payments table.

Example 2: Modified ACH Trace Number

If the individual ID is not returned as it was set, you can try to use other information, such as individual name combined with trace number. If only the trace number can be used for gateway payment ID, use that as follows.

To use only the trace number for gateway payment ID

1 At the beginning of the entryDetail_template.xml file, see the following code:

```
      %<*/>%

%check.setGatewayPaymentId(check.getTxnNumber())%

%</*>%
```

Remove the comment tags to enable the statement.

2 In the addenda_return_template.xml file, rename target1 to target to enable using trace number as gateway payment ID:

```
<traceNumber pos="80" len="15" fmt="N"
target1='%check.setGatewayPaymentId(txnNumber)%'
target2='%check.setGatewayPaymentId(stringUtil.concat(payerAcctNumber, "_",
txnNumber))%'></traceNumber>
```

Generating Accounts Receivables (A/R Files)

It is often necessary to synchronize Payment with a biller's A/R software. Payment sends A/R files periodically to a biller's A/R software, which includes the payments being made through Payment. The format of the file varies among billers. To support this function, Payment has the pmtARIntegrator job, which uses a template and XML/XSLT to generate output in a variety of file formats.

The pmtARIntegrator job queries the Payment database to get proper payments, and then writes the payments into a flat file or an XML file using the Payment Template Engine. The XML file can be further transformed into other format by using XSLT.

The default implementation of the pmtARIntegrator job performs the following steps:

1 Queries the Payment database to get a list of check and/or credit card payments. The query is defined in arQuery.xml file, which finds all the check and credit card payments where the payee_id matches the current job DDN, the status is 8 (paid) and flexible_field_3 is N.

2 Invokes the process() method of the default implementation of com.edocs.payment.tasks.ar.IARPaymentIntegrator, which is com.edocs.payment.tasks.ar.SampleARPaymentIntegrator. In this method, ARPaymentIntegrator writes the payments into a flat file or XML file using the Payment Template Engine. There are two templates provided by Payment:

   - **arFlat_template.txt**. Generates a flat A/R file
   - **arXML_template.txt**. Generates an XML file

   The output file name is: ar_yyyyMMddHHmmssSSS.extension, where extension matches the extension of the template file.

3 Inside the process() method, if the output is an XML file, SampleARPaymentIntegrator can optionally apply an XSLT file against the output file to transform it into another format. The transformed file name is: ar_trans_yyyyMMddHHmmssSSS.extention, where extension is defined by the pmtARIntegrator job configuration.

4 Inside the process() method, SampleARPaymentIntegrator updates flexible_field_3 of both check and credit card payments to Y, and writes that to database. This ensures these payments will not be processed again by the next run of pmtARIntegrator.

Customizing the arQuery.xml File

The SQL queries used by the pmtARIntegrator job are defined in an XML file, arQuery.xml, which is provided by the default Payment installation. The arQuery.xml file is based on Oracle XMLQuery technology.

CAUTION: XMLQuery supports paging, but this feature must not be used for this job.

Most of the A/R file creation is done by an implementation class of the interface com.edocs.payment.tasks.ar.IARPaymentIntegrator. This adaptor interface provides maximum flexibility for customizing this job. The default implementation is com.edocs.payment.tasks.ar.SampleARPaymentIntegrator.

Before the query is executed in the database, the job invokes the getMap() method of `IARPaymentIntegrator`, which gets a list of objects that are used to replace the variables “?” defined in the SQL query of the arQuery.xml file. For more information about `IARPaymentIntegrator`, see Accessing Oracle Self-Service E-Billing Javadoc on page 31 for details on accessing *Oracle Self-Service E-Billing* Javadoc.

The default IARPaymentIntegrator implementation, SampleARPaymentIntegrator, uses this arQuery.xml file for database query:

```
<?xml version="1.0" encoding="UTF-8"?>
<query-spec>
  <data_source_type>SQL</data_source_type>

<query name="checkQuery">
    <sql-stmt><![CDATA[select * from check_payments where payee_id = ? and statu
s = 8 ]]></sql-stmt>
    <param name="payee_id" type="java.lang.Integer" position="1"/>
    <!--param name="last_modify_time" type="java.sql.Timestamp" position="2" /-->
  </query>

  <query name="creditCardQuery">
  <sql-stmt><![CDATA[select * from creditcard_payments where payee_id = ? and st
atus  = 8 and flexible_field_3 = 'N']]></sql-stmt>
  <param name="payee_id" type="java.lang.Integer" position="1"/>
  </query>

</query-spec>
```

Two queries are defined:

- **checkQuery**. Queries check payments

- **creditCardQuery.** Queries credit card payments

  Both these queries get all the successful payments (status=8) of the current payee (biller or DDN of current job) from the relevant Oracle Self-Service E-Billing Payment tables. They both use flexible_field_3 as a flag to prevent a payment from being sent to the A/R job twice. This flag is initially set to N when the payment is created. After the A/R job runs, the SampleARPaymentIntegrator changes the flag to Y.

  When using flexible_field_3 as an A/R flag, you can create an index for it to increase performance. Oracle Self-Service E-Billing Payment provides a script just for that purpose: create_ar_index.sql. This script is not run when the Oracle Self-Service E-Billing Payment database is created, run it manually.

  Each of the queries in the arQuery.xml file has an SQL variable ('?') that must be resolved before the query can be sent to the database. The A/R job calls the getMap() method of IARPaymentIntegrator to get a Map of query variables, and uses their values to replace the '?'s in the query. The names of the Map elements match those defined in the param tags of the query tags.

  For example, the default arQuery.xml file has the param tag:

  ```
  <param name="payee_id" type="java.lang.Integer" position="1"/>
  ```

  To support this, define a Map element whose name is payee_id and whose value (which must be an Integer, and contains the DDN reference number) replaces the question mark (?) with payee_id in the query:

  ```
  select * from check_payments where payee_id = ? and status = 8 and
  flexible_field_3 = 'N'
  ```

  The query result set will be transferred to a list of checks (ICheck objects) for checkQuery, and credit cards (ICreditCard objects) for creditCardQuery, and then pass that list to the process() method of IARPaymentIntegrator.

  CAUTION: The XML Query object supports paging, but do not use this feature for A/R query.

  You can modify this file to use different queries.

Querying Case Study

The new requirement for this example is to retrieve all payments whose status is returned or paid between 5:00PM today (the job run date) and 5:00PM yesterday (yesterday's job run date).

To try a query case study

1 Change the arQuery.xml file for checkQuery:

```
<query name="checkQuery">

<sql-stmt><![CDATA[select * from check_payments where payee_id=? and status in
(8,-4) and last_modify_time >= ? and last_modify_time < ? ]] </sql-stmt>

<param name="payee_id" type="java.lang.Integer" position="1"/>
```

```
<param name="min_last_modify_time" type="java.sql.Timestamp" position="2"/>

<param name="max_last_modify_time" type="java.sql.Timestamp" position="3"/>

</query>
```

TIP: Use java.sql.Timestamp instead of java.util.Date.

2 Change the arQuery.xml file for creditCardQuery. Because you are adding more question marks to the query, override the getMap() method of the default ARPaymentIntegrator:

```
package com.edocs.ps.ar;
import java.util.*;
import com.edocs.payment.util.DateUtil;

public class MyARIntegrator extends ARPaymentIntegrator

{

   /**Override this method to populate the SQL variables in arQuery.xml

    */

public Map getMap(ARPaymentIntegratorParams payIntegratorParam,
                                 String objectFlag) throws Exception
{
        //call super class because

need to get the payee_id value
        Map map = super.getMap(payIntegratorParam, objectFlag);
        //no need to check objectFlag because we actually populate the
        //same values for both checkQuery and creditCardQuery
        Date today = new Date();

        today = DateUtil.dayStart(today);//set to 00:00:00AM
        Date today5 = DateUtil.addHours(today, 17); //set to 05:00:00PM

        Date yesterday5 = DateUtil.addHours(today, -7) ;//set to 05:00:00PM of
yesterday
        map.put(“min_last_modify_time”, DateUtil.toSqlTimestamp(yesterday5));

    map.put(“max_last_modify_time”, DateUtil.toSqlTimestamp(today5));
}
```

3 To make the cutoff time configurable instead of fixed at 5:00PM, use the flexible configuration fields of the A/R job, which are passed in as part of ARPaymentIntegratorParams. For more information about ARPaymentIntegratorParams, see Accessing Oracle Self-Service E-Billing Javadoc on page 31 to access the Javadoc.

4 Compile your class using the Payment_client.jar and Payment_common.jar that comes with Oracle Self-Service E-Billing, package the compiled class into the payment EAR files, and redeploy the EAR files.

5 Log into the Command Center and change the configuration of the A/R job to use the new implementation of the IARPaymentIntegrator, com.edocs.ps.ar.MyARIntegrator.

Customizing the arFlat_template.txt File

Payments returned by the arQuery.xml file are written to an A/R file using an Oracle Self-Service E-Billing Payment template file. Two templates come with Oracle Self-Service E-Billing:

- **arFlat_template.txt.** Generates a flat A/R file
- **arXML_template.xml.** Generates an XML A/R file

The arFlat_template.txt file generates a sample flat A/R file. If this file includes most of your required data, but the format is not what you want, you can edit the template file to generate your own format. For more information about using the Template class, see Accessing Oracle Self-Service E-Billing Javadoc on page 31.

The A/R job using arFlat_template.txt does the following:

- Loops through the list of check and credit card payments to print out their details.
- Calculates the totals for check debits, check credits, credit card debits and credit card credits (reversals).

Customizing the arXML_template.xml File

The arXML_template.xml file generates the same information as arFlat_template.txt, but in XML format. After creating the XML file, you can use XSLT to transform it into another XML file or into a flat file. The default arTransform.xsl transforms the original XML file into the same format as the one generated by arFlat_template.txt. Using XSLT is the recommended way to do the customization.

The A/R job using the arXML_template.xml file does the following:

- Loops through the list of check and credit card payments to print out their details.
- Calculates the totals for check debits, check credits, credit card debits and credit card credits (reversals).

To generate different file formats, change arTransform.xsl. Or, customize the arXML_template.xml file directly.

Customizing the arXML_template.xml File and Using XSLT to Generate an XML Flat AR File

The arXML_template.xml file generates the same information as arFlat_template.txt, but in XML format. After generating the XML file, you can use XSLT to transfer it into another XML file or into a flat file. The default arTransform.xsl transforms the XML file into the same format as the one generated by arFlat_template.txt. If you are familiar with XSLT, this is the recommended way to do the customization.

This template does the following:

- Loops through the list of check and credit card payments to print out their details.
- Calculates the totals for check debits, check credits, credit card debits and credit card credits (reversals).

To generate different file formats, change arTransform.xsl. If required, you can also customize the arXML_template.xml file.

To rename the generated files

- To rename the files generated by these utilities you must write an implementation of IARPaymentIntegrator. The following code demonstrates how to rename the XSLT output file to another name:

```
import java.io.*;
public class MyARIntegrator extends ARPaymentIntegrator
{
protected void getTransformedARFileName(ARPaymentIntegratorParams
               payIntegratorParam, ) throws Exception
{
return ”newARName.txt”;
}
}
```

Reimplement IARPaymentIntegrator

You might want to reimplement the default SampleARPaymentIntegrator if you want to add any of the following features.

To reimplement the default SampleARPaymentIntegrator

1. Rename the default AR files.
2. Change the SQL query to add more “?” variables and to set values for those variables in the IARPaymentIntegrator implementation.
3. Add any additional steps, such as putting more objects into Template context before it is parsed.
4. Change the result of the template parsing. For example, because of limitations of Template engine, sometimes unwanted empty new lines are added. Remove those lines.
5. Modify the check or credit card objects before they are updated in the database. By default, only flexible_field_3 is updated from N to Y. Another alternative is to update the check or credit card object in the template, and all your updates will be updated in the database.

   To add any of the preceding features, you must extend from SampleARPaymentIntegrator and configure the pmtARIntegrator job to use your implementation.

   You can overwrite following methods for your customization:

   - **getARFileName().** Overwrite to change the name of the AR flat file generated from arFlat_template.txt.
   - **getMap()**. Overwrite

Select Only Check or Credit Card Payments

A biller might support only one of check or credit card payments. In this case, you must configure the pmtARIntegrator job to leave the Credit card query name in XML query file field blank. To optionally remove any reference to the unavailable payment type, customize the template files (arFlat_template.txt or arXML_template.xml).

Compiling and Packaging a Custom IARIntegrator

If you reimplement IARIntegrator or you have some custom Java classes to call from the AR template, you must recompile and package your changes.

In most cases, you put your custom code into Payment_custom.jar. Unfortunately, the IARIntegrator and its related classes are packaged as part of ejb-Payment-ar.jar, not Payment_custom.jar, so a different procedure is required.

To compile, put ejb-Payment-ar.jar along with Payment_common.jar, Payment_custom.jar and Payment_client.jar in your class path to reimplement IARIntegrator.

To package, drop all your AR custom classes into the ejb-Payment-ar.jar.

A/R Filenames

The generated A/R files have default names of ar_yyyyMMddHHmmssSSS.template_file_ext, where the template_file_ext is the file extension of the template file. The XSLT transformed file has default name of ar_trans_yyyyMMddHHmmssSSS.extension, where extension is defined by the pmtARIntegrator job configuration. You can rename these files to a more meaningful name.

To rename the files, write an implementation of IARPaymentIntegrator. The following code demonstrates how to rename the XSLT output file to another name:

```
package com.edocs.ps.ar;

import com.edocs.payment.tasks.ar.*;

public class MyARIntegrator extends ARPaymentIntegrator

{

/**Override this method to give a new name*/

protected void getTransformedARFileName(ARPaymentIntegratorParams
               payIntegratorParam, ) throws Exception
{

return ”newARName.txt”;

}

}
```

Single Payment Type

A biller might have only ACH and not credit card payments, or conversely. In this case, you can customize the template files (arFlat_template.txt or arXML_template.xml) to remove any references to the unavailable payment type.

Or, when configuring the pmtARIntegrator job enter an empty value for the Check query name in XML query file or Credit card query name in XML query file parameter.

Customizing the Payment Amount Format

You can customize the payment amount format for the following features:

- Credit card registration fee
- Minimum and maximum payment amounts
- Currency pattern
- Two decimal pattern
- Payment amount threshold
- Whether to allow payments greater than the amount due
- Whether to display a warning message if the payment amount is less than the amount due

To configure the payment amount format

1 Edit the paymentService.xma.xml file, found in the *EDX_HOME*\xma\config\modules\services directory.

2 Modify the parameters in the paymentConfigurationBean section as needed:

```
<bean id="paymentConfigurationBean"

    class ="com.edocs.common.services.payment.config.PaymentConfigurationBean"
scope="singleton">

    <property name="DDNName">

       <value>ReportApp</value>

    </property>

    <property name="creditCardRegisterFee">

       <value>1.0</value>

    </property>

    <property name="paymentAmountThreshold">

       <value>NoLimit</value>

    </property>

    <property name="paymentAmountGreaterthanAmountDue">

       <value>Yes</value>

    </property>

    <property name="minimumPayAmount">

       <value>1.0</value>
    </property>

    <property name="currencyPattern">

       <value>##,##0.00</value>

    </property>

    <property name="twoDecimalPattern">

       <value>[0-9]*[G]*[0-9]*[D]?[0-9]{0,2}</value>

    </property>

    <!-- For value Yes , it displays the warning message if payment amount is less
than the amount due. for value No, not display the warning message -->

    <property name="paymentAmountLessthanAmountDue">

       <value>Yes</value>

    </property>

  </bean>
```

Configuring International Bank Routing

Oracle Self-Service E-Billing supports ACH gateways with US routing number standards as the default. To provide your customers with the option to make payments using international bank accounts, configure the check gateway specifications for the particular country.

To configure international bank routing

1. Implement the following custom classes:
   - **Check cassette class.** For the country's specific check gateway standard, including properties for communicating with the gateway, replace the default implementation in \com\edocs\payment\cassette\ach\ach_CheckCassette.class.
   - **ACH check class.** For extending the standard Check class, including properties coming from different check gateway standards. This class is used to generate files sent to the check gateway for authentication or check transaction purposes. Replace the default implementation in \com\edocs\payment\cassette\ach\AchCheck.class.

- **Returned check class.** For processing the returned check file for the gateway. Replace the default implementation in \com\edocs\payment\cassette\ach\AchReturnedCheck.class.

Replace the default files with your custom class files in the payment_custom.jar file in the following directories. For Windows, change the slashes and root as necessary.

Application Server	Jar Files and Directories

Oracle WebLogic	`EDX_HOME\J2EEApps\commandcenter\weblogic\command-center-weblogic-10-6.0.1.ear\lib\payment_custom.jar`
	`EDX_HOME\J2EEApps\ebilling\weblogic \ebilling-weblogic-10-6.0.1.ear\lib\payment_custom.jar`
	`EDX_HOME\payment\lib\payment_custom.jar`

IBM WebSphere	`EDX_HOME\J2EEApps\commandcenter\websphere\command-center-websphere-6-6.0.1.ear \lib\payment_custom.jar`
	`EDX_HOME\J2EEApps\ebilling\websphere\lib\payment_custom.jar`
	`EDX_HOME\payment\lib\payment_custom.jar`

2 Implement a custom RoutingNumber.class. This class is used to validate routing numbers when creating a new check account. This file is located in the following directory. For Windows, change the slashes and root as necessary.

- **Oracle WebLogic.** `EDX_HOME/J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/lib/ebilling-web-1.0-SNAPSHOT.jar/com/edocs/application/ebilling/payment/util/RoutingNumber.class`
- **IBM WebSphere.** `EDX_HOME/J2EEApps/ebilling/websphere/ebilling-websphere-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/lib/ebilling-web-1.0-SNAPSHOT.jar/com/edocs/application/ebilling/payment/util/RoutingNumber.class`

3 In the validation-payment.xml file, edit the validation rules for the routingNumber field property. This file is located in the following directory:

- **Oracle WebLogic.** `EDX_HOME/J2EEApps/ebilling/weblogic/ebilling-weblogic-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/validation-payment.xml`
- **IBM WebSphere.** `EDX_HOME/J2EEApps/ebilling/websphere/ebilling-websphere-10-6.0.4.ear/ebilling-web-1.0-SNAPSHOT.war/WEB-INF/validation-payment.xml`

4 Customize your templates files for generating the files sent to the gateway or for parsing the returned file from the gateway.

Packaging Oracle Self-Service E-Billing Payment Custom Code

You can package your custom code, both plug-in code and custom A/R jobs and templates, by adding it to Payment_custom.jar. The Oracle Self-Service E-Billing Payment EAR files will access this JAR, and find the custom code. The Oracle Self-Service E-Billing Payment EAR files merge into the Command Center EAR file as part of installation, so your custom code will also be seen by the Command Center.

To make this JAR file accessible by all of the Oracle Self-Service E-Billing Payment EJB, JAR and WAR files, place it in the classpath of the MANIFEST file of each JAR and WAR file. For details of how the MANIFEST file works, refer to the J2EE or EJB specifications or the SDK: Customizing and Deploying Applications document that comes with the Command Center SDK. When the EJB JAR or WAR files are loaded, this JAR will be loaded and can be accessed by the EJB JAR files or WAR files.

CAUTION: Never put your custom EJB code into Payment_custom.jar; put your EJB code in your own JAR files.

To write a new plug-in for IAchCheckSubmitPlugIn

1. Write and then compile your implementation class. You might want to use Payment_common.jar and Payment_client.jar from Oracle Self-Service E-Billing Payment as part of your class path.
2. Create a JAR file called Payment_custom.jar, or use the Payment_custom.jar from any of the Oracle Self-Service E-Billing Payment EAR files. Place your implementation class into that JAR file using the JAR command.
3. Replace all the Payment_custom.jar files under the lib directory of all the deployed Oracle Self-Service E-Billing Payment EAR files with the new Payment_custom.jar, using JAR command.
4. Deploy the new Oracle Self-Service E-Billing Payment EAR files on your application server.
5. Go to Payment Settings in the Command Center, and configure the payment gateways to use the new class by replacing the default one, com.edocs.payment.cassette.ach.AchCheckSubmitPlugIn, with your new plug-in.
6. Run the pmtCheckSubmit job, which will load the new class from Payment_custom.jar, because you added it to the classpath of the MANIFEST file of ejb-Payment-chksubmit.jar.

Debugging Payment

Follow the installation steps carefully to set up payment. After installation and initial configuration, if you still have problems, the following actions describe a few ways to help identify the cause:

- **View the Oracle WebLogic Logs.** From the Oracle WebLogic console, change the level of log messages. By default, only error messages will be printed out to the console. You can change it to print more detailed information.
- **View log files From the Command Center.** If an Oracle Self-Service E-Billing Payment job fails, you can view log files from the Command Center to see the details of the error message.

- **Turn on the Oracle Self-Service E-Billing Payment Debug Flag.** If you have problems with executing payment operations, such as making a check payment or running an Oracle Self-Service E-Billing Payment job, you can turn on the com.edocs.payment.debug flag to see more details.

  Configure your application server to use "-Dcom.edocs.payment.debug=true" as part of the JVM starting option. For example, for Oracle WebLogic on UNIX, change the startWebLogic.sh file to add another option to the java command:

  ```
  java –Dcom.edocs.payment.debug=true …
  ```

About Job Plug-Ins

Table 90 lists the plug-ins available for the payment jobs.

Table 90. Payment Job Plug-Ins

Job	Plug-in Code
pmtPaymentReminder	PaymentReminderPlugIn.java
pmtCreditCardSubmit	PayPalCreditCardSubmitPlugIn.java
pmtCheckSubmit	AchCheckSubmitPlugIn.java AddendaCheckSubmitPlugIn.java (Example implementation included.)
pmtRecurringPayment	RecurringPaymentPlugIn.java SampleRecurringPlugIn.java (Example implementation included.)

About Payment Auditing

Oracle Self-Service E-Billing Payment audits some Oracle Self-Service E-Billing Payment jobs to track a variety of transaction failures. Audits are kept for actions taken through the UI, as well as jobs.

Payment Jobs That Are Audited

The following jobs write to the audit tables:

- **pmtCheckSubmit**. Writes the following audited information:
  - Payments that failed during submission
  - Encryption exceptions
- **pmtPaymentReminder**. Writes payment reminders that were not sent, including:
  - Regular payment reminders that failed to send, for any reason, such as bad email address.
  - Check payment email that failed to send, for any reason, such as encryption error, bad email address.

  - Credit card payment email failed to send, for any reason, such as encryption error or bad email address.
- **pmtCreditCardSubmit**. Writes credit card payments that failed to submit, for example, because of encryption errors, invalid credit card information (such as invalid account) or network errors.
- **pmtIntegrator (AR)**. Writes check and credit card payments that were not written to the AR file, such as because of encryption errors or file write errors.
- **pmtRecurringPayment Job**. Check and credit card payments that failed.
- **pmtCheckSubmit and pmtCreditCardSubmit**.

UI Actions That Are Audited

Lists successful and unsuccessful payments along with a reason code.

The UI actions that trigger an audit entry are:

- Create Recurring Payment
- Update Recurring Payment
- Delete Recurring Payment
- Create Schedule Payment
- Create Instant Payment
- Cancel Future Payment (Credit Card Payment)
- Update Future Payment (Credit Card Payment)
- Cancel Future Payment (Check Payment)
- Update Future Payment (Check Payment)
- Create Payment Reminder
- Update Payment Reminder
- Delete Payment Reminder
- Create Check Account
- Edit Check Account
- Delete Check Account
- Create Credit Card Account
- Edit Credit Card Account
- Delete Credit Card Account

Example UI Audit Flow

The following steps show how a UI audit flow processes:

1 The customer selects the Setup of recurring payment option, populates the information to initially set up recurring payment, and submits it. The following information is recorded as the audit data in the recurring_payments_history table in addition to the columns defined in the recurring _payments table. (This history table contains all the columns defined in the recurring_payments (regular table) table plus the additional following columns).

Column	Value	Description
audit_operation	1001	This constant value for the operation is explained in the recurring_payment_const table.
audit_status	1	Status constant value successful operation. This constant value for the status is explained in the recurring_payment_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is an UI operation, the job ID value is 0 (not a job).
Job_name	NULL	Because this is a UI operation, job name is NULL.
Timestamp	none	The current system time when an audit occurs.

2 The customer selects Recurring Payment option, and then selects Update, and updates the recurring payment information and submits it, the following information is recorded as the audit data in recurring_payments_history table other than the columns defined in the regular recurring _payments table. (This history table contains all the columns defined in the recurring_payments (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1002	This constant value for the operation is explained in the recurring_payment_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the recurring_payment_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

3 The customer selects Recurring Payment option, and then selects Delete, the following information is recorded as the audit data in recurring_payments_history table other than the columns defined in the regular recurring _payments table. (This history table contains all the columns defined in the recurring_payments (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1003	This constant value for the operation is described in the recurring_payment_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is described in the recurring_payment_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

4 The customer selects Create Check account in the User Profile UI, and submits the new check account information, the following audit data is recorded in payment_accounts_history table other than the columns defined in the regular payment_accounts table. (This history table contains all the columns defined in the payment_accounts (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1001	This constant value for the operation is explained in the payment_account_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_account_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

5 The customer selects Update Check account in the User Profile UI, and submits the updated check account information, the following audit data is recorded in payment_accounts_history table other than the columns defined in the regular payment_accounts table. (This history table contains all the columns defined in the payment_accounts (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1002	This constant value for the operation is explained in the payment_account_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_account_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

6 The customer selects Delete Check account in the User Profile UI, and submits the delete request, the following audit data is recorded in payment_accounts_history table other than the columns defined in the regular payment_accounts table. (This history table contains all the columns defined in the payment_accounts (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1003	This constant value for the operation is explained in the payment_account_const table).
audit_status	1	Status constant value for successful operation. (this constant value for the status is explained in the payment_account_const table).
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

7 The customer selects Create Credit Card account in the User Profile UI, and submits the new credit card account information, the following audit data is recorded in payment_accounts_history table other than the columns defined in the regular payment_accounts table. (This history table contains all the columns defined in the payment_accounts (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1001	This constant value for the operation is explained in the payment_account_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_account_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

8 The customer selects Update Credit Card account in the User Profile UI, and submits the updated credit card account information, the following audit data is recorded in payment_accounts_history table other than the columns defined in the regular payment_accounts table. This history table contains all the columns defined in the payment_accounts (regular table) table and additional following columns:

Column	Value	Description
audit_operation	1002	This constant value for the operation is explained in the payment_account_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_account_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

9 The customer selects Delete Credit Card account in the User Profile UI, and submits the delete request, the following audit data is recorded in payment_accounts_history table other than the columns defined in the regular payment_accounts table. This history table contains all the columns defined in the payment_accounts (regular table) table and additional following columns:

Column	Value	Description
audit_operation	1003	This constant value for the operation is explained in the payment_account_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_account_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

10 The customer selects Create payment reminder in the User Profile UI, and submits the new payment reminder information, the following audit data is recorded in payment_reminders_history table other than the columns defined in the regular payment_reminders table. (This history table contains all the columns defined in the payment_reminders (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1001	This constant value for the operation is explained in the payment_reminder_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_reminder_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

11 The customer selects Update payment reminder in the User Profile UI, and submits the updated payment reminder information, the following audit data is recorded in payment_reminders_history table other than the columns defined in the regular payment_reminders table. (This history table contains all the columns defined in the payment_reminders (regular table) table and additional following columns).

Column	Value	Description
audit_operation	1002	This constant value for the operation is explained in the payment_reminder_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_reminder_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

12 The customer selects Delete payment reminder in the User Profile UI, and submits the delete request for the payment reminder, the following audit data is recorded in payment_reminders_history table other than the columns defined in the regular payment_reminders table. This history table contains all the columns defined in the payment_reminders (regular table) table and additional following columns:

Column	Value	Description
audit_operation	1003	This constant value for the operation is explained in the payment_reminder_const table.
audit_status	1	Status constant value for successful operation. This constant value for the status is explained in the payment_reminder_const table.
audit_reason	none	Description of the audit.
Job_id	0	Because this is a UI operation, the job ID value is 0.
Job_name	NULL	Because this is a UI operation, the job name is NULL.
Timestamp	none	The current system time when an audit occurs.

About Query Files

The following files are provided for each operating system to support queries of the audit tables:

- UNIX:
  - getAuditInfoByAccount.sh
  - getAuditInfoByAccount.sql

- getAuditInfoByPaymentId.sh
- getAuditInfoByPaymentId.sql
- getAuditInfoByPid.sh
- getAuditInfoByPid.sql

- Windows:
  - getAuditDataByAccount.bat
  - getAuditDataByAccount.sql
  - getAuditDataByPaymentId.bat
  - getAuditDataByPaymentId.sql
  - getAuditDataByPid.bat
  - getAuditDataByPid.sql
  - set_audit_isql_options.bat

Running Audit Queries

Audit queries require one of the following arguments:

- Payment ID
- User Account Number
- PID

The audit queries are implemented in batch files, which require the user argument and date range. The results are displayed on the console.

Before running the queries, you must perform setup. The description for each query describes the setup.

Query Audit Data by Payment ID

Displays data from all history tables which have a payment ID column. This query performs a select on each table where the Payment ID matches and the time_stamp is between the fromTime and toTime values. The following tables are queried:

- check_payments_history
- creditcard_payments_history
- payment_bill_summaries_history
- payment_email_history

Query Audit Data by User Account Number

Displays data from all history tables which have a payer ID column. This query performs a select on each table where the payer ID matches Account Number, and whose time_stamp is between fromTime and toTime. The AccountNumber is the account number with the biller (payee_id column). The following tables are queried:

- check_payments_history
- creditcard_payments_history
- payment_bill_summaries_history
- recurring_payments_history

Query Audit Data by PID

Displays data from all the history tables which have a PID column. This query performs a select on each table where the PID matches and whose time_stamp is between fromTime and toTime. The following tables are queried:

- check_payments_history
- creditcard_payments_history
- payment_accounts_history
- recurring_payments_history

Setting Up a Query

Before running the queries, you must perform setup tasks.

To set up a query

1. Set the database connection parameters.
2. Configure TNS Listener for Oracle (Client/Server).
3. Configure DB2 Clients for Windows.
4. Check execution permissions for shell scripts.
5. Specify database connection parameters.

Follow the configuration instructions for your operating system.

Configuring Windows

For Windows, you must edit set_isql_options.bat before running the queries. The file constrains the following line:

```
set ISQL_OPTIONS=-U <username> -P <password> -S <sqlsvr-Servername> -d <database
name>
```

Edit this file and enter your values for username, password, server name and database name. For example:

```
set ISQL_OPTIONS=-U edx1 -P edx1 -S EDXSERVER -d edxDB
```

Configuring UNIX

For UNIX, the database connection string is embedded in the file. You must edit the connection parameters in each file before running the queries. The connection parameters are as follows:

On Oracle:

```
sqlplus <username>/<password>@<TNS name>
```

For example:

```
sqlplus edx1/edxadmin@edxdb
```

TNS Listener for Oracle (Client/Server)

The TNS Listener has to be configured for the Oracle database in Windows and UNIX for client/server.

Permissions for UNIX

Grant execution permissions for shell scripts to run successfully. For example:

```
$ chmod 755 *.sh
```

Running the Queries in Windows and MSSQL

This topic describes how to run queries in Windows and MSSQL.

Querying Audit Data by Payment ID

Change your working directory to the location of the query script files, and run getAuditDataByPaymentId.bat. This file requires three parameters: Payment ID, From Timestamp, and To Timestamp. The execution format is:

getAuditDataByPaymentId *Payment_ID*, *From Date*, *To Date*

For example:

```
getAuditDataByPaymentId 123465564,'2008-01-01','2009-12-12'
```

where:

- *Payment_ID* is numeric.
- *From Date* and *To Date* are in YYYY-MM-DD format.

Querying Audit Data by Account

Change your working directory to the location of the query script files, and run getAuditDataByAccount.bat. This file requires three parameters: Account Number, From Timestamp, and To Timestamp. The execution format is:

getAuditDataByAccount *Account_Number*, *From Date*, *To Date*

For example:

```
getAuditDataByAccount '123465564','2008-01-01’,'2009-12-12'
```

where:

- *Account_Number* is a string.
- *From Date* and *To Date* are in YYYY-MM-DD format.

Querying Audit Data by PID

Change your working directory to the location of the query script files, and run getAuditDataByPid.bat. This file requires three parameters: PID, From Timestamp, and To Timestamp. The execution format is:

getAuditDataByPid *PID*, *From Date, To Date*

For example:

```
getAuditDataByPid '123465564','2008-01-01’,'2009-12-12'
```

where:

- *PID* is a string.
- *From Date* and *To Date* are in YYYY-MM-DD format.

Running the Queries in Oracle Database

This topic describes how to run queries in Oracle Database.

Querying Audit Data by Payment ID

Change your working directory to the location of the query script files, and run getAuditDataByPaymentId.bat. This file requires three parameters: Payment ID, From Timestamp, and To Timestamp. The execution format is:

getAuditDataByPaymentId *Payment_ID*, *From Date, To Date*

For example:

```
getAuditDataByPaymentId 123465564,'2008-01-01’,'2009-12-12'
```

where:

- *Payment_ID* is numeric.
- *From Date* and *To Date* are in YYYY-MM-DD format.

Querying Audit data by Account

Change your working directory to the location of the query script files, and run getAuditDataByAccount.bat. This file requires three parameters: Account Number, From Timestamp, and To Timestamp. The execution format is:

```
getAuditDataByAccount Account_Number,From Date, To Date
```

For example:

```
getAuditDataByAccount '123465564','2008-01-01','2009-12-12'
```

where:

- *Account_Number* is a string.
- *From Date* and *To Date* are in YYYY-MM-DD format.

Querying Audit Data by PID

Change your working directory to the location of the query script files, and run getAuditDataByPid.bat. This file requires three parameters: PID, From Timestamp, and To Timestamp. The execution format is:

```
getAuditDataByPid PID, From Date, To Date
```

For example:

```
getAuditDataByPid '123465564','2008-01-01','2009-12-12'
```

where:

- *PID* is a string.
- *From Date* and *To Date* are in YYYY-MM-DD format.

Running the Queries in UNIX

This topic explains running queries in UNIX and the Oracle Database.

Querying Audit Data by Payment ID

Change your working directory to the location of the query script files, and run getAuditInfoByPaymentId.sh. This file requires three parameters: Payment ID, From Timestamp, and To Timestamp. The execution format is:

```
$ ./getAuditInfoByPaymentId.sh Payment_ID, From Date, To Date
```

For example:

```
$ ./getAuditInfoByPaymentId.sh 123465564 '2008-01-01' '2009-12-12'
```

where:

- *Payment_ID* is numeric.
- *From Date* and *To Date* are in YYYY-MM-DD format.
- Arguments are separated by spaces.

Querying Audit Data by Account

Change your working directory to the location of the query script files, and run getAuditInfoByAccount.sh. This file requires three parameters: Account Number, From Timestamp, and To Timestamp. The execution format is:

```
$ ./getAuditInfoByAccount.sh Account_Number, From Date, To Date
```

For example:

```
& ./getAuditInfoByAccount.sh '123465564' '2008-01-01’ '2009-12-12'
```

where:

- *Account_Number* is a string
- *From Date* and *To Date* are in YYYY-MM-DD format
- Arguments are separated by spaces

Query Audit Data by PID

Change your working directory to the location of the query script files, and run getAuditInfoByPid.sh. This file requires three parameters: PID, From Timestamp, and To Timestamp. The execution format is:

```
$ ./getAuditInfoByPid.sh PID, From Date, To Date
```

For example:

```
$ ./getAuditInfoByPid '123465564' '2008-01-01’ '2009-12-12'
```

where:

- *PID* is a string.
- *From Date* and *To Date* are in YYYY-MM-DD format.
- Arguments are separated by spaces.

Change your working directory to the location of the query script files, and run getAuditDataByPid.sh. This file requires three parameters: PID, From Timestamp, and To Timestamp.

Audit Database

The Oracle Self-Service E-Billing Payment database supports auditing.

Columns for Audit

The following tables have the new columns:

- check_payments_history
- creditcard_payments_history

The history tables have all the columns that the base table has (check_payments and creditcard_payments) plus the columns listed in Table 91.

Table 91. Additional Columns in History Tables

Column Name	Comments
audit_operation	Defined in corresponding constant tables
audit_status	Defined in corresponding constant tables
audit_reason	Description of the audit
job_id	Pwc job ID
job_name	User given job name (see Job Name Entries)
time_stamp	The record insertion time. For example: 1/18/2004 11:47:38 AM

New Tables

The following tables are based on the table name with _history at the end. They have all the columns in the base table, plus the new columns listed in Table 91 on page 260 to support auditing.

- payment_accounts_history
- payment_bill_summeries_history
- payment_reminder_history
- recurring_payments_history

payment_email_history

This table is new, and not based on a previous table. It has the columns listed in Table 92 plus the columns listed in the preceding table to support auditing.

Table 92. Payment Email History Table Columns

Column Name	Comments
type	This indicates the purpose of the email. Possible values are listed in Table 93 on page 261.
payee id	DDN
payer_id	User ID
account_number	Check or credit card number
payment_id	Payment ID
to_address	Receivers email address. If there are multiple addresses, they will be separated by a semicolon.
content	Content; Length of the email content must be truncated based on the Email Content Audit Length configuration parameter.
audit_operation	Defined in corresponding constant tables
audit_status	Defined in corresponding constant tables
audit_reason	Description of the audit
job_id	Pwc job ID
job_name	User given job name (see Job Name Entries)
time_stamp	The record insertion time. For example: 1/18/2004 11:47:38 AM
to_address	Receivers email address. If there are multiple addresses, they will be separated by a semicolon.

Table 93 lists the possible values for email types and description.

Table 93. Email Types

Email Type	Description
0	Unknown email type.
1	A fixed date payment reminder email.
2	Before due date payment reminder email.
3	After due date payment reminder email.
4	Check status notification email.
5	Credit card status notification email.
6	Recurring payment cancelled email.

Table 93. Email Types

Email Type	Description
7	Recurring payment scheduled email.
8	Payment account status notification email.
9	Credit card expiration notification email.

Audit Table Constants

Table 94 lists the tables that have audit information and the names of the corresponding code tables that explain the numeric codes for audit columns. See the tables in your Payment database for the latest descriptions for each code.

Table 94. Audit Table Constants

Constant Table Name	History Table Name
credit_card_const	creditcard_payments_history
check_const	check_payments_history
recurring_payment_const	recurring_payment_history
payment_email_const	payment_email_history
payment_bill_summaries_const	payment_bill_summaries history
payment_account_const	payment_accounts_history
payment_reminders_const	payment_reminders_history

Job Name Entries

User job names are combined with a shortened version of the task name to keep database entries manageable. The name of the job given by the user is combined with a shortened version of the task name as follows:

```
<job name given by the Admin>-<shorten task name>
```

Table 95 shows the shortened name for each job.

Table 95. Job Name Entries

Task Name	Shortened Task Name
CheckSubmitTask	ChkSubTsk
CheckUpdateTask	ChkUpdTsk
PaymentIntegratorTask	PmtIntTsk
CreditCardExpNotifyTask	CCExpNTsk
CreditCardSubmitTask	CCSubTsk
CreditCardUpdateTask	CCUpdTsk

Table 95. Job Name Entries

Task Name	Shortened Task Name
ConfirmEnrollTask	ConEnrTsk
NotifyEnrollTask	NotEnrTsk
RecurPaymentSchedulerTask	RcuSchTsk
RecurPaymentSynchronizerTask	RcuSynTsk
PaymentReminderTask	PmtRmdTsk
SubmitEnrollTask	SubEnrTsk
CustomTask	CustomTsk

Implementing Custom Oracle Self-Service E-Billing Payment Cartridges

You can implement two custom cartridges:

- Implementing a Demonstration Cartridge
- Implementing a Custom Credit Card Cartridge

Implementing a Demonstration Cartridge

Oracle Self-Service E-Billing Payment provides an example cartridge that demonstrates how to implement a custom cartridge. The code is in the /vobs/payment/com/edocs/payment/cassette/demo directory. There are two cartridges:

- **demo_CheckCassette.java**. For check payments.
- **demo_CreditCardCassette.java.** For credit card payments.

The example cartridge delegates all API calls to demo_CheckProcessorProxy.java and demo_CreditCardProcessorProxy.java to communicate with a dummy payment gateway.

If you configure a DDN to use the demonstration cartridge, then you can make payments against it from the user interface.

Implementing a Custom Credit Card Cartridge

The example cartridge is based on the interface com.edocs.payment.cassette.ICreditCardCassette, which extends from com.edocs.payment.cassette.IPaymentCassette, which then extends from com.edocs.payment.cassette.IEnrollmentCassette. In general, do not modify IEnrollmentCassette, because it defines how to verify a credit card when a user enrolls it through the user interface.

To implement the cartridge, extend your cartridge implementation from PaymentCassette, and implement ICreditCardCassette:

```
public class MyCreditCardCassette extends PaymentCassette implements
ICreditCardCassette
```

Use demo_CreditCardCassette.java to create your implementation. You can use three implementation methods:

- IPaymentCassette.getDefaultConfigAttributes()
- ICreditCardCassette.authorize()
- ICreditCardCassette.batchAuthorize()

You must implement IPaymentCassette.getDefaultConfigAttributes() to return a list of parameters (of type com.edocs.payment.config.Attribute), which are used to configure the cartridge. Calling IPaymentCassette.getDefaultConfigAttributes() causes those parameters to be displayed in the Payment Settings of the Command Center, where you can use them to configure the cartridge. These parameters include the global ones, the ones shared by both credit card and check types, and the ones specific to this credit card cartridge. Your implementation of getDefaultConfigAttributes() must at least return the global and shared parameters in that list. See demo_CreditCardCassette.getDefaultConfigAttributes() in the *Oracle Self-Service E-Billing* Javadoc, and the file demo_CreditCardAttributes.java for more information. See "Accessing Oracle Self-Service E-Billing Javadoc" on page 31 for details on accessing the *Oracle Self-Service E-Billing* Javadoc.

If you want to support instant payments, then you must implement the ICreditCardCassette.authorize() method. In this method, you must get the payment information from the ICreditCard object that is passed in, then send it to the payment gateway.

The payment gateway sends back a response, which you use to update the status of the ICreditCard object:

- If the payment is authorized, set the status to settled by calling:

  ```
  ICreditCard.setStatus(CreditCardState.SETTLED);
  ```

- If the payment failed authorization, set status to failed-authorize by calling:

  ```
  ICreditCard.setStatus(CreditCardState.FAILED_AUTHORIZE);
  ```

  You could also call ICreditCard.setTxnErrMsg() to log an error message.

- If there is a system or network error (Payment failed to connect to payment gateway), set the status to failed by calling:

  ```
  ICreditCard.setStatus(CreditCardState.FAILED);
  ```

  You could also call ICreditCard.setTxnErrMsg() to log an error message.

  When you call these methods, Oracle Self-Service E-Billing Payment updates the credit card information in the database. The Oracle Self-Service E-Billing Payment JSP pages get the credit card information from the user and pass the information to the cartridge. After the card is processed, Oracle Self-Service E-Billing Payment updates Oracle Self-Service E-Billing Payment database.

  If your application supports scheduled payments, then you must implement ICreditCardCassette.batchAuthorize(). This method is called by the CreditCardSubmit job, which extracts all the scheduled payments from the database and sends them to the payment gateway. Your cartridge must perform the following actions:

1. Get the scheduled payments from Oracle Self-Service E-Billing Payment database. There are examples of using the APIs in demo_CreditCardCassette.batchSubmit().

2 Loop through the list of payments and send them to the payment gateway. Set the status of each payment the same way as for instant payments. After setting the status and other information, call the Oracle Self-Service E-Billing Payment API to update this credit card back to Oracle Self-Service E-Billing database (note that this is different from Instant payments, because Oracle Self-Service E-Billing Payment does not update the database).

3 Package your custom cartridge. With Oracle WebLogic, package the custom cartridge into Payment_custom.jar which is in the lib directory.

4 Prepopulate Oracle Self-Service E-Billing Payment database.

5 Tell Oracle Self-Service E-Billing Payment about your cartridge implementation class by populating the payment_gateway_configure table. If your cartridge class name is com.edocs.ps.MyCreditCardCartridge, and you want to name it “customCCardCartridge”, use:

6 insert into payment_gateway_configure(GATEWAY,PAYMENT_TYPE,CARTRIDGE_CLASS)values(‘customCCardCartridge’, ‘ccard’, ‘com.edocs.ps.MyCreditCardCartridge’);

7 When you go to Payment Settings of Command Center and configure a DNN for your credit card cartridge, the JSP page will read the list of available cartridges from this table and allow you to select one of them.

8 After you finish all the preceding steps, create a DDN, configure a cartridge for it and then make the payments from UI.

Avoiding Paying a Bill More Than Once

By default, Oracle Self-Service E-Billing Payment allows a bill to be paid more than once. If you want to ensure that a bill can only be paid once, add a unique key constraint on the bill_id field of the check_payments table. You can run the set_unique_bill_id.sql script located in the *EDX_HOME/*payment/db/ directory to set the unique constraint. In the path, *EDX_HOME* is the directory where you installed Oracle Self-Service E-Billing. Note, the bill_id in Oracle Self-Service E-Billing Payment is the same as the doc ID in the Command Center.

If a customer tries to pay a bill that has already been paid (either from the UI or by a previously scheduled recurring payment) after the unique key constraint has been added, the customer will receive an error message saying that the bill has been already paid. If the bill is paid from the UI and a recurring payment tries to pay it again, the payment will fail and an email notification message will be sent to the customer (if recurring payments are configured for that email notification).

Adding this constraint will not prevent a customer from making a payment using a bill ID. For example, a customer can still make a payment directly from the Make Check Payment link, which allows him or her to make a payment without specifying a bill.

The unique key constraint only informs customers that the bill has been paid when they try to pay a bill that has already been paid. If you want to provide additional features, such as disabling the payment button when the bill has already been paid, you must query the database to get that information. Be careful when adding extra functions, because performing additional database queries can affect Oracle Self-Service E-Billing Payment performance. Make sure the proper index has been created if you plan to create a new query.

Handling Multiple Payee ACH Accounts

By default, Oracle Self-Service E-Billing Payment only allows one payee (biller) ACH account for a DDN, which is limited by Payment Settings. However, some billers can have multiple ACH accounts and their users will usually choose to pay to one of the ACH accounts when scheduling a payment. The way that the user chooses the ACH account to pay with can be based on some business rules added to the JSP. The rest of this topic describes a solution to this problem.

The assumptions for this solution are:

- All ACH accounts are at the same bank, which means they have the same immediate origination and immediate destination but different company name and company ID.
- The business logic elements required to route the payment transaction to one ACH account versus another is available or can be made available in the web application and in the execution context of an Oracle Self-Service E-Billing Payment plugin.

Oracle Self-Service E-Billing also assumes there are multiple ACH accounts and there is one DDN for this biller. This DDN is the Real DDN.

To handle multiple payee ACH accounts

1. Create a real DDN. You use this real DDN to configure Payment Settings for one of the ACH accounts.
2. Create virtual DDNs: Create N – 1 virtual DDNs, where each of their Payment Settings is configured to one of the N – 1 ACH accounts, respectively. Make sure the immediate origination and immediate destination are the same for all DDNs but their company name and company ID are different.

   NOTE: There will be no ETL load jobs run against these virtual DDNs. They are used solely for payment purposes.
3. Customize the UI: The UI must employ some business logic to determine which DDN (effectively, ACH account) the payment transaction is to be entered against and set the payee ID of the payment to that DDN.
4. Run the pmtCheckSubmit Job: Configure a single pmtCheckSubmit job under the real DDN and configure it to pull payments from the all the N –1 virtual DDNs in addition to the real DDN. The payments from the same DDN will be under same batch.
5. Run the pmtCheckUpdate Job: pmtCheckUpdate processes the ACH return file. Because return files include returns from all DDNs and the pmtCheckUpdate job can process these returns, create one pmtCheckUpdate job under the real DDN to process all the returned transactions (even though the returns could belong to other virtual DDNs).
6. Run the Payment pmtRecurringPayment Job: A single recurring payment job configured with the real DDN is required. A Recurring Payment plug-in is required to execute the same logic as in scheduled payment; that is, apply the business rules to determine which DDN (effectively, ACH account) the recurring payment must be applied against. Override the preSchedulePayment() method of the plug-in for this purpose.
7. Change the Payment pmtPaymentReminder Job setting: Six payment reminders, one for each DDN, must be configured.

8. Run the pmtARIntegrator Job: The AR_Query.xml file is an XML definition of the database query that queries the Oracle Self-Service E-Billing Payment tables to build the default A/R file. The default query must be customized to include the virtual DDNs. Because the query is using the DDN reference numbers, you must pass that information into the query using one of the following methods:
   - Directly hard code the DDN references numbers in the query, though this is risky in the sense that if the DDN is recreated, your query will fail.
   - Extend the SampleARIntegrator and overwrite the getMap() method and use com.edocs.payment.util.DDNUtil to find out the DDN reference number of a DDN, then set it as a "?" parameter used by the query. In this solution, the DDN names are hard coded but not the DDN reference numbers.
   - Pass in the names of virtual DDNs as a flexible job configuration parameter from the job UI. The getMap() method can then parse the parameter to get the list of virtual DDNs. This method is recommended.
9. Add support for the ACH Prenote: If you are using ACH prenote, then you must create pmtSubmitEnroll, pmtConfirmEnroll and pmtNotifyEnroll jobs for each virtual DDN, which means you will get N prenote ACH files. pmtSubmitEnroll cannot aggregate prenotes from different DDNs into one.

Using Payment APIs

Use the following APIs to customize Payment. These are part of the `com.edocs.common.api.services.payment` package:

- BillDepot
- CustomRecurringPaymentPlugin
- PayPalCreditCardSubmitPlugin
- DummyUserAccountAccessor
- IPayment
- IPaymentAccountService
- IPaymentService
- IRecurringPaymentService
- Payment
- PaymentAccountService
- PaymentConfigurationBean
- PaymentService
- RecurringPaymentService

9 Customizing the Customer Service Representative Application

This chapter covers customizing the CSR application for your implementation. It includes the following topics:

CSR Integration and Impersonation APIs

Struts Web actions are available only as public APIs.

The following packages are available for customizing the Oracle Self-Service E-Billing CSR:

- com.edocs.application.csr.actions
- com.edocs.application.csr.common
- com.edocs.application.csr.exceptions
- com.edocs.application.csr.forms

CSR Capabilities

The Customer Service Representative (CSR) application delivered with Oracle Self-Service E-Billing provides an interface to create and manage CSR administrators and organizations. Through this application a CSR can also impersonate a user. As with Billing and Payment users, when a CSR enrolls in a CSR-enabled application, Oracle Self-Service E-Billing creates profiles in the database. Depending on the CSR roles configured, a CSR can be limited to specific UI views and actions on behalf of another user. SAF authorizes access based on the permissions set for the CSR role.

CSR Access (Impersonate User)

When a CSR logs into the CSR application, he or she intends to administer organizations, search for users or other CSRs, or impersonate another user to provide support for that user. A CSR can see all the users with whom he or she works and click the Impersonate hyperlink for a user.

The CSRImpersonationUrl request attribute is configured in the csr-xma.xml file. The configurable points for impersonate are in the csr.xma.xml file. For details on how to access the *Oracle Self-Service E-Billing* Javadoc for more information about the CSR application, see Accessing Oracle Self-Service E-Billing Javadoc on page 31. Acegi provides the authentication and UMF classes to update, create and delete users. Clicking the impersonate hyperlink executes CSRAction.impersonate().

CSR Application

The CSR application WAR file contains the tiles (*.JSP) for the application. Under the `war/src/main/webapp` directory are a variety of packages containing tiles that address key CSR view functions such as impersonating and finding a CSR's customer (access-cust), enrolling the CSR and searching for a customer's CSR (manage-csr), enrolling the customer (manage-cust), and searching and managing organizations (manage-org).

Under the `\web-actions\src\main\java\com\edocs\application\csr` (compiled source) directory are action, form, and tag classes which comprise the model and controller of the CSR application. The common package contains a variety of CSR helper classes for logging in, enrolling, authentication, and configuration.

See the `war/src/main/webapp/WEB-INF` for the struts configuration JavaBeans and forwarding actions for this CSR application. The tiles configuration also resides in this file.

See the `web-actions\src\main\config\csr.xma.xml` file for how to configure access to the customer application from the CSR application and the list of CSR roles that are enabled.

In the csr.xma.xml file, you must modify properties custAppURL and custLogoutAppURL to the value where the Billing and Payment application is deployed. By default, the property values are:

- https://localhost:7001/ebilling/j_acegi_security_check?
- https://localhost:7001/ebilling/logout.do? respectively

The following properties must point to the I.P. address, for example:

- https://10.1.1.1:7001/ebilling/j_acegi_security_check?
- https://10.1.1.1:7001/ebilling/logout.do?

Contents of csr.xma.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/
dtd/spring-beans.dtd">



<beans>



 <!-- XMA specific definitions -->
```

```
    <bean id="GlobalConfigurationBean"
class="com.edocs.application.csr.common.CSRConfiguration">


     <property name="custAppURL">

        <value>http://10.149.189.225:7006/ebilling/j_acegi_security_check?</value>

     </property>

     <property name="custLogoutAppURL">

        <value>http://10.149.189.225:7006/ebilling/logout.do?</value>

     </property>

     <property name="userNameParam">

        <value>j_username</value>

     </property>

     <property name="passwordParam">

        <value>j_password</value>

     </property>

     <property name="csrParam">

        <value>csr</value>

     </property>

     <property name="activeStatus">

        <value>Active</value>

     </property>

     <property name="inActiveStatus">

        <value>Inactive</value>

     </property>

    </bean>

</beans>
```

10 Input File Specifications and Data Mapping

This chapter describes the input file specifications and data mapping tasks. It includes the following topics:

NOTE: Before running core Extract Transform and Load (ETL) tasks, you must process all billing data files using a customized preprocessor.

For information on running ETL and other jobs, see *Administration Guide for Oracle Self-Service E-Billing*.

Preprocessor Tasks

Oracle Self-Service E-Billing requires that all input bill data files conform to the file format specified in this section. The flat file format is pipe delimited. Each row in the file has specific record type associated with it indicating the type of the record. The preprocessor performs the following tasks:

- Converts data file from an outside billing system file format to the one Oracle Self-Service E-Billing uses.
- Converts all dimension value literal strings into dimension value business keys.

You can use many dimension values in a data file. For example, the data file might contain a record with the following text: service 781-359-1000 Peak 2000 minutes. In this record, Peak could be interpreted as one of Tariff dimension value, which might have a business key PEAK_CALL. For the ETL process to recognize this record indicating that the service with number 781-359-1000 made a total of 2000 call minutes during peak hours, Oracle Self-Service E-Billing expects the file to contain PEAK_CALL (the business KEY string rather than the word Peak, the literal string that came from the billing system).

Data File Loading Tasks

The following tasks must occur when loading a data file:

- Preload any new dimension values for dimension tables.
- Run the preprocessor for each file you are loading.

This approach enables bill files in different languages to be stored with the correct business key value, allowing for data to be aggregated at a later time.

NOTE: Files must be Unicode compliant so that Oracle Self-Service E-Billing can process data in multiple languages.

About ETL File Processing

ETL processes two types of data files:

- Files to populate any new dimension information, including the following 3-character record types (REC_TYPE): 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
- Files to populate fact data into all the fact tables and some dimension tables, including the following 4-character record types (REC_TYPE)s: 0000, 1000, 1100, 1200, 2000, 2100, 3000, 3100, 3200, 3300, 3400, 4000

File Format for Dimension Level Information

This topic shows the file format for dimension level information.

Table 96 shows the file format for payment type information.

Table 96. Payment Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
100	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
100	Payment Type Code	2	50	VARCHAR2	No	COL1	EDX_RPT_PAYMENT_TYPE_DIM	PAYMENT_TYPE_CD
100	Payment Type Name	3	100	VARCHAR2	No	COL2	EDX_RPT_PAYMENT_TYPE_DIM	PAYMENT_TYPE_NAME

Table 97 shows the file format for the adjustment type information.

Table 97. Adjustment Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
110	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
110	Adjustment Type Code	2	50	VARCHAR2	No	COL1	EDX_RPT_ADJUSTMENT_TYPE_DIM	ADJUSTMENT_TYPE_CD
110	Adjustment Type Name	3	100	VARCHAR2	No	COL2	EDX_RPT_ADJUSTMENT_TYPE_DIM	ADJUSTMENT_TYPE_NAME

Table 98 shows the file format for charge type information.

Table 98. Charge Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
120	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
120	Charge Type Code	2	50	VARCHAR2	No	COL1	EDX_RPT_CHARGE_TYPE_DIM	CHARGE_TYPE_CD
120	Charge Type Name	3	100	VARCHAR2	No	COL2	EDX_RPT_CHARGE_TYPE_DIM	CHARGE_TYPE_NAME

Table 99 shows the file format for sub-charge type information.

Table 99. Sub-Charge Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
130	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
130	SubCharge Type Code	2	50	VARCHAR2	No	COL1	EDX_RPT_SUB_CHARGE_TYPE_DIM	SUB_CHARGE_TYPE_CD
130	SubCharge Type Name	3	100	VARCHAR2	No	COL2	EDX_RPT_SUB_CHARGE_TYPE_DIM	SUB_CHARGE_TYPE_NAME

Table 100 shows the file format for plan type information.

Table 100. Plan Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
140	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
140	Plan Type Code	2	50	VARCHAR2	No	COL1	EDX_RPT_PLAN_TYPE_DIM	PLAN_TYPE_CD
140	Plan Type Name	3	100	VARCHAR2	No	COL2	EDX_RPT_PLAN_TYPE_DIM	PLAN_TYPE_NAME

Table 101 shows the file format for product and sub-product type information.

Table 101. Product and Sub-Product Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
150	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
150	Product Code	2	50	VARCHAR2	No	COL1	EDX_RPT_PRODUCT_DIM	PRODUCT_CD
150	Product Desc	3	100	VARCHAR2	No	COL2	EDX_RPT_PRODUCT_DIM	PRODUCT_NAME
150	Subproduct Code	4	50	VARCHAR2	No	COL3	EDX_RPT_SUB_PRODUCT _DIM	SUB_PRODUCT_CD
150	Subproduct Desc	5	100	VARCHAR2	No	COL4	EDX_RPT_SUB_PRODUCT _DIM	SUB_PRODUCT_NAME
150	Subproduct Charges	6	16,2	NUMBER	No	COL5	EDX_RPT_SUB_PRODUCT _DIM	SUB_PRODUCT_CHAR GES
150	Subproduct Unit	7	20	VARCHAR2	No	COL6	EDX_RPT_SUB_PRODUCT _DIM	SUB_PRODUCT_UNIT
150	Product Note1	8	2000	VARCHAR2	No	COL7	EDX_RPT_PRODUCT_DIM	PRODUCT_NOTE1
150	Product Note2	9	2000	VARCHAR2	No	COL8	EDX_RPT_PRODUCT_DIM	PRODUCT_NOTE2
150	Product Note3	10	2000	VARCHAR2	No	COL9	EDX_RPT_PRODUCT_DIM	PRODUCT_NOTE3

Table 102 shows the file format for service usage type information.

Table 102. Service Usage Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
160	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
160	Usage Type Code	2	50	VARCHAR2	No	COL1	EDX_RPT_USAGE_TYPE _DIM	USAGE_TYPE_CD
160	Usage Type Name	3	100	VARCHAR2	No	COL2	EDX_RPT_USAGE_TYPE _DIM	USAGE_TYPE_NAME

Table 103 shows the file format for tariff type information.

Table 103. Tariff Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
170	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
170	Tariff Code	2	50	VARCHAR2	No	COL1	EDX_RPT_TARIFF_DIM	TARIFF_CD
170	Tariff Name	3	100	VARCHAR2	No	COL2	EDX_RPT_TARIFF_DIM	TARIFF_NAME

Table 104 shows the file format for other type (dimension) related information.

Table 104. Other Type (Dimension) Related Information Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
180	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
180	Region Code	2	50	VARCHAR2	No	COL1	EDX_RPT_REGION_DIM	REGION_CD
180	Region Name	3	100	VARCHAR2	No	COL2	EDX_RPT_REGION_DIM	REGION_NAME
190	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
190	Carrier Code	2	50	VARCHAR2	No	COL1	EDX_RPT_CARRIER_DIM	CARRIER_CD
190	Carrier Name	3	100	VARCHAR2	No	COL2	EDX_RPT_CARRIER_DIM	CARRIER_NAME
200	Rec Type	1	3	VARCHAR2	No	REC_TYPE		
200	Calling/ Called City and State	2	100	VARCHAR2	No	COL1	EDX_RPT_AREA_CD_DIM	AREA_CD
200	Calling/ Called Country	3	100	VARCHAR2	No	COL2	EDX_RPT_AREA_CD_DIM	COUNTRY_CD

Statement Level File Format

This topic shows the file format for statement-level information.

Table 105 shows the file format for summary-level detail information.

Table 105. Summary-Level Detail Record Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
0000	Rec Type	1	3	VARCHAR2	Yes	N/A	N/A	N/A
0000	HEADER TYPE	2	20	VARCHAR2	Yes	N/A	N/A	N/A
0000	BILLING SYSTEM	3	20	VARCHAR2	Yes	N/A	N/A	N/A
1000	Rec Type	1	3	VARCHAR2	Yes	N/A	N/A	N/A
1000	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_STATEMENT_FACT	STATEMENT_NUMBER
1000	Company ID	3	20	VARCHAR2	No	COL2	EDX_RPT_COMPANY_DIM	COMPANY_CD
1000	Company Name	4	255	VARCHAR2	No	COL3	EDX_RPT_COMPANY_DIM	COMPANY_NAME
1000	Statement Date	5	8	DATE	No	COL4	EDX_RPT_STATEMENT_FACT	STATEMENT_DATE
1000	Billing Cycle Start Date	6	8	DATE	Yes	COL5	EDX_RPT_STATEMENT_FACT	BILL_CYCLE_START_DATE
1000	Billing Cycle End Date	7	8	DATE	Yes	COL6	EDX_RPT_STATEMENT_FACT	BILL_CYCLE_END_DATE
1000	Previous Balance	8	16,2	NUMBER	No	COL7	EDX_RPT_STATEMENT_FACT	PREVIOUS_BALANCE
1000	Total Payment Posted	9	16,2	NUMBER	No	COL8	EDX_RPT_STATEMENT_FACT	TOTAL_PAYMENT_POSTED
1000	Total Adjustments	10	16,2	NUMBER	No	COL9	EDX_RPT_STATEMENT_FACT	TOTAL_ADJUSTMENTS
1000	Balance Forward Due	11	16,2	NUMBER	No	COL10	EDX_RPT_STATEMENT_FACT	BALANCE_FORWARD_DUE
1000	Total Current Charge Due	12	16,2	NUMBER	No	COL11	EDX_RPT_STATEMENT_FACT	TOTAL_CURRENT_CHARGE_DUE
1000	Total Amount Due	13	16,2	NUMBER	No	COL12	EDX_RPT_STATEMENT_FACT	TOTAL_AMOUNT_DUE
1000	Monthly Service Charges	14	16,2	NUMBER	No	COL13	EDX_RPT_STATEMENT_FACT	MONTHLY_CHARGE_AMT
1000	Usage Charges	15	16,2	NUMBER	No	COL14	EDX_RPT_STATEMENT_FACT	USAGE_CHARGE_AMT
1000	Credits	16	16,2	NUMBER	No	COL15	EDX_RPT_STATEMENT_FACT	CREDIT_ADJUST_AMT
1000	Other Charges	17	16,2	NUMBER	No	COL16	EDX_RPT_STATEMENT_FACT	OTHER_CHARGE_AMT
1000	Taxes and Fees	18	16,2	NUMBER	No	COL17	EDX_RPT_STATEMENT_FACT	TAXES_SURCHARGES_FEE

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
1000	Flex Field_1	19	16,2	NUMBER	No	COL18	EDX_RPT_STATEMENT_FACT	FLEX_FIELD1
1000	Flex Field_2	20	16,2	NUMBER	No	COL19	EDX_RPT_STATEMENT_FACT	FLEX_FIELD2
1000	Flex Field_3	21	16,2	NUMBER	No	COL20	EDX_RPT_STATEMENT_FACT	FLEX_FIELD3
1000	Flex Field_4	22	16,2	NUMBER	No	COL21	EDX_RPT_STATEMENT_FACT	FLEX_FIELD4
1000	Flex Field_5	23	16,2	NUMBER	No	COL22	EDX_RPT_STATEMENT_FACT	FLEX_FIELD5
1000	Minimum Amount Due	24	16,2	NUMBER	No	COL23	EDX_RPT_STATEMENT_FACT	MINIMUM_DUE_AMT
1000	Statement Due Date	25	8	DATE	No	COL24	EDX_RPT_STATEMENT_FACT	STATEMENT_DUE_DATE
1000	Statement Currency	26	50	VARCHAR2	No	COL25	EDX_RPT_STATEMENT_FACT	CURRENCY_TYPE_CD
1000	Statement Country	27	50	VARCHAR2	No	COL26	EDX_RPT_STATEMENT_FACT	COUNTRY_CD
1000	Statement Time Zone	28	50	VARCHAR2	No	COL27	EDX_RPT_STATEMENT_FACT	TIME_ZONE_CD
1000	Note1	29	2010	VARCHAR2	No	COL28	EDX_RPT_STATEMENT_FACT	NOTE1
1000	Note2	30	2010	VARCHAR2	No	COL29	EDX_RPT_STATEMENT_FACT	NOTE2
1000	Note3	31	2010	VARCHAR2	No	COL30	EDX_RPT_STATEMENT_FACT	NOTE3
1000	Note4	32	2010	VARCHAR2	No	COL31	EDX_RPT_STATEMENT_FACT	NOTE4
1000	Note5	33	2010	VARCHAR2	No	COL32	EDX_RPT_STATEMENT_FACT	NOTE5
1000	MEDIA_TYPE	34	50	VARCHAR2	No	COL33	EDX_RPT_STATEMENT_FACT	MEDIA_TYPE
1000	Corporation Account No	35	255	VARCHAR2	No	COL34	EDX_RPT_COMPANY_DIM	FLEX_FIELD1
1000	Corporation Tax ID	36	255	VARCHAR2	No	COL35	EDX_RPT_COMPANY_DIM	FLEX_FIELD2
1000	Street	37	255	VARCHAR2	No	COL36	EDX_RPT_COMPANY_DIM	FLEX_FIELD3
1000	City	38	255	VARCHAR2	No	COL37	EDX_RPT_COMPANY_DIM	FLEX_FIELD4
1000	State	39	255	VARCHAR2	No	COL38	EDX_RPT_COMPANY_DIM	FLEX_FIELD5
1000	Country	40	255	VARCHAR2	No	COL39	EDX_RPT_COMPANY_DIM	FLEX_FIELD6

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
1000	Zipcode	41	10	VARCHAR2	No	COL40	EDX_RPT_COMPANY_DIM	FLEX_FIELD7
1000	Company display name	42	255	VARCHAR2	No	COL41	EDX_RPT_COMPANY_DIM	FLEX_FIELD8

Table 106 shows the file format for statement payment fact information.

Table 106. Statement Payment Fact Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
1100	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE		
1100	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_STATEMENT_PAYMENT_FACT	STATEMENT_KEY
1100	Payment Type Code	3	50	VARCHAR2	Yes	COL2	EDX_RPT_STATEMENT_PAYMENT_FACT	PAYMENT_TYPE_KEY
1100	Payment Amount	4	16,2	NUMBER	Yes	COL3	EDX_RPT_STATEMENT_PAYMENT_FACT	PAYMENT_AMOUNT
1100	Payment Date	5	8	DATE	No	COL4	EDX_RPT_STATEMENT_PAYMENT_FACT	PAYMENT_DATE
1100	Payment Note	6	255	VARCHAR2	No	COL5	EDX_RPT_STATEMENT_PAYMENT_FACT	PAYMENT_NOTE

Table 107 shows the file format for statement adjustment fact information.

Table 107. Statement Adjustment Fact Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
1200	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE		
1200	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_STATEMENT_ADJUST_FACT	STATEMENT_KEY
1200	Adjustment Type Code	3	50	VARCHAR2	Yes	COL2	EDX_RPT_STATEMENT_ADJUST_FACT	ADJUSTMENT_TYPE_KEY
1200	Adjustment Amount	4	16,2	NUMBER	Yes	COL3	EDX_RPT_STATEMENT_ADJUST_FACT	ADJUSTMENT_AMOUNT
1200	Adjustment Date	5	8	DATE	No	COL4	EDX_RPT_STATEMENT_ADJUST_FACT	ADJUSTMENT_DATE
1200	Service Number	6	20	VARCHAR2	No	COL5	EDX_RPT_STATEMENT_ADJUST_FACT	SERVICE_NUMBER
1200	Adjustment Note	7	255	VARCHAR2	No	COL6	EDX_RPT_STATEMENT_ADJUST_FACT	ADJUSTMENT_NOTE

Account Level File Format

This topic shows the account level file formats.

Table 108 shows the file format for account fact charge information.

Table 108. Account Fact Charges (charge Summary for Account) Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
2000	Rec Type	1	4	VARCHAR 2	Yes	REC_TYPE		
2000	Statement Number	2	20	VARCHAR 2	Yes	COL1	EDX_RPT_ACCOUNT_FACT	STATEMENT_KEY
2000	Account Number	3	20	VARCHAR 2	Yes	COL2	EDX_RPT_ACCOUNT_DIM	ACCOUNT_NUM
2000	Account Owner Name	4	255	VARCHAR 2	Yes	COL3	EDX_RPT_ACCOUNT_DIM	CONTACT_NAME
2000	Address1	5	200	VARCHAR 2	No	COL4	EDX_RPT_ACCOUNT_DIM, EDX_RPT_ADDRESS_DIM	ADDRESS1, ADDRESS1
2000	Address2	6	200	VARCHAR 2	No	COL5	EDX_RPT_ACCOUNT_DIM, EDX_RPT_ADDRESS_DIM	ADDRESS2, ADDRESS2
2000	City	7	100	VARCHAR 2	No	COL6	EDX_RPT_ACCOUNT_DIM, EDX_RPT_ADDRESS_DIM	CITY, CITY
2000	State	8	100	VARCHAR 2	No	COL7	EDX_RPT_ACCOUNT_DIM, EDX_RPT_ADDRESS_DIM	STATE, STATE
2000	Country	9	100	VARCHAR 2	No	COL8	EDX_RPT_ACCOUNT_DIM, EDX_RPT_ADDRESS_DIM	COUNTRY, COUNTRY
2000	Zip	10	20	VARCHAR 2	No	COL9	EDX_RPT_ACCOUNT_DIM, EDX_RPT_ADDRESS_DIM	ZIP, ZIP
2000	Monthly Service Charges	11	16,2	NUMBER	Yes	COL 10	EDX_RPT_ACCOUNT_FACT	MONTHLY_CHARGE_AMT
2000	Usage Charges	12	16,2	NUMBER	Yes	COL 11	EDX_RPT_ACCOUNT_FACT	USAGE_CHARGE_AMT
2000	Credits	13	16,2	NUMBER	Yes	COL 12	EDX_RPT_ACCOUNT_FACT	CREDIT_ADJUST_AMT
2000	Other Charges	14	16,2	NUMBER	Yes	COL 13	EDX_RPT_ACCOUNT_FACT	OTHER_CHARGE_AMT
2000	Taxes and Fees	15	16,2	NUMBER	Yes	COL 14	EDX_RPT_ACCOUNT_FACT	TAXES_SURCHARGES_FEE
2000	Total Charge Amount	16	16,2	NUMBER	Yes	COL15	EDX_RPT_ACCOUNT_FACT	TOTAL_CHARGE_AMT
2000	Charge Flag	17	1	VARCHAR 2	Yes	COL16	EDX_RPT_ACCOUNT_FACT	CHARGE_FLAG
2000	Account Type	18	20	VARCHAR 2	No	COL17	EDX_RPT_ACCOUNT_FACT	ACCOUNT_TYPE_KEY
2000	Flex Field_1	19	16,2	NUMBER	No	COL18	EDX_RPT_ACCOUNT_FACT	FLEX_FIELD1
2000	Flex Field_2	20	16,2	NUMBER	No	COL19	EDX_RPT_ACCOUNT_FACT	FLEX_FIELD2
2000	Flex Field_3	21	16,2	NUMBER	No	COL20	EDX_RPT_ACCOUNT_FACT	FLEX_FIELD3

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
2000	Flex Field_4	22	16,2	NUMBER	No	COL21	EDX_RPT_ACCOUNT_FACT	FLEX_FIELD4
2000	Flex Field_5	23	16,2	NUMBER	No	COL22	EDX_RPT_ACCOUNT_FACT	FLEX_FIELD5

Table 109 shows the file format for information about account level charges at the charge type level.

Table 109. Account Level Charges at Charge Type Level Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
2100	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE		
2100	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_ACCOUNT_ CHARGE_FACT	STATEMENT_KEY
2100	Account Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_ACCOUNT_ CHARGE_FACT	ACCOUNT_KEY
2100	Charge Type Code	4	50	VARCHAR2	Yes	COL3	EDX_RPT_ACCOUNT_ CHARGE_FACT	CHARGE_TYPE_KEY
2100	Monthly Service Charges	5	16,2	NUMBER	Yes	COL4	EDX_RPT_ACCOUNT_ CHARGE_FACT	MONTHLY_CHARGE_AMT
2100	Usage Charges	6	16,2	NUMBER	Yes	COL5	EDX_RPT_ACCOUNT_ CHARGE_FACT	USAGE_CHARGE_AMT
2100	Credits	7	16,2	NUMBER	Yes	COL6	EDX_RPT_ACCOUNT_ CHARGE_FACT	CREDIT_ADJUST_AMT
2100	Other Charges	8	16,2	NUMBER	Yes	COL7	EDX_RPT_ACCOUNT_ CHARGE_FACT	OTHER_CHARGE_AMT
2100	Taxes and Fees	9	16,2	NUMBER	Yes	COL8	EDX_RPT_ACCOUNT_ CHARGE_FACT	TAXES_SURCHARGES_FEE
2100	Total Charge Amount	10	16,2	NUMBER	Yes	COL9	EDX_RPT_ACCOUNT_ CHARGE_FACT	TOTAL_CHARGE_AMT
2100	Charge Note	11	255	VARCHAR2	No	COL10	EDX_RPT_ACCOUNT_ CHARGE_FACT	CHARGE_NOTE

Service Level File Format

This topic shows the file format for account level information.

Table 110 shows the file format for information about the total charges at the service agreement level for an account under a statement.

Table 110. Service Level Total Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3000	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE		
3000	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_SERVICE_FACT	STATEMENT_KEY
3000	Account Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_SERVICE_DIM	ACCOUNT_KEY
3000	Service Number	4	20	VARCHAR2	Yes	COL3	EDX_RPT_SERVICE_DIM	SERVICE_NUM
3000	ServiceOwner Name	5	100	VARCHAR2	No	COL4	N/A	N/A
3000	Monthly Service Charges	6	16,2	NUMBER	Yes	COL5	EDX_RPT_SERVICE_FACT	MONTHLY_CHARGE_A MT
3000	Usage Charges	7	16,2	NUMBER	Yes	COL6	EDX_RPT_SERVICE_FACT	USAGE_CHARGE_AMT
3000	Adjustments	8	16,2	NUMBER	Yes	COL7	EDX_RPT_SERVICE_FACT	CREDIT_ADJUST_AMT
3000	Other Charges	9	16,2	NUMBER	Yes	COL8	EDX_RPT_SERVICE_FACT	OTHER_CHARGE_AMT
3000	Taxes and Fees	10	16,2	NUMBER	Yes	COL9	EDX_RPT_SERVICE_FACT	TAXES_SURCHARGES _FEE
3000	Total Charge Amount	11	16,2	NUMBER	Yes	COL10	EDX_RPT_SERVICE_FACT	TOTAL_CHARGE_AMT
3000	Charge Note	12	255	VARCHAR2	No	COL11	EDX_RPT_SERVICE_FACT	CHARGE_NOTE
3000	Flex Field_1	13	16,2	NUMBER	No	COL12	EDX_RPT_SERVICE_FACT	FLEX_FIELD1
3000	Flex Field_2	14	16,2	NUMBER	No	COL13	EDX_RPT_SERVICE_FACT	FLEX_FIELD2
3000	Flex Field_3	15	16,2	NUMBER	No	COL14	EDX_RPT_SERVICE_FACT	FLEX_FIELD3
3000	Flex Field_4	16	16,2	NUMBER	No	COL15	EDX_RPT_SERVICE_FACT	FLEX_FIELD4
3000	Flex Field_5	17	16,2	NUMBER	No	COL16	EDX_RPT_SERVICE_FACT	FLEX_FIELD5

Table 111 shows the file format for the service level on charge type information.

Table 111. Service Level on Charge Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3100	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE	N/A	N/A
3100	Statemen t Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_SERVICE_CHARGE _FACT	STATEMENT_KEY
3100	Account Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_SERVICE_CHARGE _FACT	ACCOUNT_KEY
3100	Service Number	4	20	VARCHAR2	Yes	COL3	EDX_RPT_SERVICE_CHARGE _FACT	SERVICE_KEY
3100	Charge Type Code	5	50	VARCHAR2	Yes	COL4	EDX_RPT_SERVICE_CHARGE _FACT	CHARGE_TYPE_KEY

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3100	Sub Charge Type Code	6	50	VARCHAR2	No	COL5	EDX_RPT_SERVICE_CHARGE_FACT	SUB_CHARGE_TYPE_KEY
3100	Monthly Charge Amount	7	10	NUMBER	Yes	COL6	EDX_RPT_SERVICE_CHARGE_FACT	CHARGE_AMT
3100	Charge Note	8	255	VARCHAR2	No	COL7	EDX_RPT_SERVICE_CHARGE_FACT	CHARGE_NOTE

Table 112 shows the file format for the service charge on product and plan information.

Table 112. Service Charge on Product and Plan Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3200	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE	N/A	N/A
3200	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_SERVICE_PRODUCT_FACT	STATEMENT_KEY
3200	Account Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_SERVICE_PRODUCT_FACT	ACCOUNT_KEY
3200	Service Number	4	20	VARCHAR2	Yes	COL3	EDX_RPT_SERVICE_PRODUCT_FACT	SERVICE_KEY
3200	Plan Type Code	5	50	VARCHAR2	Yes	COL4	EDX_RPT_SERVICE_PRODUCT_FACT	PLAN_TYPE_KEY
3200	Product Code	6	50	VARCHAR2	Yes	COL5	EDX_RPT_SERVICE_PRODUCT_FACT	PRODUCT_CHILD_KEY
3200	Monthly Charge Amount	7	16,2	NUMBER	Yes	COL6	EDX_RPT_SERVICE_PRODUCT_FACT	CHARGE_AMT
3200	Product Note	8	255	VARCHAR2	No	COL7	EDX_RPT_SERVICE_PRODUCT_FACT	PRODUCT_NOTE

Table 113 shows the file format for service charge on usage type information.

Table 113. Service Charge on Usage Type Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3300	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE		
3300	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_SERVICE_USAGE_FACT	STATEMENT_KEY
3300	Account Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_SERVICE_USAGE_FACT	ACCOUNT_KEY
3300	Service Number	4	20	VARCHAR2	Yes	COL3	EDX_RPT_SERVICE_USAGE_FACT	SERVICE_KEY

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3300	Usage Type Code	5	50	VARCHAR2	Yes	COL4	EDX_RPT_SERVICE_USAGE_FACT	USAGE_TYPE_KEY
3300	Total Usage	6	16,2	NUMBER	Yes	COL5	EDX_RPT_SERVICE_USAGE_FACT	TOTAL_USAGE
3300	Usage Unit Code	7	50	VARCHAR2	Yes	COL6	EDX_RPT_SERVICE_USAGE_FACT	UNIT_KEY
3300	Amount	8	16,2	NUMBER	Yes	COL7	EDX_RPT_SERVICE_USAGE_FACT	TOTAL_CHARGE_AMT
3300	Usage Note	9	255	VARCHAR2	No	COL8	EDX_RPT_SERVICE_USAGE_FACT	USAGE_NOTE

Table 114 shows the file format for the service charges on various tariff information.

Table 114. Service Charges on Various Tariff Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
3400	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE	N/A	N/A
3400	Statement Number	2	20	VARCHAR2	Yes	COL1	EDX_RPT_SERVICE_TARIFF_FACT	STATEMENT_KEY
3400	Account Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_SERVICE_TARIFF_FACT	ACCOUNT_KEY
3400	Service Number	4	20	VARCHAR2	Yes	COL3	EDX_RPT_SERVICE_TARIFF_FACT	SERVICE_KEY
3400	Usage Type Code	5	50	VARCHAR2	Yes	COL4	EDX_RPT_SERVICE_TARIFF_FACT	USAGE_TYPE_KEY
3400	Tariff Code	6	50	VARCHAR2	Yes	COL5	EDX_RPT_SERVICE_TARIFF_FACT	TARIFF_KEY
3400	Allowance	7	20	VARCHAR2	Yes	COL6	EDX_RPT_SERVICE_TARIFF_FACT	ALLOWANCE
3400	Total Usages	8	16,2	NUMBER	Yes	COL7	EDX_RPT_SERVICE_TARIFF_FACT	TOTAL_USAGE
3400	Usage Unit Code	9	50	VARCHAR2	Yes	COL8	EDX_RPT_SERVICE_TARIFF_FACT	UNIT_KEY
3400	Billable	10	16,2	NUMBER	Yes	COL9	EDX_RPT_SERVICE_TARIFF_FACT	BILLABLE
3400	Amount	11	16,2	NUMBER	Yes	COL10	EDX_RPT_SERVICE_TARIFF_FACT	CHARGE_AMT

Service Detail Level File Format

Table 115 shows the file format for account level file information.

Table 115. Service Call Usage Details (Voice, Data, Message) Format

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
4000	Rec Type	1	3	VARCHAR2	Yes	REC_TYPE	N/A	N/A
4000	Usage Name	2	50	VARCHAR2	Yes	COL1	EDX_RPT_SERVICE_DETAIL_FACT	USAGE_TYPE_KEY
4000	Statement Number	3	20	VARCHAR2	Yes	COL2	EDX_RPT_SERVICE_DETAIL_FACT	STATEMENT_KEY
4000	Account Number	4	20	VARCHAR2	Yes	COL3	EDX_RPT_SERVICE_DETAIL_FACT	ACCOUNT_KEY
4000	Service Number	5	20	VARCHAR2	Yes	COL4	EDX_RPT_SERVICE_DETAIL_FACT	SERVICE_KEY
4000	Called Date	6	8	DATE	Yes	COL5	EDX_RPT_SERVICE_DETAIL_FACT	DATE_KEY
4000	Called Time	7	10	VARCHAR2	Yes	COL6	EDX_RPT_SERVICE_DETAIL_FACT	DURATION
4000	Called Number	8	20	VARCHAR2	Yes	COL7	EDX_RPT_SERVICE_DETAIL_FACT	CALLED_NUM
4000	Tariff Code	9	50	VARCHAR2	Yes	COL8	EDX_RPT_SERVICE_DETAIL_FACT	TARIFF_KEY
4000	Call Type Code	10	50	VARCHAR2	No	COL9	EDX_RPT_SERVICE_DETAIL_FACT	CALL_TYPE_KEY
4000	Direction Code	11	20	VARCHAR2	Yes	COL10	EDX_RPT_SERVICE_DETAIL_FACT	DIRECTION_KEY
4000	Service Type Code	12	50	VARCHAR2	No	COL11	EDX_RPT_SERVICE_DETAIL_FACT	SERVICE_TYPE_KEY
4000	Total Usages	13	16,2	NUMBER	Yes	COL12	EDX_RPT_SERVICE_DETAIL_FACT	TOTAL_USAGE
4000	Usages Unit Code	14	50	VARCHAR2	Yes	COL13	EDX_RPT_SERVICE_DETAIL_FACT	UNIT_KEY
4000	Other Charge	15	16,2	NUMBER	Yes	COL14	EDX_RPT_SERVICE_DETAIL_FACT	OTHER_CHARGE_AMT
4000	Total Charge	16	16,2	NUMBER	Yes	COL15	EDX_RPT_SERVICE_DETAIL_FACT	TOTAL_CHARGE_AMT
4000	Calling City and State	17	100	VARCHAR2	Yes	COL16	EDX_RPT_SERVICE_DETAIL_FACT	CALLING_AREA_CD_KEY
4000	Calling Country Code	18	100	VARCHAR2	Yes	COL17	EDX_RPT_SERVICE_DETAIL_FACT	CALLING_AREA_CD_KEY
4000	Reference Number	19	100	VARCHAR2	No	COL18	EDX_RPT_SERVICE_DETAIL_FACT	REFERENCE_NUM
4000	Carrier Code	20	50	VARCHAR2	No	COL19	EDX_RPT_SERVICE_DETAIL_FACT	CARRIER_KEY
4000	Region Code	21	50	VARCHAR2	No	COL20	EDX_RPT_SERVICE_DETAIL_FACT	REGION_KEY
4000	Note1	22	2000	VARCHAR2	No	COL21	EDX_RPT_SERVICE_DETAIL_FACT	NOTE1
4000	Note2	23	2000	VARCHAR2	No	COL22	EDX_RPT_SERVICE_DETAIL_FACT	NOTE2
4000	Note3	24	2000	VARCHAR2	No	COL23	EDX_RPT_SERVICE_DETAIL_FACT	NOTE3

RECORD TYPE	FIELD	POSITION	MAX LENGTH	DATA TYPE	REQ?	STAGING TABLE (STG_CDR) COLUMN NAME	DB_TABLE_NAME	DB_COLUMN_NAME
4000	Note4	25	2000	VARCHAR2	No	COL24	EDX_RPT_SERVICE_DETAIL_FACT	NOTE4
4000	Note5	26	2000	VARCHAR2	No	COL25	EDX_RPT_SERVICE_DETAIL_FACT	NOTE5
4000	Called City and State	27	100	VARCHAR2	Yes	COL26	EDX_RPT_SERVICE_DETAIL_FACT	CALLED_AREA_CD_KEY
4000	Called Country Code	28	100	VARCHAR2	Yes	COL27	EDX_RPT_SERVICE_DETAIL_FACT	CALLED_AREA_CD_KEY

File Record and Table Mapping

Table 116 shows the mapping between record types and Oracle Self-Service E-Billing database tables.

Table 116. File Record and Table Mapping Table

Record Type	Table Name	Pre-Populated
100	EDX_RPT_PAYMENT_TYPE_DIM	Can be populated from data file
110	EDX_RPT_ADJUSTMENT_TYPE_DIM	Can be populated from data file
120	EDX_RPT_CHARGE_TYPE_DIM	Can be populated from data file
130	EDX_RPT_SUB_CHARGE_TYPE_DIM	Can be populated from data file
140	EDX_RPT_PLAN_TYPE_DIM	Can be populated from data file
150	EDX_RPT_PRODUCT_DIM	Can be populated from data file
150	EDX_RPT_SUB_PRODUCT_DIM	Can be populated from data file
160	EDX_RPT_USAGE_TYPE_DIM	Can be populated from data file
170	EDX_RPT_TARIFF_DIM	Can be populated from data file
180	EDX_RPT_REGION_DIM	Can be populated from data file
190	EDX_RPT_CARRIER_DIM	Can be populated from data file
200	EDX_RPT_AREA_CD_DIM	Can be populated from data file
1000	EDX_RPT_STATEMENT_FACT	Fact Data
1000	EDX_RPT_COMPANY_DIM	Can be populated from data file
1000	EDX_RPT_PERIOD_DIM	Can be populated from data file
1000	EDX_RPT_COUNTRY_DIM	Can be populated from data file
1000	EDX_RPT_COMPANY_DIM	Can be populated from data file
1000	EDX_RPT_CURRENCY_TYPE_DIM	Can be populated from data file

Record Type	Table Name	Pre-Populated
1000	EDX_RPT_TIME_ZONE_DIM	Can be populated from data file
1100	EDX_RPT_STATEMENT_PAYMENT_FACT	Fact Data
1200	EDX_RPT_STATEMENT_ADJUST_FACT	Fact Data
2000	EDX_RPT_ACCOUNT_DIM	Populates from data file
2000	EDX_RPT_ADDRESS_DIM	Populates from data file
2000	EDX_RPT_ACCOUNT_FACT	Fact Data
2100	EDX_RPT_ACCOUNT_CHARGE_FACT	Fact Data
3000	EDX_RPT_SERVICE_DIM	Populates from data file
3000	EDX_RPT_SERVICE_FACT	Fact Data
3100	EDX_RPT_SERVICE_CHARGE_FACT	Fact Data
3200	EDX_RPT_SERVICE_PRODUCT_FACT	Fact Data
3300	EDX_RPT_SERVICE_USAGE_FACT	Fact Data
3400	EDX_RPT_SERVICE_TARIFF_FACT	Fact Data
4000	EDX_RPT_SERVICE_DETAIL_FACT	Fact Data

Internationalization Support Settings

Oracle Self-Service E-Billing supports multiple languages.

Unicode data storage requires a Unicode Database solution, which involves creating a Unicode-based database using UTF-8 as the encoding not only for CHAR and VARCHAR2 character datatypes but also for all SQL names and literals. To implement the Unicode Database solution, the Oracle Self-Service E-Billing database character set is configured as AL32UTF8, the Oracle name for UTF-8.

The NLS_LENGTH_SEMANTICS parameter in the init.ora (parameter) file of the target Oracle database is set to CHAR instead of the default BYTE to enable global character semantic support.

Character semantics changed the way multibyte characters were treated in Oracle Database. Instead of doubling or tripling column or variable precision, setting NLS_LENGTH_SEMANTICS = CHAR causes Oracle Database to treat storage of the string 'Today' the same as the Japanese string ''. With this setting, glyphs (characters) are the measure for column and variable precision rather than the bytes required to store the characters.

Oracle Self-Service E-Billing uses the init.ora file parameter settings shown in Table 117 for the OLAP and OLTP database instances.

Table 117. Internationalization Support Settings

Parameter	**Value**
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA
NLS_NUMERIC_CHARACTERS	.
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	YYYY-MM-DD
NLS_DATE_LANGUAGE	AMERICAN
NLS_CHARACTERSET	AL32UTF8
NLS_SORT	BINARY
NLS_TIME_FORMAT	HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT	DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT	HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT	DD-MON-RR HH.MI.SSXFF AM TZR
NLS_DUAL_CURRENCY	$
NLS_NCHAR_CHARACTERSET	AL16UTF16
NLS_COMP	BINARY
NLS_LENGTH_SEMANTICS	CHAR
NLS_NCHAR_CONV_EXCP	FALSE
NLS_LANGUAGE	AMERICAN

Index