Oracle® iPayment

Implementation Guide

Release 11*i*

Part No. A95478-05

May 2005

ORACLE®

Oracle iPayment Implementation Guide, Release 11*i*

Part No. A95478-05

Contributors: Ramasubramanian Balasundaram, Jonathan Leybovich, Rajiv Menon, Elizabeth Newell, Aalok Shah

Contents

3 Using iPayment with External Front End Applications

4 Using iPayment with External Payment Systems

D Back-End APIs for Gateways

E Extensibility

Send Us Your Comments

Oracle iPayment Implementation Guide, Release 11*i*

Part No. A95478-05

Oracle welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us via the postal service.

- Electronic mail: appsdoc_us@oracle.com
- FAX: (650) 506-7200 Attention: Oracle Applications Documentation
- Postal service:
  Oracle Corporation
  Oracle Applications Documentation
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Welcome to Release 11.5.10 of the Oracle iPayment Implementation Guide. This guide is your primary source of information to implement Oracle iPayment.

This preface contains these topics:

- Audience for this Guide
- How To Use This Guide
- Other Information Sources
- Installation and System Administration
- Other Implementation Documentation
- Training and Support
- Do Not Use Database Tools to Modify Oracle Applications Data
- About Oracle
- Your Feedback

Audience for this Guide

This guide assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Oracle iPayment

  If you have never used Oracle iPayment, Oracle suggests you attend one or more of the Oracle iPayment training classes available through Oracle University.
- The Oracle Applications graphical user interface.

  To learn more about the Oracle Applications graphical user interface, read the *Oracle Applications User's Guide*.

See Other Information Sources for more information about Oracle Applications product information.

How To Use This Guide

This document contains the information you need to implement Oracle iPayment.

This manual contains these chapters and appendixes:

- Chapter 1, "Overview"

  Chapter 1 describes the important issues that should be considered prior to implementing Oracle iPayment.

- Chapter 2, "Configuring iPayment"

  Chapter 2 describes detailed information on the tasks you should perform to implement Oracle iPayment.

- Chapter 3, "Using iPayment with External Front End Applications"

  Chapter 3 explains the public APIs used in Oracle iPayment with external front end applications.

- Chapter 4, "Using iPayment with External Payment Systems"

  Chapter 4 explains the APIs used in Oracle iPayment with external payment systems.

- Appendix A, "Risk Management"

  Oracle iPayment supports risk management. Electronic commerce applications can incorporate this feature to detect fraudulent payments. Appendix A explains how electronic commerce applications can utilize the risk management functionality of Oracle iPayment.

- Appendix B, "Error Handling"

  Oracle iPayment returns a response object to each API that an electronic commerce application calls. Appendix B provides detailed information on the errors that can occur in Oracle iPayment.

- Appendix C, "iPayment PL/SQL APIs"

  Appendix C describes the public PL/SQL API used by Oracle iPayment. Electronic commerce applications (EC-Apps) may use these interfaces for processing credit card and bank account transfer payment related operations.

- Appendix D, "Back-End APIs for Gateways"

  Appendix D describes the back-end processing APIs used in Oracle iPayment.

- Appendix E, "Extensibility"

  Oracle iPayment can be integrated with a back end payment system by implementing oracle.apps.iby.extend.TxnCustomizer interface. Appendix E explains how to implement this interface.

- Appendix F, "Configuring CyberCash Servlet"

  Appendix F describes how to configure the CyberCash servlet.

- Appendix G, "Configuring Paymentech"

  Appendix G describes how to configure the Paymentech servlet.

- Appendix H, "Configuring FDC North"

  Appendix H describes how to configure the FDC North servlet.

- Appendix I, "Configuring Concord EFSnet"

  Appendix I describes how to implement the Concord EFSnet servlet.

- Appendix J, "Configuring Citibank"

  Appendix J describes how to configure the Citibank credit card servlet.

- Appendix K, "Profile Options"

  Appendix K describes profile options for Oracle iPayment.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Other Information Sources

You can choose from many sources of information, including online documentation, training, and support services, to increase your knowledge and understanding of Oracle iPayment.

If this guide refers you to other Oracle Applications documentation, use only the Release 11*i* versions of those guides.

Online Documentation

All Oracle Applications documentation is available online (HTML or PDF).

- **PDF Documentation-** See the Online Documentation CD for current PDF documentation for your product with each release. This Documentation CD is also available on Oracle*MetaLink* and is updated frequently.
- **Online Help -** You can refer to Oracle Applications Help for current HTML online help for your product. Oracle provides patchable online help, which you can apply to your system for updated implementation and end user documentation. No system downtime is required to apply online help.
- **Release Content Document -** See the Release Content Document for descriptions of new features available by release. The Release Content Document is available on Oracle*MetaLink*.
- **About document -** Refer to the About document for information about your release, including feature updates, installation information, and new documentation or documentation patches that you can download. The About document is available on Oracle*MetaLink*.

Related Documentation

Oracle iPayment shares business and setup information with other Oracle Applications products. Therefore, you may want to refer to other guides when you set up and use Oracle iPayment.

You can read the guides online by choosing Library from the expandable menu on your HTML help window, by reading from the Oracle Applications Document Library CD included in your media pack, or by using a Web browser with a URL that your system administrator provides.

If you require printed guides, you can purchase them from the Oracle Store at http://oraclestore.oracle.com.

Guides Related to All Products

Oracle Applications User's Guide

This guide explains how to enter data, query, run reports, and navigate using the graphical user interface (GUI). This guide also includes information on setting user profiles, as well as running and reviewing reports and concurrent processes.

You can access this user's guide online by choosing "Getting Started with Oracle Applications" from any Oracle Applications help file.

Guides Related to This Product

Oracle Payables User Guide

This manual describes how accounts payable transactions are created and entered into Oracle Payables. This manual also contains detailed setup information for Oracle Payables and discusses suppliers, banks, invoices, and also explains how to create payments and run reports.

Oracle Receivables User Guide

This manual describes how accounts receivables transactions are created and entered into Oracle Receivables. This manual also contains detailed setup information for Oracle Payables and discusses customers, banks, invoices, and reporting.

Oracle iPayment Concepts and Procedures Guide

This manual describes an overview of iPayment and its components, and provides process-oriented, task-based procedures for using the user interface to set up the application and perform essential business tasks. This manual also provides details on the integration of iPayment and Oracle Payables and viewing the key performance metrics such as transaction summaries, payee summaries, and other critical performance indicators.

Oracle iReceivables Implementation Guide

This manual describes the setup tasks that you need to perform for iReceivables and information you need to configure iReceivables to suit your business requirements.

Oracle Collections User Guide

This manual explains the key features and process flows in Collections.

Oracle iStore Implementation and Administration Guide

This manual explains the information needed to implement, administer, and maintain Oracle iStore.

Installation and System Administration

Oracle Applications Concepts

This guide provides an introduction to the concepts, features, technology stack, architecture, and terminology for Oracle Applications Release 11*i*. It provides a useful first book to read before an installation of Oracle Applications. This guide also introduces the concepts behind Applications-wide features such as Business Intelligence (BIS), languages and character sets, and Self-Service Web Applications.

Installing Oracle Applications

This guide provides instructions for managing the installation of Oracle Applications products. In Release 11*i*, much of the installation process is handled using Oracle Rapid Install, which minimizes the time to install Oracle Applications and the Oracle technology stack by automating many of the required steps. This guide contains instructions for using Oracle Rapid Install and lists the tasks you need to perform to finish your installation. You should use this guide in conjunction with individual product user guides and implementation guides.

Upgrading Oracle Applications

Refer to this guide if you are upgrading your Oracle Applications Release 10.7 or Release 11.0 products to Release 11*i*. This guide describes the upgrade process and lists database and product-specific upgrade tasks. You must be either at Release 10.7 (NCA, SmartClient, or character mode) or Release 11.0, to upgrade to Release 11*i*. You cannot upgrade to Release 11*i* directly from releases prior to 10.7.

Maintaining Oracle Applications

Use this guide to help you run the various AD utilities, such as AutoUpgrade, AutoPatch, AD Administration, AD Controller, AD Relink, License Manager, and others. It contains how-to steps, screenshots, and other information that you need to run the AD utilities. This guide also provides information on maintaining the Oracle applications file system and database.

Oracle Applications System Administrator's Guide

This guide provides planning and reference information for the Oracle Applications System Administrator. It contains information on how to define security, customize menus and online help, and manage concurrent processing.

Oracle Alert User's Guide

This guide explains how to define periodic and event alerts to monitor the status of your Oracle Applications data.

Oracle Applications Developer's Guide

This guide contains the coding standards followed by the Oracle Applications development staff and describes the Oracle Application Object Library components that are needed to implement the Oracle Applications user interface described in the *Oracle Applications User Interface Standards for Forms-Based Products*. This manual also provides information to help you build your custom Oracle Forms Developer forms so that the forms integrate with Oracle Applications.

Oracle Applications User Interface Standards for Forms-Based Products

This guide contains the user interface (UI) standards followed by the Oracle Applications development staff. It describes the UI for the Oracle Applications products and how to apply this UI to the design of an application built by using Oracle Forms.Oracle Applications System Administrator's Guide

This guide provides planning and reference information for the Oracle Applications System Administrator. It contains information on how to define security, customize menus and online help, and manage concurrent processing.

Other Implementation Documentation

Oracle Applications Product Update Notes

Use this guide as a reference for upgrading an installation of Oracle Applications. It provides a history of the changes to individual Oracle Applications products between Release 11.0 and Release 11*i*. It includes new features, enhancements, and changes made to database objects, profile options, and seed data for this interval.

Oracle Workflow Administrator's Guide

This guide explains how to complete the setup steps necessary for any Oracle Applications product that includes workflow-enabled processes, as well as how to monitor the progress of runtime workflow processes.

Oracle Workflow Developer's Guide

This guide explains how to define new workflow business processes and customize existing Oracle Applications-embedded workflow processes. It also describes how to define and customize business events and event subscriptions.

Oracle Workflow User's Guide

This guide describes how Oracle Applications users can view and respond to workflow notifications and monitor the progress of their workflow processes.

Oracle Workflow API Reference

This guide describes the APIs provided for developers and administrators to access Oracle Workflow.

Oracle Applications Flexfields Guide

This guide provides flexfields planning, setup and reference information for the Oracle iPayment implementation team, as well as for users responsible for the ongoing maintenance of Oracle Applications product data. This guide also provides information on creating custom reports on flexfields data.

Oracle eTechnical Reference Manuals

Each eTechnical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for a specific Oracle Applications product. This information helps you convert data from your existing applications, integrate Oracle Applications data with non-Oracle

applications, and write custom reports for Oracle Applications products. Oracle eTRM is available on Oracle*Metalink*

Oracle Self–Service Web Applications Implementation Manual

This manual contains detailed information about the overview and architecture and setup of Oracle Self–Service Web Applications. It also contains an overview of and procedures for using the Web Applications Dictionary.

Oracle Order Management APIs and Open Interfaces Manual

This manual contains up-to-date information about integrating with other Oracle Manufacturing applications and with your other systems. This documentation includes APIs and open interfaces found in Oracle Order Management Suite.

Other Information Sources

For more information, see the latest versions of the following manuals.

- *iPayment JavaDoc* (Available on Metalink)
- Apache Server Documentation (http://www.apache.com)
- Apache's mod-ssl documentation (http://www.mod-ssl.org/docs)
- Java Developer's Guide (http://www.sun.com)

Training and Support

Training

Oracle offers a complete set of training courses to help you and your staff master Oracle iPayment and reach full productivity quickly. These courses are organized into functional learning paths, so you take only those courses appropriate to your job or area of responsibility.

You have a choice of educational environments. You can attend courses offered by Oracle University at any one of our many education centers, you can arrange for our trainers to teach at your facility, or you can use Oracle Learning Network (OLN), Oracle University's online education utility. In addition, Oracle training professionals can tailor standard courses or develop custom courses to meet your needs. For example, you may want to use your organization structure, terminology, and data as examples in a customized training session delivered at your own facility.

Support

From on-site support to central support, our team of experienced professionals provides the help and information you need to keep Oracle iPayment working for you. This team includes your technical representative, account manager, and Oracle’s large staff of consultants and support specialists with expertise in your business area, managing an Oracle server, and your hardware and software environment.

Do Not Use Database Tools to Modify Oracle Applications Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle Applications data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle Applications tables are interrelated, any change you make using Oracle Applications can update many tables at once. But when you modify Oracle Applications data using anything other than Oracle Applications, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

About Oracle

Oracle develops and markets an integrated line of software products for database management, applications development, decision support, and office automation, as well as Oracle Applications, an integrated suite of more than 160 software modules for financial management, supply chain management, manufacturing, project systems, human resources and customer relationship management.

Oracle products are available for mainframes, minicomputers, personal computers, network computers and personal digital assistants, allowing organizations to integrate different computers, different operating systems, different networks, and even different database management systems, into a single, unified computing and information resource.

Oracle is the world's leading supplier of software for information management, and the world's second largest software company. Oracle offers its database, tools, and applications products, along with related consulting, education, and support services, in over 145 countries around the world.

Your Feedback

Thank you for using Oracle iPayment and this user guide.

Oracle values your comments and feedback. In this guide is a reader's comment form that you can use to explain what you like or dislike about Oracle iPayment or this user guide. Mail your comments to the following address or call us directly at (650) 506-7000.

Oracle Applications Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Or, send electronic mail to appsdoc_us@oracle.com.

1

Overview

This chapter presents the important issues you should consider prior to implementing Oracle iPayment. Topics include:

- Planning Your Implementation
- Which Payment System Should You Use?
- Is Your Merchant Terminal Based or Host Based?
- What Electronic Commerce Applications Are You Using?
- Which APIs Should Electronic Commerce Applications Handle?
- Which Bank Account Transfer Operations Should You Implement?
- Which Credit Card and Purchase Card Operations to Implement?
- Which Risk Factors Should You Implement?
- Does Your Application Need to Present Information in Different Languages?
- Installing Oracle iPayment

Planning Your Implementation

Before you begin implementing Oracle iPayment, you must make several key business and application decisions.

The following sections help you find answers to these questions. Your answers determine which APIs you should use, which parameters you must pass, and which code samples are relevant to your applications to help you implement Oracle iPayment.

Which Payment System Should You Use?

Oracle iPayment requires partnering with a third party payment system for communicating to bank processors and acquirer's banks. Some of the factors which may help you decide are:

- Do you want to use an existing integration or build your own?
- Do you want to integrate with a vendor offering a product or a service?
- Do you want to integrate with a gateway or a processor model payment system?
- Does the payment system support the payment methods that you are implementing, (for example, Concord supports credit card, PINless debit card, Purchase card and Electronic Funds Transfer transactions)

The following table lists the back-end payment systems that are integrated and shipped with iPayment, as well as the operations each payment sustem supports.

Back End Payment System	Credit Card	Purchase Card	PINless debit card	Inbound	EFT Online Validation	Outbound
CyberCash*	Yes	No	No	Yes**	No	No
Paymentech	Yes	Yes	Yes	Yes	Yes	No
First Data (North)	Yes	Yes	No	No	No	No
Concord EFS	Yes	Yes	Yes	Yes	No	No
Citibank Credit Card	Yes	No	No	No	No	No

* CyberCash no longer accepts new customers

** US ACH only

> **Note:** The supported operations may change. Contact the payment system for the most recent information.

The following table lists some back-end payment systems that provide their own field-installable servlets for integration with iPayment,and the operations VeriSign supports.

Back End Payment System	Credit Card	Purchase Card	PINless debit card	Electronic Funds Transfer	
				Inbound	Outbound
VeriSign	Yes	No	No	Yes	No

Is Your Merchant Terminal Based or Host Based?

The choice of being a terminal-based or host-based merchant is generally determined by the business type, the number of transactions per day, and the model supported by the acquiring bank. As a developer of an EC application, you only need to know the type of payee for the application you are developing, so that you can choose the appropriate APIs.

If your payee is terminal-based, then you may integrate the Close Batch API into the EC application, to enable the payee to perform close batches through the EC application instead of the payment system's native interface. If your payee is host-based, then you may want to ignore the Close Batch API because the processor automatically closes batches at predetermined intervals.

If the payee is host-based, then payment capture takes care of getting the payment, and reconciliation is not necessary. Therefore, the Close Batch API and the Query Batch Status API are not required for host-based payees.

Note: Processor-model payment systems are always host-based.

What Electronic Commerce Applications Are You Using?

The choice of electronic commerce applications depends on the applications you are using with iPayment namely:

- Preintegrated Oracle applications
- External front-end applications

Preintegrated Oracle applications include iStore, Order Capture, Telesales, Order Management, Oracle Receivables, Oracle Payables and Collections.

> **Note:** You need to follow the instructions present in the respective documentation for iPayment's interaction with preintegrated Oracle applications.

For external front-end applications, you need to decide the iPayment front-end API to be implemented and then implement them.

Which APIs Should Electronic Commerce Applications Handle?

Oracle iPayment provides payment instrument registration APIs for registering payment instruments such as credit cards, bank accounts, PINless debit cards, and purchase cards. It also provides payment processing APIs that can perform credit card, PINless debit card, and purchase card operations, such as, authorization, capture, and bank account transfer operations. Risk management APIs are provided to perform risk analysis. Based on your requirements, you need to decide the operations your electronic commerce (EC) applications needs to implement.

Note: Each preintegrated Oracle application implements the iPayment API relevant to its operation. Therefore, if you are planning to use preintegrated Oracle application, you need not implement anything further.

Payment Instrument Registration APIs

These APIs are mandatory if you decide to use the offline payment processing feature of Oracle iPayment Payment APIs in your EC application. EC applications can implement registration of payment instruments using Payment Instrument Registration APIs, and instrument identifiers, that are generated, during payment requests with Oracle iPayment.

Payment Processing APIs

You must decide whether to:

- Implement online or offline payment processing or both
- Accept credit card, PINless debit card payments, purchase cards, or bank account transfers or a combination
- Implement the risk functionality to detect fraudulent transactions

Risk Management APIs

Oracle iPayment provides two Risk management APIs. If you want to perform risk evaluation independently and not as part of the Authorization API, then these independent APIs can be called from your EC application.

The following information describes some of the decisions you have to make if you are accepting bank account transfer payments, credit card, PINless debit card, or purchase card payments.

Which Bank Account Transfer Operations Should You Implement?

Oracle iPayment supports offline bank account payment requests. Besides payment requests for bank account transfers, Oracle iPayment also supports modification, cancellation, and inquiry operations. There is no need for any special settlement operations.

Oracle iPayment also supports Electronic Funds Transfer online validations for bank account transfers. The validations are online and real-time whereas the actual funds transfer are perfomed offline. The funds transfer are not performed online because the transaction requires one or two business days for completion.

Note: EFT Online Validations are not offered by all payment systems.

Which Credit Card and Purchase Card Operations to Implement?

Oracle iPayment provides APIs for authorization, settlement, and querying transaction status. You do not have to use all these APIs. You can choose to have your EC application handle only authorization, thus reducing development costs but requiring the payee to do more work for settlement and reconciliation.

The following table compares the authorization only with authorization and settlements.

Authorization Only	Authorization and Settlement
The integration effort is relatively minimal because you have to use no more than two APIs.	The integration effort is significant because you have to use several APIs.
The payee has to settle transactions through the native payment system administration tool. (For example, by going to the payment system's web page).	The payee can settle transactions directly through the EC application.

Note: For setting up credit card payments in iStore, see the latest *Oracle iStore and iMarketing Implementation Guides*.

Which Risk Factors Should You Implement?

Oracle iPayment provides risk management functionality for credit card, PINless debit card, and purchase card transactions in EC applications for both business-to-business and business-to-consumer models. Oracle iPayment includes a number of built-in risk factors and provides the option to the payees to run or not run the risk evaluation functionality for each payment operation. Payees can also run the risk evaluation for operations which handle amounts exceeding a specified amount.

A risk factor includes any information which a payee wants to use to evaluate the risk of the customer wanting to buy goods or services from the payee. Examples of risk factors are: address verification, time of purchase, payment amount, etc. These risk factors can be configured for each payee (merchant or biller).

Risk management functionality enables payees and EC service providers to manage the risk involved in processing transactions online. It allows businesses to have any number of predefined risk factors to verify the identity of their customers, assess their customer credit rating, and risk rating in a secure environment. For more information, see *Oracle iPayment Concepts and Procedures Guide.*

Does Your Application Need to Present Information in Different Languages?

If your application needs to present information in different languages or character sets, then you need to know about National Language Support (NLS).

Would Your Application Need National Language Support (NLS)?

Your application may need to use NLS if either of the following is true:

- The EC application and the payment system use different languages or character sets. For example, the EC application may use a Japanese EUC character set while the payment system uses a Japanese Shift-JIS character set.
- Clients of the EC application use different languages. For example, a web site that is expecting customers from all over the world might want to present its EC application in different languages for different customers.

To enable character conversion in all these environments, the EC application and the payment system must convey the language and character set information to Oracle iPayment.

How Do Applications Convey Language Information to Oracle iPayment?

To communicate information about the language and character set to Oracle iPayment, an EC application and payment system servlet must pass a special parameter (`NlsLang`). This parameter is a part of every API included in this guide.

`NlsLang` is an optional parameter. If your EC application does not need to handle non-Latin1 character set parameters and does not need to communicate to clients or payment systems in different languages, you do not need to use this parameter.

How does Oracle iPayment Use NlsLang?

If the EC application does not pass the `NlsLang` parameter, Oracle iPayment passes information from the EC application to the payment service servlet without performing any conversion of character sets.

If the EC application does pass a value for `NlsLang` to Oracle iPayment, then Oracle iPayment tries to convert parameters based on the value of `NlsLang` before sending those parameters to the payment system servlet.

To do so, Oracle iPayment first checks its database for the list of preferred and optional languages for that payment system. The information in the database reflects what the Oracle iPayment administrator entered using the Oracle iPayment administration user interface.

Second, Oracle iPayment does one of the following, depending on what it finds in the database:

- If the database lists a language that matches the value of `NlsLang`, Oracle iPayment keeps the value of `NlsLang` and passes it to the payment system servlet.
- If the database does not list a language matching the value of `NlsLang`, Oracle iPayment uses the language specified as the preferred language for that payment system, thus changing the value of `NlsLang` before sending it to the payment system servlet.

Finally, Oracle iPayment converts the values of other parameters so that they are sent to the payment system servlet in the language specified by `NlsLang`.

This conversion process works only in one direction. From the EC application to the payment system servlet. If the payment system sets up `NlsLang` when it sends the data back, Oracle iPayment uses that information only to store the value of `OapfVendErrmsg` in its database. Oracle iPayment does not convert data sent from the payment system servlet back to the EC application.

Format of the NLS_LANG Parameter

The value of this parameter follows the same format as Oracle Server's `NLS_LANG` environment variable:

```
language_territory.charset
```

For example, `JAPANESE_JAPAN.JA16EUC` is a valid value for `NlsLang`.

Format of the Response Body Data From Payment System Servlets

Oracle iPayment does not convert the response received from the payment system servlet in the response body. It only treats the data as binary and sends it directly to the EC application.

However, if any binary information is sent (such as wallet data), then Oracle iPayment converts the character set of the binary data to that specified by the value of `NlsLang`.

Installing Oracle iPayment

To install Oracle iPayment, see the latest *Installing Oracle Applications 11i*.

2

Configuring iPayment

This chapter presents detailed information on the tasks you should perform to implement Oracle iPayment. Topics include:

- Overview of Oracle iPayment Implementation Steps
- Creating an Oracle iPayment User
- Overview of iPayment Servlets
- Implementing Field Installable Servlets
- Configuring Oracle iPayment Servlets
- Configuring the ECApp Servlet
- Setting Up SSL Security for the ECApp Servlet
- Configuring iPayment Sample Servlet
- Configuring iPayment Loopback Servlet
- Setting Up SSL Security for Payment System Servlet Communication
- Enabling the Scheduler
- Registering Electronic Commerce Applications
- Loading Risky Instruments
- Enabling the XML Framework
- Setting up Entities in the Oracle iPayment User Interface

Overview of Oracle iPayment Implementation Steps

This table gives you an overview about the steps that are required for implementing Oracle iPayment in different scenarios.

Implementation Steps	Oracle iPayment with other preintegrated Oracle Applications[1]	Standalone new install or 3i standalone implementation upgrading to 11i standalone	3i implementation upgrading to 11i[2]
Creating an Oracle iPayment User	Mandatory	Mandatory	Mandatory
Configuring the ECApp Servlet	Mandatory	Mandatory if you are using PL/SQL APIs	Mandatory
Configuring iPayment Sample Servlet	Mandatory only if you want to test iPayment installation for a payment gateway.	Mandatory only if you want to test iPayment installation for a payment gateway.	Mandatory only if you want to test iPayment installation for a payment gateway.
Configuring iPayment Loopback Servlet	Mandatory only if you want to test iPayment installation for a processor model payment system.	Mandatory only if you want to test iPayment installation for a processor model payment system.	Mandatory only if you want to test iPayment installation for a processor model payment system.
Configuring CyberCash Servlet	Mandatory only if you are using Cybercash as a payment system	Mandatory only if you are using Cybercash as a payment system	Mandatory only if you are using Cybercash as a payment system
Configuring Paymentech	Mandatory only if you are using Paymentech as a payment system	Mandatory only if you are using Paymentech as a payment system	Mandatory only if you are using Paymentech as a payment system
Configuring FDC North	Mandatory only if you are using FDC (North) as a payment system	Mandatory only if you are using FDC (North) as a payment system	Mandatory only if you are using FDC (North) as a payment system
Configuring Concord EFSnet	Mandatory only if you are using Concord as a payment system	Mandatory only if you are using Concord as a payment system	Mandatory only if you are using Concord as a payment system
Configuring Citibank	Mandatory only if you are using Citibank as a payment system	Mandatory only if you are using Citibank as a payment system	Mandatory only if you are using Citibank as a payment system
Enabling the Scheduler	Not Necessary	Mandatory	Not Necessary

Implementation Steps	Oracle iPayment with other preintegrated Oracle Applications[1]	Standalone new install or 3i standalone implementation upgrading to 11i standalone	3i implementation upgrading to 11i[2]
Loading Risky Instruments	Not Utilized.The integrated applications do not utilize this functionality	Optional	Not Applicable
Enabling the XML Framework	Mandatory	Mandatory	Mandatory
Setting up Entities in the Oracle iPayment User Interface	Mandatory	Mandatory	Mandatory
Implementing Electronic Commerce Applications APIs	Not Necessary-has already been implemented	Mandatory	Not Applicable
Implementing Back-end Payment System APIs	Mandatory if you are not using existing integration.	Mandatory if you are not using an existing integration.	Implement as a servlet and not as a cartridge

[1] Preintegrated Oracle Applications include iStore, Order Capture, Telesales, Order Management, Oracle Receivables, Oracle Payables and Collections.

[2] 3i Implementation upgrading to 11i but retaining existing functionality (same as a non-Oracle client).

Creating an Oracle iPayment User

You can access the Oracle iPayment user interfaces by creating separate users based on the business needs. For example, by using this procedure to create an iPayment administrative user, the Oracle iPayment administrator is differentiated from the sysadmin user thereby allowing better security. You can then log in as this created user. A user can have multiple responsibilities and roles.

Note: The iPayment Administrator User Interface uses Oracle Application's standard OA HTML framework. The new user interface replaces the JTF UI for iPayment administration. To access the daily business close reports, you continue to login using the JTF UI.

Prerequisites

- Oracle 11i installed.
- Oracle iPayment with responsibility, menu, security roles, and permissions should be installed.

Steps

1. Access the Oracle iPayment user interface through the Oracle Admin Console at the following URL:

   http://<machine>:<port>/OA_HTML/US/ICXINDEX.htm

   Replace the machine and the port with the name of the machine and the port where the Apache server is installed.

2. Login as —

   Username: SYSADMIN

   Password: SYSADMIN

3. Navigate to the Users tabs on the Admin Console, and click on the User Maintenance link in the side navigation bar.
4. Navigate to Security > User. Click Define.
5. Enter the username and password for the application user.
6. Assign the iPayment Payment Administrator responsibility to the application user in the Responsibility tabbed region.

7. Save your work.
8. Use the application user to login and access the new iPayment Administrator UI through the standard Self Service Applications Login page.
9. Log off as sysadmin and login through the Self Service Applications Login page using the new username.

   You can also use this application user to access the daily business close reports that is in the JTF UI. In order to do this, you need to perform the following additional setup steps.

Note: You can link a user to more than one responsibility. For information, on the valid responsibilities in iPayment, see Assigning Roles and Responsibilities to an iPayment User.

To Access the JTF User Interface

1. In the Define User form, add the "iPayment Daily Business Close User" responsibility for the user you created above.
2. Save your changes.
3. Navigate to Profile > System Profile option in the form.
4. Click User. Type the newly created user name in the field.
5. Search for JTF_PROFILE_DEFAULT% profile option using wildcards.
6. Edit the profile fields for the user id that was created.

   JTF_PROFILE_DEFAULT_APPLICATION: appID (for Oracle iPayment it is 673).

   There are additional, less important profiles which can also be set up (i.e., ICX_ LANGUAGE). If these profiles are not set up, the site's default profiles are used. For a complete list of profile options, see 'System Profile Options' in the latest *CRM Foundation Components Implementation Guide*. For more information, see 'Setting User Responsibilities for an existing AOL User' in the latest *CRM Foundation Components Concepts and Procedures Guide*.
7. Click Save.
8. Exit from Self Service Applications.
9. Login to the Admin Console as SYSADMIN from the following URL:

   http://<machine>:<port>/OA_HTML/jtflogin.jsp
10. Navigate to the Users tabs on the Admin Console.

11. Click on the User Maintenance link in the side navigation bar.
12. Query the user that you created using the Self Service Applications.
13. Click on the user name link to open the User-Details page. Click Roles.
14. Select the roles associated with the responsibility. Move it to the Assigned Roles column.

    For more information on the valid roles for each responsibility in iPayment, see Assigning Roles and Responsibilities to an iPayment User.
15. Click Update.

    For more information, see 'Assigning Roles to the User' in the latest *CRM Foundation Components Implementation Guide.*
16. You have just finished the additional steps required for accessing the JTF UI. Now you may access the daily business close reports UI through the JTF login page with the user.
17. Log off as sysadmin and login through the JTF Login page using the newly created user name to access the iPayment Transaction reporting UI.

    When a user with multiple responsibilities logs in for the first time, the system prompts the user to select a default responsibility.

Assigning Roles and Responsibilities to an iPayment User

You can assign roles and responsibilities to a new user or to an existing user. To create a user, see Creating an Oracle iPayment User. You can assign multiple responsibilities to a user. For users with "iPayment Daily Business Close User" responsibility, you should link the appropriate role as defined in the table below.

Note: Administrative users can only access the new UIs if they have the iPayment Payment Administrator responsibility assigned to them. For existing administrative users, you need to manually assign this responsibility.

This table lists the seeded iPayment responsibilities and their description.

Responsibility	Description
System Administrator for iPayment	Users with this responsibility has access to "Visibility Configuration" UIs to create and update iPayment Visibility Classes.
iPayment Payment Administrator	This is the new responsibility required to access the new UI using the Self Service framework.
iPayment for Payroll Clerk	Users with this responsibility can only see the operations screen for outbound bank payments
iPayment for Receivables Clerk	Users with this responsibility can only see the operations screen for inbound bank remittances.
iPayment Daily Business Close User	Users with this responsibility have access to the iPayment Transaction Reporting UI.

This table lists the responsibility and the corresponding roles

Responsibility	Role	Permissions
iPayment Daily Business Close User	IBY_DBC_ROLE	IBY_DBC_VIEW_PERMISSION

Overview of iPayment Servlets

Oracle iPayment provides a complete payment solution. The Payment System Integration Model allows integration with third party payment systems for credit card, purchase card, PINless debit card, and bank account transfer processing. The payment systems communicate with the payment processors and the acquires/banks to process payment transactions.

Oracle iPayment integrations packaged with EBusiness suite products start functioning after you install and configure the ECApp servlet. The ECApp servlet provides an interface to the iPayment engine to process payment related operations such as authorization, capture, and return. The ECApp servlet is primarily used for the PL/SQL APIs provided by iPayment. Click on the following link for steps on configuring the ECApp servlet.

- Configuring the ECApp Servlet

There are three options for integrating with third party payment systems, also known as back end payment systems.

- Use the payment system integration provided by Oracle iPayment. For inbound payments using credit, PINless debit cards, or purchase cards Oracle iPayment provides integration with CyberCash, Paymentech, Concord EFS, Citibank, and First Data (North). Use the following links to guide you to implement the appropriate payment system in your organization.

  ### Sample Servlet

  - Configuring iPayment Sample Servlet
  - Configuring iPayment Loopback Servlet

  ### Credit Card/Purchase Card Servlets

  - Appendix F, "Configuring CyberCash Servlet"
  - Appendix G, "Configuring Paymentech"
  - Appendix H, "Configuring FDC North"
  - Appendix I, "Configuring Concord EFSnet"
  - Appendix J, "Configuring Citibank"

- Use the payment integration provided by the vendor. Many payment system vendors have partnered with Oracle to build integration with Oracle iPayment. These field

installable servlets are available from Oracle's payment system partners, such as VeriSign.

- Build integration by using the published Payment System Integration Model for credit cards, PINless debit cards, and purchase cards. See Implementing Back-end Payment System APIs for instructions on how to build your own field installable servlets.

Implementing Field Installable Servlets

Oracle iPayment supports field-installable servlets. These are payment system servlets not bundled with Oracle iPayment. This feature allows a payee to acquire a new, additional, or upgraded payment system servlet and configure it in the same way as the payment system servlets bundled with Oracle iPayment.

The ability to add field-installable servlets provides payment flexibility and allows new releases of Oracle iPayment and the payment systems to be independent of each other. It also enables electronic commerce applications to customize the payment system for their specific needs and regions.

Field-installable payment system servlets for Oracle iPayment are usually available from Oracle's payment system partners, such as VeriSign.

Configuring Oracle iPayment Servlets

Oracle iPayment has several Java Servlets, some of which are not configured as a part of Oracle Applications Rapid Installation process. Follow the instructions given below to configure them.

These instructions assume that you know how to configure Java Servlets with Apache Web Server. In particular, we assume you know where to find Apache and JServ configuration files on the node where the Apache Web Server is installed. For more information, see Apache documentation available at http://www.apache.org.

Note: This guide includes instructions for several platforms. We assume you are familiar with the particular platform you are configuring. For example, environment variables in UNIX look like $ABC/lib. In Windows NT, the environment variables look like %ABC%\lib.

Logon to Web Server Node

Log on to your Web Server node as the applmgr user and run the environment file to set up the Oracle Applications environment. Your environment should have the following variable defined:

$IBY_TOP refers to the top-level directory of Oracle iPayment installation. In Windows NT or 2000, Oracle iPayment top level directory is located in %APPL_TOP%\iby.

Note: Apache and Jserv may not interpret environment variables in their configuration files. Expand any environment variables of the type $ABC to the values they actually contain on your installation. For example, if $IBY_TOP is defined at /u03/apps/iby/11.5, you need to replace $IBY_TOP with /u03/apps/iby/11.5 in the instructions below.

Verify That a Common Servlet Zone is Configured in Your Environment.

A servlet zone should already exist in your Apache Web Server installation. Check the jserv.properties file for a line beginning with “zones=”. If you see such a line, a servlet zone has been set up. By default this zone is called “root”. The root zone is associated with the zone.properties file. It you are using a different zone and not the root zone, you may have to make the changes listed below in a different <SERVLET_ZONE>.properties file. Similarly, your servlets will be invoked as:

http://<hostname>:<port>/<SERVLET_ZONE>/<servlet_name>

Click the links below to configure the respective servlets:

- Configuring the ECApp Servlet
- Appendix F, "Configuring CyberCash Servlet"
- Appendix G, "Configuring Paymentech"
- Appendix H, "Configuring FDC North"
- Appendix I, "Configuring Concord EFSnet"

Setting iPayment JVM parameters

iPayment back end servlets may be installed on a different host from the iPayment engine. When installed on a different machine, the servlet has no access to the profile values set up in the data base. In such case, you can set the values using the JVM parameters for each iPayment instance.

```
-DAFLOG_ENABLED=TRUE
-DAFLOG_LEVEL=<vlue  such as ERROR>
-DAFLOG_MODULE=iby%
-DAFLOG_FILENAME=<file name with path such as /tmp/aferror1.log>
-DIBY_XML_BASE=<value of XML base>
```

You can set these parameters by passing them as command line arguments to the Java executable.

See iPayment Profile Options for more details on the specific profile options that can be set in the database.

Load Balancing Recommendations

The maximum number of concurrent requests that a servlet can process without blocking is equal to the number of JServ instances running in its servlet zone. You should have a number of JServ instances running equal to the average number of concurrent requests, if not slightly more since, under load balancing, JServ instances are randomly chosen, making it possible that two concurrent requests could be sent to a JServ instance when an idle one is already available.

Running multiple JServ instances within a zone does not significantly add to your CPU load versus running a single instance, but it does add to your memory load as each instance requires its own JVM. On Solaris, each JVM requires over 6MB of main memory though less than 4MB are actually used, since JVMs share common libraries.

Note: With most processors, the online port is never released by iPayment, but must be continuously held. Additionally, most processors impose a limitation of having only a single active connection, often for reasons such as security.

As the online port is never released by iPayment, the recommendation is to have a dedicated JVM for that servlet. Therefore we can have only one servlet per JVM and hence one port. So iPayment processor model servlets do not support load balancing.

Configuring the ECApp Servlet

An ECApp servlet is the only front-end servlet in iPayment. You need to configure the ECApp servlet in order to use the PL/SQL API of Oracle iPayment and for Oracle iPayment 3i Backward Compatibility API.

Set up the Virtual Path Mapping for ECApp Servlet

The ECApp Servlet is automatically set up (and named ibyecapp) by Rapid Install. You can use the following instructions to set up the servlet manually, or to confirm that the ECApp servlet is configured properly.

Add the following line to your zone.properties file in the Servlet Aliases section:
servlet.ecapp.code=oracle.apps.iby.ecservlet.ECServlet

This allows the ECAppservlet to be invoked as: http://<hostname>:<port>/servlet/ecapp

Where <hostname> is the name of the server on which you are running Oracle iPayment. <port> is the port number where ECAppservlet has been installed.

Setting Up SSL Security for the ECApp Servlet

If the ECAPP servlet is located at an HTTPS URL, then you must set these two wallet profile options: IBY: Wallet Location and IBY: Wallet Password. See iPayment Profile Options for more details.

The public certificate of the web server which hosts the ECAPP servlet must have been imported as a trusted certificate into the Oracle wallet. See Using Oracle Wallet Manager in the *Oracle Advanced Security Administration Guide*.

Configuring iPayment Sample Servlet

The iPayment sample servlet is a gateway model servlet that you can use to test your iPayment implementation without having to register with a real payment system or set up and configure the payment system specific servlet. The sample servlet only supports core iPayment operations such as authorization, capture, and return for credit cards.

You can use the sample servlet to test the integration between your EC application and iPayment. All transactions sent to the sample servlet should succeed, unless the amount matches certain pre-set values, in which case an error is induced. You can use the integration to simulate error scenarios and test error handling in the calling EC application.

This table lists the pre-defined amounts and their associated error codes.

Amount	Error Message
1001	Communication error when contacting the gateway. Please try again.
1002	Given order id used for a previous transactions.
1004	A parameter to this transactions is either malformed or missing.
1005	Generic BEP error occurred. Please check error code.
1008	Transaction. type is not valid or not supported for this merchant.
1016	Internal BEP failure. Please check error code.
1017	Account does not have sufficient funds to complete this transaction.
1019	Invalid credit card number/expiration date.
1020	Authorization declined.
1021	Voice authorization code incorrect.

Installing the Sample Servlet

Use the following steps to configure the sample servlet.

1. Add the following alias statement to the configuration file of the servlet zone you wish the sample servlet to run in:

   ```
   servlet.oramipp_lop.code=oracle.apps.iby.bep.loop.LoopBackServlet
   ```

   Note: This line should already be in the properties file after you have installed iPayment. You only need to verify that it exists.

2. In the same configuration file, provide the following servlet parameters:

This table lists the is zone-wide parameters (set by a statement of the form servlet.default.initArgs=).

Parameter	Example Value	Description
errorfile	/tmp/error.log	Debug file used to write errors and stack traces to.
debugfile	/tmp/debug.log	Log file used to write debugging messages to.
debug	true, false	Turns debugging on or off.

Configuring Sample Servlet as a Payment System

Once the sample servlet is installed and configured, the servlet must be added as a payment system in order to be used. Login to the iPayment administrative GUI as the administrative user and create a payment system for the sample servlet with the following values:

Name: Sample Servlet

Suffix: lop

Payment System Type: Gateway

Base URL: example- http://localhost:8080/servlets

Administration URL: http://www.yourcompany.net

Supported Payment Instrument: Credit Card

> **Note:** The sample payment system should already exist in the iPayment setup. You only need to verify that it exists.

Adding a Merchant Account

For each payee that uses the sample servlet, enter any value for the payment system identifier:

example - Loop

Testing the Sample Payment System

To test the sample payment system, create a transaction using the pages on the Operations tab in the iPayment administrative UI. Verify that you have a routing rule which routes the transaction to the sample payment system and that your transaction matches the routing rule. For more information, see Understanding Routing Rules and Managing Operations.

Configuring iPayment Loopback Servlet

The iPayment loopback servlet is a processor model servlet that you can use to test your iPayment implementation without having to register with a real payment system or set up and configure the payment system specific servlet. The loopback servlet supports core iPayment operations such as authorization, capture, and return for credit cards in addition to inbound and outbound bank payments.

You can use the loopback servlet to test the integration between your EC application and iPayment. All transactions sent to the sample servlet should succeed unless the amount matches certain pre-set values, in which case an error is induced. You can use the integration to simulate error scenarios and thus test error handling in the calling EC application.

The processor model servlet does not connect to any back-end payment system but emulates the behavior of a payment system returning successful responses to requests.

See Understanding Gateway-Model and Processor-Model Payment Systems in the *Oracle iPayment Concepts and Procedures Guide* for more details.

This table lists the pre-defined amounts and their associated error codes for credit card transactions. The loopback servlet should return Success for any credit card transaction with an amount other than the ones specified in the table below.

Amount	Error Message
10	Invalid Merchant Account.
20	Decline - Do not honor
30	Expired card
40	Hold call: Pick up card - Lost
50	Hold call: Pick up card - Stolen
60	Insufficient funds
70	Expired Card

Installing the Loopback Servlet

The processor model loopback servlet requires no database connectivity and can be installed on a different host from iPayment. To install on a different host, follow these steps:

1. Copy directory $APPL_TOP/java and directory $IBY_TOP/xml to the new machine.

2. Add $APPL_TOP/java to the CLASSPATH of the Jserv instance the servlet will run and set the "xmlbase parameter" to the location of the copied $IBY_TOP/xml. For details on setting the "xmlbase" parameter, see Setting iPayment JVM parameters.
3. Follow the configuration steps.

Configuration

The following configuration steps are mandatory regardless of whether iPayment and the loopback servlet are on the same machine or not:

1. Add the following alias statement to the configuration file of the servlet zone you wish the Processor Model Loopback Servlet servlet to run in:

```
servlet.oramipp_lpr.code=oracle.apps.iby.bep.proc.loopproc.LoopProcServlet.
```

2. In the same configuration file, provide the following servlet parameters:

   For setting the zone-wide parameters, see Table G–1.

   This table lists parameters particular to the Loopback servlet (set by a statement of the form servlet.oramipp_lpr.initArgs=).

Parameter	Example Value	Description
ARCHIVE	/var/archive	Directory where iPayment response files will be written to. If communication between iPayment and the servlet fails in the middle of a transaction and iPayment retries that transaction at a later date, the archive directory will allow the servlet to know the original results of the transaction and so forward those to iPayment instead of re-attempting the request (thus avoiding double billing or double authorization).
MAX_ARCHIVE_AGE	10	Maximum age (in days) that a response file will be saved in the archive. The FDC North servlet will remove all responses in the archive older than this age every time it starts.
LPR_ONLINE_IP	192.168.0.1	Please specify any IP address of any valid host machine to which the servlet can establish a connection. This is a technical requirement and no data is sent to this machine.
LPR_ONLINE_PORT	8000	Port number to use along with the above IP address.
LOCAL_BATCH_DIR	/tmp/batch	Directory where batch files to are written to.

Parameter	Example Value	Description
LOCAL_EFT_ BATCH_DIR	test/12345	Directory where inbound and outbound payment files are written.
LOCAL_ QUERY_DIR	test/data/12345	Directory where query files are picked up from.

Configuring Loopback Servlet as a Payment System

Once the loopback servlet is installed and configured, the servlet must be added as a payment system in order to be used. Login to the iPayment administrative GUI as the administrative user and create a payment system for the loopback servlet with the following values:

Name: iPayment Loopback Servlet

Suffix: lpr

Payment System Type: Processor

Base URL: example- http://localhost:8080/servlets

Administration URL: http://localhost:8080/servlets

Supported Payment Instrument: credit card, bank account (BR and DD), bank payment

Adding a Processor Model Loopback Servlet Merchant Account

For each payee that will you want to use Processor Model Loopback Servlet, you can enter any value as a Payment System Identifier.

example - Loop

Enabling the Scheduler

Because the loopback servlet is a processor-model payment servlet, all transactions except authorizations should always be OFFLINE transactions. When a BATCHCLOSE operation is submitted, the iPayment engine picks up and sends the transactions to the loopback servlet. The servlet does not submit these to any payment system, but the transactions are updated to a successful status emulating the behavior of a real payment system.

The following is a list of valid tasks you may submit form the scheduler:

BATCHCLOSE

BATCHQUERY

PDCBATCHCLOSE

PDCBATCHQUERY

PDCBATCHRETRY

EFTBATCHCLOSE

EFTBATCHRETRY

EFTPBATCHRETRY

EFTPBATCHCLOSE

Testing the Sample Payment System

To test the loopback payment system, create a test credit card transaction using the pages on the Operations tab in the iPayment administrative UI. Verify that you have a routing rule which routes the transaction to the processor model loopback payment system and that your transaction matches this rule. For more information see Routing Rules and Managing Operations in the *Oracle iPayment Concepts and Procedures Guide.*

You can also generate inbound and outbound bank payment test transactions from Oracle Payables and Oracle Receivables and test the iPayment implementation by defining appropriate routing rules.

Setting Up SSL Security for Payment System Servlet Communication

When Oracle iPayment communicates with the payment system servlets, the information exchanged may be sensitive information such as credit card numbers. If the communication is not secure, it poses a security risk.

The security risk increases in the following circumstances:

- If Oracle iPayment and the payment systems are installed on separate machines
- If Oracle iPayment is running outside your firewall

Steps

- To set up a back end payment system servlet with secured sockets layer follow the procedures in Apache's mod-ssl documentation (http://www.mod-ssl.org/docs). Make sure that your SSL server has a complete certificate chain to the root certificate. SSL's client toolkit requires it.
- Set up the BASE URL parameter of back end payment system using https as the protocol.

Setting Up SSL Runtime for Oracle iPayment

Oracle iPayment requires a set of runtime libraries for supporting SSL communication. These runtime SSL libraries are included with the Oracle 8*i* distribution, but are not installed on an applications tier by default. If you are using Oracle iPayment, you must follow these steps to manually configure SSL on your web server.

To configure the SSL:

1. Copy SSL runtime libraries to $JAVA_TOP.
2. Log on to your web server as the applmgr user and run the environment file for the appropriate product group.
3. Go to the $JAVA_TOP directory, create a subdirectory "ssl", and enter that subdirectory. For example:

   % cd $JAVA_TOP

   % mkdir ssl

   % cd ssl
4. Copy the following three files from any 8*i* installation to the current directory:

$ORACLE_HOME/jlib/javax-ssl-1_1.jar

$ORACLE_HOME/jlib/jssl-1_1.jar

$ORACLE_HOME/lib/libnjssl8.so

Note: $ORACLE_HOME in this case refers to your 8*i* directory, not the default Oracle Home, which is based on 8.0.6.

Note: If you do not have an 8*i* installation on your web server, you can copy these files from your database server using the ftp command.

---

5. Set up the runtime environment variables.

   If you are building your electronic commerce application as a servlet and JServ is set up to start automatically, you need to modify CLASSPATH and LD_LIBRARY_PATH in your servlet engine's configuration.

   If your JServ is set up to start manually, you need to modify the CLASSPATH and LD_LIBRARY_PATH in your shell environment variables, or in the script used to start JServ (for example, jservctl).

   Here is an example for modifying these variables in the Apache servlet engine (JServ) configuration file. For Apache JServ, you have to edit the jserv.properties file to set the CLASSPATH and LD_LIBRARY_PATH environment variables. To add the two SSL jar files from step 1 to the CLASSPATH, add the following lines to jserv.properties:

   wrapper.classpath=$JAVA_TOP/ssl/javax-ssl-1_1.jar

   wrapper.classpath=$JAVA_TOP/ssl/jssl-1_1.jar

   To add the shared library from step 1 to the LD_LIBRARY_PATH, you must find the line in jserv.properties that begins with:

   wrapper.env=LD_LIBRARY_PATH=

   and add the following to the end of that line:

   $JAVA_TOP/ssl

   Note: Use a colon to separate the directory you are adding from the existing ones.

If there is no such LD_LIBRARY_PATH line, create one by adding the following line to jserv.properties:

wrapper.env=LD_LIBRARY_PATH=$JAVA_TOP/ssl

If you have a stand-alone application, you need to modify CLASSPATH and LD_LIBRARY_PATH. Append:$JAVA_TOP/ssl/javax-ssl-1_1.jar: $JAVA_TOP/ssl/javax-ssl-1_1.jar to CLASSPATH and append:$JAVA_TOP/ssl to LD_LIBRARY_PATH environment variable.

Note: You may not have defined the $JAVA_TOP environment variable in your environment. In that case, you should include the fully qualified physical path.

Enabling the Scheduler

The iPayment scheduler provides the ability to handle payment transactions that cannot be processed in real-time. Such transactions may be of two kinds - transactions that can be processed some time after they are submitted to iPayment, or transactions where the back-end payment system cannot process requests in real-time. Scheduling is also useful for automating recurrent associated tasks such as batch closes. Batch closes are performed in a processor-model payment system like Paymentech.

The iPayment scheduler can be configured to perform specific tasks with each invocation. The tasks to be performed are specified through task parameters.

For the scheduler to run successfully, ensure that jsdk.jar library and ApacheJServ.jar are in the CLASSPATH of the machine where the scheduler is running.

Registering Electronic Commerce Applications

All the APIs that an electronic commerce application calls must pass its identifier, which lets Oracle iPayment track the application that the requests are coming from. The identifier generated during registration must be stored by the application. You must only register applications that are not part of the Oracle e-Business suite. All electronic commerce application needs to pass the identifier in the API calls. Oracle iPayment provides an ECConfig utility, to add, modify, or list electronic commerce applications.

Requirements for Setting up and Using the ECConfig Utility

- Java executable in your application environment
- $APPL_TOP/java in your CLASSPATH environment variable.

  This is included in the classpath after you set up the applications environment
- These two jar files must also exist in the path:

  /iAS/Apache/Jsdk/lib/jsdk.jar
  /iAS/Apache/Jserv/libexec/ApacheJServ.jar

Using the EcConfig Utility

- To add an electronic commerce application, use the following command:

```
java-DJTFDBCFILE=<dbc file location>-Dframework.Logging.system.filename=<log
file> -Dservice.Logging.common.filename=<logfile> oracle.apps.iby.ecapp.EcConfig
add “Ec App Name” “Short Name”
Example: java-DJTFDBCFILE=<dbc file
location>-Dframework.Logging.system.filename=<log file>
-Dservice.Logging.common.filename=<logfile> oracle.apps.iby.ecapp.EcConfig add
“my ec application”  “myapp”
```

- To modify a registered electronic commerce application, use the following command:

```
java-DJTFDBCFILE=<dbc file location>-Dframework.Logging.system.filename=<log
file> -Dservice.Logging.common.filename=<logfile> oracle.apps.iby.ecapp.EcConfig
modify <id> 'Ec App Name' 'Short Name'
```

<id> is the identifier of the electronic commerce application that was generated while adding the electronic commerce application. You can also retrieve the identifiers of applications using the list command.

Example: java-DJTFDBCFILE=<dbc file
location>-Dframework.Logging.system.filename=<log file>

```
-Dservice.Logging.common.filename=<logfile> oracle.apps.iby.ecapp.EcConfig
modify 1234 “ec app name” “ecapp”
```

- To list all the registered electronic commerce applications use the following command:

```
java-DJTFDBCFILE=<dbc file location>-Dframework.Logging.system.filename=<log
file> -Dservice.Logging.common.filename=<logfile> oracle.apps.iby.ecapp.EcConfig
list
```

Loading Risky Instruments

The Risky Instruments upload utility is a Java application used to store risky payment instruments. It is called RiskyInstrUtil.

Requirements

- Java executable in your application environment
- Oracle Applications Java class Library in the CLASSPATH. The Oracle Applications Java class Library is included in the classpath after you set up the applications environment.

Java Commands

```
java-DJTFDBCFILE=<dbc file location> -Dframework.Logging.system.filename=<log
file> -Dservice.Logging.common.filename=<logfile>
oracle.apps.iby.irisk.admin.RiskyInstrUtil [ADD/DELETE] [filename]
```

This command requires an operation and a filename. It modifies the risky instruments table in the database depending on the entries in the file.

Or

```
java-DJTFDBCFILE=<dbc file location>-Dframework.Logging.system.filename=<log
file> -Dservice.Logging.common.filename=<logfile>
oracle.apps.iby.irisk.admin.RiskyInstrUtil DELETE all
```

This command deletes all the risky instruments in the table.

File Format

- Each line corresponds to one risky instrument.
- The fields are comma separated and are in the following order: Payee identifier, instrument type, and creditcard number. Instrument type has to be a CREDITCARD. For example:

  payee1, CREDITCARD, 4500234023453345

- For the add operation, each risky instrument in the file, that has a valid payee identifier, instrument type, and a new credit card number, is added to the table.
- For the delete operation, each risky instrument that matches the payee identifier, instrument type, and the credit card fields, is deleted from the table.
- The command prints the results of the operation on each risky instrument in the file.

Enabling the XML Framework

iPayment incorporates a XML framework allowing it to communicate with BEPs using XML. Enabling this framework is mandatory and requires the following steps:

- Oracle's XML parsing libraries (xmlparserv2.jar and sax2.zip) must be in iPayment's CLASSPATH. Please check the relevant properties files for the Jserv instance iPayment is running on. By default, both libraries are included in the Jserv configuration of Oracle's Internet Application Server (IAS).
- The IBY: XML_BASE property (and, optionally, the IBY: JAVA_XML_LOG property) must have correct values. See 'iPayment Properties' in the *Oracle iPayment Concepts and Procedures Guide* for a description of both properties.

Setting up Entities in the Oracle iPayment User Interface

To set up Oracle iPayment user interface, see the *Oracle iPayment Concepts and Procedures Guide.*

3

Using iPayment with External Front End Applications

This chapter explains the public APIs used in Oracle iPayment. Topics include:

- Overview of Oracle iPayment APIs
- Implementing Electronic Commerce Applications APIs
- Security Considerations

Overview of Oracle iPayment APIs

Oracle iPayment provides APIs which can be implemented.

- Implementing Electronic Commerce Applications APIs: these APIs are mainly used for payment processing.

Implementing Electronic Commerce Applications APIs

Oracle iPayment provides various types of APIs to integrate electronic commerce applications with Oracle iPayment.

If you are using an electronic commerce application other than the preintegrated Oracle applications, you must implement the electronic commerce application's APIs to link your application to iPayment.

Electronic commerce applications can embed the Oracle iPayment functionality within their application, which eliminates the need to access Oracle iPayment as a standalone application, and hence improves performance, and simplifies the setup.

This section describes the various APIs that are provided to the electronic commerce applications for using the features of Oracle iPayment. The APIs were categorized into these categories:

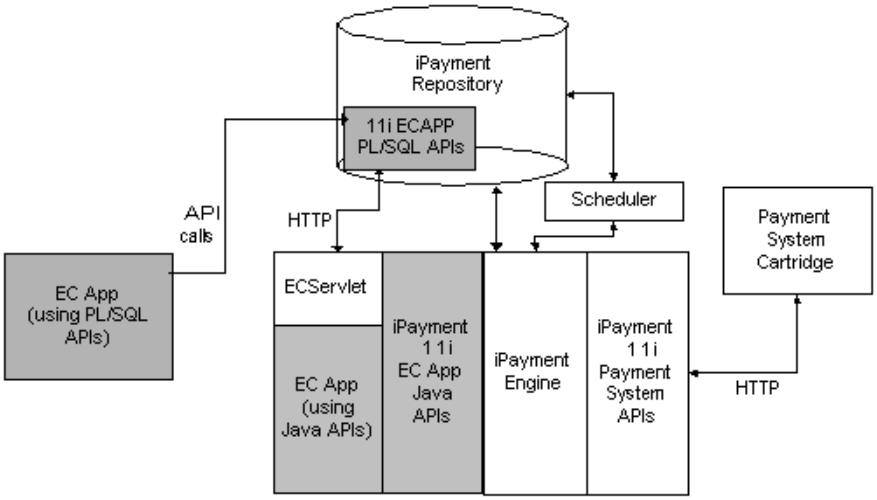
- Payment Instrument Registration APIs
- Payment Processing APIs
- Risk Management APIs
- Credit Card Validation APIs
- Status Update API

Oracle iPayment provides APIs in these programming languages:

- Java APIs for Electronic Commerce Application
- PL/SQL APIs for Electronic Commerce Applications

This diagram shows the integration of APIs with Oracle iPayment.

Figure 3–1 Oracle iPayment integrating with APIs



Payment Instrument Registration APIs

Payment Instrument APIs provide the functionality to register a payor's bank, credit card, PINless debit card, or purchase card.

OraInstrAdd

This API is provided to register a user's bank, credit card, PINless debit card, or purchase card account information with Oracle iPayment. Oracle iPayment generates a PmtInstId if this registration is successful. This identifier is used for payment transactions or for deleting, modifying, or inquiring about this account. Instrument number (credit card number, PINless debit card, purchase card number, or bank account number) and payor identifier together have to be unique.

OraInstrMod

This API is provided to modify registered payment instrument account information with Oracle iPayment.

OraInstrDel

This API is provided to delete registered payment instrument account information.

OraInstrInq

There are two inquiry APIs. One queries instrument information for a single given instrument. The other queries all registered payment instruments for a given payor. The result may contain a mix of credit cards, PINless debit cards, purchase cards, or bank accounts.

Payment Processing APIs

These APIs are the transactional APIs that support various payment operations. The electronic commerce applications use these APIs to process various transaction types. For example, authorization of credit cards, PINless debit cards, and purchase cards, transfer of funds from one bank account to another, capture, cancel, return, and others. A list of such APIs are provided below.

OraPmtReq

This API supports authorization and authorization with capture for credit card, PINless debit card, and purchase card payments. This API also supports inbound account transfers and electronic funds transfer online validation.

When an electronic commerce application is ready to invoke a payment request (possibly due to a user action), it calls this API. If the operation is successful, a transaction identifier is generated by Oracle iPayment and is returned as part of the result. This transaction identifier can be used later by the electronic commerce application to initiate any other operation on a payment.

For example, to modify a payment or capture a payment, the electronic commerce application sends this identifier with other information that is needed to perform the operation requested.

If a payment is either a credit card, PINless debit card, or a purchase card payment, and the request is online, Oracle iPayment can perform risk analysis with the payment request (Authorization).

To enable risk analysis with authorization, either setup the payment request with risk flag set to true in one of its input objects (Refer to Java Documentation for details) or check the Enabled radio button in the Risk Management Status screen for that payee. If either of the conditions are satisfied, the electronic commerce application will check the Riskresp object that is returned as part of the payment response object to the Payment Request API. Electronic commerce applications can also invoke the Payment Request API to evaluate a specific formula by passing the PaymentRiskInfo object.

This API is also used after a voice authorization is done to enable Oracle iPayment to handle follow-on operations. To use it for a voice authorization, set up the payment request's input objects with the Voice Authorization flag set to true and the Authorization Code variable set to the authorization code issued by the financial institution. See *Oracle iPayment Java Documentation* for details.

OraPmtCanc

A scheduled payment can be canceled by an electronic commerce application using this API.

OraPmtQryTrxn

This API provides interface for inquiring the status or history of a payment to electronic commerce application. If a payment has been scheduled and the payment system supports an inquiry operation, the latest status is obtained from the payment system. Otherwise it sends the latest status of the payment as it is in Oracle iPayment. History of a payment can also be obtained.

OraPmtCapture

When a credit card or purchase card is used as part of a payment request and only an authorization is requested, the electronic commerce application has to capture the payment at a later time. The following APIs allow the electronic commerce application to capture all such payments.

OraPmtReturn

This API is used for credit card, PINless debit card, and purchase card specific operations. It allows processing returns from the payor.

Gateway model payment systems process capture operations online. If the capture is still in the Gateway's open batch (that is, the batch has not been closed) you should call OraPmtVoid. If the batch has been closed, you need to call OraPmtReturn. The batch needs to be closed again before the return is processed. This can be confusing since Gateways can be set up to close batches automatically, for example, once per day.

Processor model payment systems process captures offline. If the capture is still in iPayment's open batch, call OraPmtVoid. If the batch has been closed, call OraPmtReturn. The batch needs to be closed again after the return operation for the return to be processed.

OraPmtInq

This API retrieves the payment related information that was sent at the time of a payment request (OraPmtReq API). This information includes payment instrument, payee, tangible id (bill or order), and payor. If the electronic commerce application does not store the payment information, then this is a useful API to support modification of payment requests. It can retrieve the payment information and display it to the end user for modification.

OraPmtVoid

This API allows electronic commerce application to void operations submitted earlier. OraPmtVoid API is supported only to void certain credit card, and purchase card operations. Oracle iPayment supports both online and offline OraPmtVoid API calls.

Voiding auths electronically is not supported by some processors or gateways. Only a few card-issuing banks supported it while the vast majority did not. Cancelling an authorization could only be done manually (by phone) or by letting the auth expire.

Thus, within iPayment, calling OraPmtVoid for an Online Auth results in the current payment system servlets returning status 8 - Operation not Supported. For an Offline Auth, you can void the Authorization if it is still in the iPayment open batch and has not yet been sent to the payment system.

OraPmtCredit

This API provides credit and Electronic Fund Transfer (EFT) operations. Electronic commerce applications can use this API to give stand-alone credit to the customer. If the operation is successful, a transaction identifier is generated by Oracle iPayment. This Identifier is used later to initiate any other operation on the payment. For example, to cancel the credit, electronic commerce application sends this identifier with other information that is needed to perform the cancellation.

OraPmtCloseBatch

The Close Batch API allows a payee or an electronic commerce application to close a batch of previously performed credit card, or purchase card transactions and if necessary PINless debit card. The transaction types that are included in a batch are: capture, return, and credit. This operation is mandatory for a terminal-based merchant.

A host-based merchant may not have to explicitly close the batch because the batch is generally closed at predetermined intervals automatically by the processor. An electronic commerce application has to get this information from its merchant’s acquirer.

OraPmtQueryBatch

This API provides an interface to the electronic commerce application to query the status of an existing batch and a closed batch.

Risk Management APIs

These APIs allow electronic commerce applications to do risk analysis independently for credit card, PINless debit card, and purchase card transactions. These APIs together can evaluate any risk formula that is configured for a payee.

A risk formula can contain any number of risk factors with different weights associated with them. When Risk API 1 is called, it evaluates all the factors configured in the formula except the AVS Code risk factor. If a risk formula has an AVS Code risk factor, then, Risk API 1, in the response object, indicates that the formula has an AVS Code risk factor. This allows electronic commerce applications to completely or partially check the risk formula and decide whether to perform an authorization or not.

If the response of the first Risk API 1 indicates that the payment is not risky, then electronic commerce application can perform the authorization and complete the rest of the evaluation by calling Risk API 2.

Electronic commerce applications can call Risk API 2 by passing the same payee id, the formula name, and the AVS code that was returned during the authorization response and the risk score that was returned as part of the response in Risk API 1. The response object of Risk API 2 contains the finally evaluated risk score.

Risk API 1

This API evaluates the risk formula associated with the payee id passed as part of the input object, PmtRiskInfo. This API can evaluate a specific formula or the implicit formula depending on the input object. After evaluation, this API constructs the response object indicating if the AVS Code risk factor is a part of the formula or not by setting the flag, AVSCodeFlag. If this flag is set to true, then electronic commerce applications need to call the Risk API 2 to complete the risk evaluation of the formula.

Risk API 2

This API needs to be called when the AVSCodeFlag in RiskAPI 1 response object indicates that the formula contains AVS Code factor. When this API is called, it only evaluates the AVS code factor. The input object of this API contains the same payee id and the formula name that was passed in Risk API 1 and the AVS Code that was returned by the payment system for the payment request. The response object that this API returns, contains the final risk score of the formula.

Credit Card Validation APIs

The Credit Card Validation APIs provide methods for determining the credit card type of a credit card number and for doing basic authentication. Since most credit card types specify the number of digits and a prefix for all valid credit card accounts in their company name, it is possible to determine the credit card types of most credit card numbers. Also, since the digits of most credit card types must (using a special algorithm) be evenly divisible by 10, it is possible to determine if a credit card number is valid or not. These APIs do not perform some of the more advanced credit card verification techniques available to back end payment systems, such as billing address verification. These APIs allow many common errors to be caught, such as wrongly typed or truncated credit card digits. By allowing common errors to be caught by the electronic commerce application, performance is improved, since the cost of calling these APIs is much less than sending a request to the back end payment system.

The Credit Card Validation APIs are created as part of the `IBY_CC_VALIDATE` package and this package is installed in the APPS schema.

Main Methods of Credit Card Validation APIs

The Credit Card Validation APIs consist of three main methods.

1. Method `StripCC` is used to format a raw credit card number input by the customer. `StripCC` removes common filler characters such as hyphens and spaces until it produces a credit card number consisting only of digits. `StripCC` must be called before the credit card number is passed to the other methods.
2. Method `GetCCType` returns the credit card type of a credit card number, where each credit card type, including values for invalid and unknown types is a constant in the package.
3. Method `ValidateCC`, which takes both a credit card number and date. It returns a boolean value indicating whether the credit card can still be used or not.

> **Note:** The IN parameters `p_api_version` and `p_init_msg_list` and the OUT parameters `x_msg_count` and `x_msg_data` are ignored. If an unexpected error occurs, `x_return_status` will be set to `FND_API.G_RET_STS_UNEXP_ERROR`. This will happen if the credit card number has invalid characters in it.

```
DECLARE
-- each character specifies a possible filler characters in the credit
```

```
-- card number; i.e. a character that can safely be stripped away
p_fill_chars VARCHAR(3) := '* -#';
p_cc_number VARCHAR(20) := '4111*1111 1111-1111#';
p_api_version NUMBER := 1.0;
p_init_msg_list VARCHAR2(2000) := ' ';
x_return_status VARCHAR2(2000);
x_msg_count NUMBER;
x_msg_data VARCHAR2(2000);
-- will hold the credit card number stripped of all characters except
-- digits; credit card numbers must be of this form for the GetCCType
-- and ValidateCC methods
v_clean_cc VARCHAR(20);
-- variable to be set by GetCCType method
v_cc_type IBY_CC_VALIDATE.CCType;
-- variable set by ValidateCC method; indicates if the credit card is
-- still usable
v_cc_valid BOOLEAN;

-- credit card expr date; rolled to the end of the month
-- by the ValidateCC method
v_expr_date DATE := SYSDATE();
BEGIN
-- the credit card number must first be stripped of all non-digits!!
IBY_CC_VALIDATE.StripCC( p_api_version, p_init_msg_list, p_cc_number,
p_fill_chars, x_return_status, x_msg_count, x_msg_data,
v_clean_cc );
-- check that illegal characters were not found
IF x_return_status != FND_API.G_RET_STS_UNEXP_ERROR THEN
  IBY_CC_VALIDATE.GetCCType( p_api_version, p_init_msg_list, v_clean_cc,
x_return_status, x_msg_count, x_msg_data, v_cc_type);
 IF x_return_status != FND_API.G_RET_STS_UNEXP_ERROR THEN
   IF v_cc_type=IBY_CC_VALIDATE.c_InvalidCC THEN
DBMS_OUTPUT.PUT_LINE('Credit card number not a valid one.');
   ELSE
DBMS_OUTPUT.PUT_LINE('Credit card number OK.');
   END IF;
   IBY_CC_VALIDATE.ValidateCC( p_api_version, p_init_msg_list, v_clean_cc,
v_expr_date, x_return_status, x_msg_count, x_msg_data, v_cc_valid);
   IF v_cc_valid THEN
DBMS_OUTPUT.PUT_LINE('Credit card is valid.');
   ELSE
DBMS_OUTPUT.PUT_LINE('Credit card number invalid or has expired.');
   END IF;
END IF;
END;
```

Note: An overloaded version of the `StripCC` method exists. It takes all the same arguments as the version used above except `p_fill_chars`. It gets its filler characters from the package constant `c_FillerChars`, which allows spaces and hyphens to be interspersed within the credit card number.

Status Update API

Oracle iPayment has defined a PL/SQL API that must be implemented by electronic commerce applications when offline payment processing is performed. This API allows the electronic commerce application to receive a status update. This API must be defined in a package. The naming convention of the package and signature of the API are defined below. Electronic commerce applications must implement the package according to the syntax defined and create the package in the APPS schema if they have offline payments.

The package name has to be of the format <application_short_name>_ecapp_pkg. The application_short_name is a three-letter short name that was given in electronic commerce application registration. The package should have defined update_status procedure with the following signature:

```
PROCEDURE UPDATE_STATUS(
totalRows                IN        NUMBER,
txn_id_Tab               IN        APPS.JTF_VARCHAR2_TABLE_100,
req_type_Tab             IN        APPS.JTF_VARCHAR2_TABLE_100,
Status_Tab               IN        APPS.JTF_NUMBER_TABLE,
updatedt_Tab             IN        APPS.JTF_DATE_TABLE,
refcode_Tab              IN        APPS.JTF_VARCHAR2_TABLE_100,
o_status                 OUT       VARCHAR2,
o_errcode                OUT       VARCHAR2,
o_errmsg                 OUT       VARCHAR2,
o_statusindiv_Tab IN OUT APPS.JTF_VARCHAR2_TABLE_100);
```

The following list describes the field names in the above signature:

1. **totalRows**: total number of rows being passed for the update.
2. **txn_id_Tab:** table of transaction identifiers for which the update is sent.
3. **req_type_Tab:** table of request types corresponding to the Transaction Identifier. For each transaction, there might be a req_type associated with it and the electronic commerce application has to update the correct transaction, based on txn_id and req_type. The reason for having a req-type is to uniquely identify the transaction. For the same transaction identifiers, there can be multiple transactions. e.g. Authorization and Capture. Electronic commerce applications can uniquely identify the transaction based on the values in trxnid and req_type.

   This table lists the various kinds of request types and their descriptions.

req_type	Description
ORAPMTCAPTURE	Capture transaction
ORAPMTCREDIT	Credit transaction
ORAPMTREQ	Authorize transaction
ORAPMTRETURN	Return transaction
ORAPMTVOID	Void transaction

4. **Status_Tab:** table of statuses corresponding to each transaction.

   This table lists the various values and their statuses.

Value	Status
0	Paid
5	Payment failed
13	Scheduled
15	Failed
17	Unpaid
18	Submitted

> **Note:** Please refer to Table D-15 for a complete list of values and their statuses.

5. **updatedt_Tab:** table for the last update date for each transaction.
6. **refcode_Tab:** table for the reference code for each transaction.
7. **o_status:** the overall status of the procedure. If there are errors in trying to execute the procedure, electronic commerce application should set up an appropriate value in this field.
8. **o_errcode:** the error code for any errors which might have occurred during processing.
9. **o_errmsg:** the error message for the error.
10. **o_statusindiv_Tab:** table of status values which have been updated. If the status value has been updated by the electronic commerce application for a particular transaction, it

should set the value to TRUE for that transaction, otherwise, it should set the value to FALSE.

Note: In the above procedure, for each transaction there will be an entry in the table parameters. If there were ten transactions of this electronic commerce application, whose status has changed, there will be ten entries in each table parameters.

When Does the Scheduler Invoke the API?

The Scheduler picks up all the offline payment transactions to be scheduled every time it is run. After all the offline payment transactions are processed either successfully or unsuccessfully, the Scheduler has to update the status changes, if any, of each transaction, to the appropriate electronic commerce application. To update the electronic commerce application, the Scheduler calls the PL/SQL API, which is implemented by that electronic commerce application.

Pseudo Code for Implementing the PL/SQL API by Electronic Commerce Application

For each row update, the status is based on the request type and the transaction identifier. If the update is successful, then set up the status value appropriately.

```
for i in 1..totalRows
;update the tables with status,  updatedate, and refinfo information
update tables using status_Tab[i],  updatedt_Tab[i], refCode_Tab[i] for
   the transaction with id txn_id_Tab[i] and req_type_tab[i]
if update is successful
    o_statusindiv_Tab[i] := 'TRUE'
else
    o_statusindiv_Tab[i] := 'FALSE'
end for;
return
```

Java APIs for Electronic Commerce Application

All administration and inbound payment processing functionalities are provided via the Java PaymentService interface. The following information describes how to access and use Java APIs. Refer to Oracle iPayment JavaDoc for more details.

> **Note:** Guest user properties need to be setup in the database before any operation can be performed. Please refer to the Setup Document provided by CRM Foundation for more details.

Obtaining /Releasing the Payment Service Handle

The OraPmt class offers convenient ways to obtain Payment Service handle (PaymentService) for the user. The application can call various APIs using this handle.

- To obtain the payment service handle, use the following method:

  ```
  static public PaymentService init() throws PSException
  ```

  This API provides Payment Service handle to the user and takes care of all the necessary session initialization steps.

- To release a Payment Service handle with the session, use the following method:

  ```
  static public void end() throws PSException
  ```

Sample code

The following code gives an example of how these APIs are used.

```
public static void main(String[] args) {
try {
     PaymentService paymentService =  OraPmt.init();
     // now you can call all kinds of APIs
     //PSResult result = paymentService.OraPmtReq(...);

} catch (PSException pe) {
     // exception handling
     System.out.println("Error code is: " + pe.getCode());
     System.out.println("Error message is: " + pe.getMessage());
     }

finally {
              try {
```

```
                    OraPmt.end();
                } catch (PSException pe) {
                    // exception handling
                    System.out.println("Error code is: " + pe.getCode());
                    System.out.println("Error message is: " +
                                        pe.getMessage());
                }
            }
    }
```

Checking Returned Result from Payment Service API

PSResult is the returned object of all PaymentService APIs. To obtain the status of the operation, use the following API:

```
public String getStatus();
```

```
This API returns one of the following constants:
PSResult.IBY_SUCCESS// action succeeded
PSResult.IBY_WARNING// action succeeded with warning
PSResult.IBY_INFO// not yet in use
PSResult.IBY_FAILURE// action failed
```

If SUCCESS or WARNING is invoked, a result object can always be obtained by using the following API:

```
 public Object getResult();
```

If FAILURE is invoked, a result object may be returned for payment operation APIs, if this failure occurred with back- end payment system.

The actual object returned varies with each API. It could be an integer or one of the payment response objects. You need to clearly cast it. For a list of castings, refer to the Oracle iPayment Java Documentation for the PaymentService interface.

If WARNING or FAILURE is invoked, a warning or error message is returned. Use the following two APIs to retrieve error codes and error messages.

```
public String getCode();// get the error code 'IBY_XXXXXX'
public String getMessage(); // get the error message text
```

The following sample code illustrates the behavior of PSResult object.

```
public Object checkResult(PSResult pr) {
     String status = pr.getStatus();
     if (status.equals(PSResult.IBY_FAILURE)) {
```

```
        // in case of failure, only error message is expected
        System.out.println("error code is : " + pr.getCode());
        System.out.println("error message is : " + pr.getMessage());
        Object res=pr.getResult();
        if (res!=null) System.out.printIn ("failure occured with backend
Payment system");
        return res;
   }

   if (status.equals(PSResult.IBY_SUCCESS)) {
      // in case of success, only result object is expected
      Object res = pr.getResult();
      return res;  // you need cast this to specific object
      // based on the APIs you called
  }

   if (status.equals(PSResult.IBY_WARNING)) {
       // in case of warning, both result object and message are
       // expected
       // warning is returned only for Payment APIs in case of
       // offline scheduling
       System.out.println("warning code is : " + pr.getCode());
       System.out.println("warning message is : " + pr.getMessage());
       Object res = pr.getResult();
       return res; // you need cast it here too
   }

   // currently IBY_INFO is not yet returned by any PaymentService API
   System.out.println("Illegal status VALUE in PSResult! " +
   pr.getStatus());
   return null;
}
```

Using Payment Service API

After a payment service handle is obtained via the OraPmt class, you can call any of the following APIs in Payment Service interface. For details, refer to JavaDoc.

Here is some sample codes for the Payment Instrument API, and Payment Processing APIs. These codes use the checkResult call.

Registering a Credit Card

```
public void instrAPISample(PaymentService paymentService,
                           int ecappId)    {
    PSResult pr;
```

```
     Object obj;
     CreditCard cc;
     Address addr;
     int instrid_cc;
     String payerid = "payer1";

     addr = new Address("Line1", "Line2", "Line3", "Redwood Shores",
                        "San Mateo", "CA", "US", "94065");

     // credit card
     cc = new CreditCard();
     cc.setName("My Credit Card");
     cc.setFIName("CitiBank");
     cc.setInstrBuf("This is my credit card description.");
     cc.setInstrNum("4111111111111111"); // the credit card number
     cc.setCardType(Constants.CCTYPE_VISA); // the credit card type, should
     // match the credit card number, if set
     cc.setExpDate(new java.sql.Date(101, 0, 10)); // Jan 10, 2001
     cc.setHolderName("Mary Smith");
     cc.setHolderAddress(addr);

     // add the credit card
     pr = paymentService.oraInstrAdd(ecappId, payerid, cc);
     obj = checkResult(pr);
     if (obj ==  null) return; // registration failure
     instrid_cc = ((Integer) obj).intValue();

     System.out.println("Credit card registered successfully " +
                        "with instrument id " + instrid_cc);
}
```

Sending a Credit Card Authorization Request

```
// perform an ONLINE credit card authorization with payment service
public void paymentAPISample(PaymentService paymentService, int ecAppId) {
     Bill t;
     CoreCreditCardReq reqTrxn;
     CreditCard cc;
     PSResult pr;
     CoreCreditCardAuthResp resp;

     // set up the tangible object
     t = new Bill();
     t.setId("orderId1");
```

```
        t.setAmount(new Double(21.00));
        t.setCurrency("USD");
        t.setRefInfo("refInfo");
        t.setMemo("memo");
        t.setUserAccount("userAcct");

        // set up the transaction object
        reqTrxn = new CoreCreditCardReq();
        reqTrxn.setNLSLang("American_America.US7ASCII");
        reqTrxn.setMode(Transaction.ONLINE);
        reqTrxn.setSchedDate(new java.sql.Date(100, 5, 10)); //June 10, 2000
        reqTrxn.setAuthType(Constants.AUTHTYPE_AUTHONLY);

        // set up the payment instrument
        cc = new CreditCard();
        cc.setId(100);  // assuming we have previously registered credit
                        // card with instrument id 100

        pr = // assuming payee1 has already been configured with the payment
             // service
             paymentService.oraPmtReq(ecAppId, "payee1", "", cc, t,
             reqTrxn);

        resp = (CoreCreditCardAuthResp) checkResult(pr);
        if (resp == null) return;
        System.out.println("Request finished with transaction id: " +
        resp.getTID());

}
```

Registering a Purchase Card

```
        public void instrAPISample(PaymentService paymentService,
                                   int ecappId)    {
          PSResult pr;
          Object obj;
          PurchaseCard pc;
          Address addr;
          int instrid_pc;
          String payerid = "payer1";

          addr = new Address("Line1", "Line2", "Line3",
                             "Redwood Shores", "San Mateo", "CA",
                             "US", "94065");
```

```
  // purchase card
  pc = new PurchaseCard();
  pc.setName("My Purchase Card");
  pc.setFIName("CitiBank");
  pc.setInstrBuf("This is my purchase card description.");
  pc.setInstrNum("4111111111111111"); // the purchase card
                                      // number
  pc.setCardType("Constants.CCTYPE_VISA"); // the purchase
  // card type, should match the purchase card number, if
  // set
  pc.setCardSubtype("P");
  pc.setExpDate(new java.sql.Date(101, 0, 10));
                                // Jan 10, 2001
  pc.setHolderName("Mary Smith");
  pc.setHolderAddress(addr);

  // add the purchase card
  pr = paymentService.oraInstrAdd(ecappId, payerid, pc);
  obj = checkResult(pr);
  if (obj == null) return; // registration failure
  instrid_pc = ((Integer) obj).intValue();

  System.out.println("Purchase Card registered " +
                     "successfully with instrument id " +
                     instrid_pc);
}
```

Sending a Purchase Card Authorization Request

```
// perform an ONLINE purchase card authorization with
          // payment service
          public void paymentAPISample(PaymentService paymentService,
                                       int ecAppId)   {
            Bill t;
            PurchaseCardReq reqTrxn;
            PurchaseCard pc;
            PSResult pr;
            CoreCreditCardAuthResp resp; // since purchase card
            // authorization responses are identical to credit card
            // responses. See javadoc for details.

            // set up the tangible object
            t = new Bill();
            t.setId("orderId1");
            t.setAmount(new Double(21.00));
```

```
  t.setCurrency("USD");
  t.setRefInfo("refInfo");
  t.setMemo("memo");
  t.setUserAccount("userAcct");

  // set up the transaction object
  reqTrxn = new PurchaseCardReq();
  reqTrxn.setNLSLang("American_America.US7ASCII");
  reqTrxn.setMode(Transaction.ONLINE);
  reqTrxn.setSchedDate(new java.sql.Date(100, 5, 10));
                                      // June 10, 2000
  reqTrxn.setAuthType(Constants.AUTHTYPE_AUTHONLY);
  reqTrxn.setPONum("PONum");
  reqTrxn.setTaxAmount("1.50");
  reqTrxn.setShipToZip("94065");
  reqTrxn.setShipFromZip("94404");

  // set up the payment instrument
  pc = new PurchaseCard();
  pc.setId(100); // assuming we have previously registered
                 // purchase card with instrument id 100

  pr = // assuming payee1 has already been configured with
       // the payment service
       paymentService.oraPmtReq(ecAppId, "payee1", "", pc,
                                t, reqTrxn);

  resp = (CoreCreditCardAuthResp) checkResult(pr);
  if (resp == null) return;
  System.out.println("Request finished with " +
                     "transaction id: " + resp.getTID());
}
```

PL/SQL APIs for Electronic Commerce Applications

Oracle iPayment provides PL/SQL APIs to those electronic commerce applications that require or prefer PL/SQL interfaces for processing payment operations. There is an additional HTTP call when PL/SQL APIs are called. When electronic commerce applications invoke these PL/SQL APIs, the APIs in return, call the electronic commerce servlet through HTTP.

Oracle iPayment PL/SQL APIs provide all payment related processing and two Risk APIs. The functionality of these APIs is the same as the Java APIs.

PL/SQL APIs are created as part of IBY_PAYMENT_ADAPTER_PUB package and these packages are installed in the APPS schema.

Requirements

1. PL/SQL Package **IBY_PAYMENT_ADAPTER_PUB** must be installed in the APPS schema.
2. An administrator must set up Oracle iPayment URL property to Oracle iPayment electronic commerce servlet's URL using the iPayment administration user interface before invoking the APIs.

The following PL/SQL code helps you to understand how Oracle iPayment PL/SQL APIs can be invoked. This example code invokes the Payment Request API using a credit card. It also passes risk related information for risk evaluation.

```
DECLARE
   p_api_version      NUMBER := 1.0;

   --To initialize message list.
   p_init_msg_list VARCHAR2(2000)  := FND_API.G_TRUE;
   p_commit               VARCHAR2(2000)  := FND_API.G_FALSE;
   p_validation_level   NUMBER  := FND_API.G_VALID_LEVEL_FULL;
   p_ecapp_id             NUMBER := 0;
   p_payee_rec            IBY_PAYMENT_ADAPTER_PUB.Payee_rec_type;
   p_payer_rec            IBY_PAYMENT_ADAPTER_PUB.Payer_rec_type;
   p_pmtinstr_rec   IBY_PAYMENT_ADAPTER_PUB.PmtInstr_rec_type;
   p_tangible_rec        IBY_PAYMENT_ADAPTER_PUB.Tangible_rec_type;
   p_pmtreqtrxn_rec   IBY_PAYMENT_ADAPTER_PUB.PmtReqTrxn_rec_type;
   p_riskinfo_rec        IBY_PAYMENT_ADAPTER_PUB.RiskInfo_rec_type;
   x_return_status      VARCHAR2(2000);
                                       -- output/return status
   x_msg_count         NUMBER;
                                       -- output message count
```

```
   x_msg_data           VARCHAR2(2000);
                                        -- reference string for getting output
message text
   x_reqresp_rec       IBY_PAYMENT_ADAPTER_PUB.ReqResp_rec_type;
                                        -- request specific output response
object
   l_msg_count         NUMBER;
   l_msg_data           VARCHAR2(2000);

BEGIN
  p_ecapp_id := 66;        -- iPayment generated ECAppID
  p_payee_rec.Payee_ID := 'ipay-payee1';    -- payee's ID
  p_payer_rec.Payer_ID   := 'ipay-cust1';      -- payer's ID
  p_payer_rec.Payer_Name  := 'Cust1';         -- Payer's (Customer's name)
  p_pmtreqtrxn_rec.PmtMode := 'ONLINE';
                                                  -- Payment mode (Can be
ONLINE/OFFLINE)
  p_tangible_rec.Tangible_ID  := 'tangible_id1';  -- Tangible ID / order ID
  p_tangible_rec.Tangible_Amount := 25.50; -- Amount for the transaction
  p_tangible_rec.Currency_code := 'USD';   -- Currency for the transaction
  p_tangible_rec.RefInfo := 'test_refinfo3';
  p_pmtreqtrxn_rec.Auth_Type := upper('authonly');    -- request type
  p_pmtinstr_rec.CreditCardInstr.CC_Type := 'Visa';
                                                                          --
payment instrument type
  p_pmtinstr_rec.CreditCardInstr.CC_Num := '4111111111111111';
                                                                        --
payment instrument number
  p_pmtinstr_rec.CreditCardInstr.CC_ExpDate := to_char(sysdate+300);
                                                                        --
payment instr. Expiration date

 --5. RISK INPUTS
  p_riskinfo_rec.Formula_Name := 'test3';     -- Risk formula name
  p_riskinfo_rec.ShipToBillTo_Flag := 'TRUE';
                                -- Flag showing if ship to address same as Bill
to address
  p_riskinfo_rec.Time_Of_Purchase := '08:45';
                                 -- Time of purchase

  IBY_PAYMENT_ADAPTER_PUB.OraPmtReq
( p_api_version,
p_init_msg_list,
p_commit,
p_validation_level,
```

```
                p_ecapp_id ,
                p_payee_rec,
                p_payer_rec,
                p_pmtinstr_rec,
                p_tangible_rec,
                p_pmtreqtrxn_rec,
                p_riskinfo_rec ,
                x_return_status,
                x_msg_count ,
                x_msg_data  ,
                x_reqresp_rec);
END;
Payment Request Related Response. Printing Only If Status Is Success
  If(Char(X_Reqresp_Rec.Response.Status = ‘S’) Then

         -- Offline Mode Related Response
        If P_Pmtreqtrxn_Rec.Pmtmode = 'OFFLINE' Then
            Dbms_Output.Put_Line('Transaction ID = ' || To_Char(X_Reqresp_
Rec.Trxn_ID));
            Dbms_Output.Put_Line (‘ X_Reqresp_
Rec.Offlineresp.Earliestsettlement_Date = ' ||
                                                To_Char(X_Reqresp_
Rec.Offlineresp.Earliestsettlement_Date));
            Dbms_Output.Put_Line('X_Reqresp_Rec.Offlineresp.Scheduled_Date = '
||
To_Char(X_Reqresp_Rec.Offlineresp.Scheduled_Date));
        Else
            Dbms_Output.Put_Line('Transaction ID = ' || To_Char(X_Reqresp_
Rec.Trxn_ID));
            Dbms_Output.Put_Line('X_Reqresp_Rec.Authcode = ' || X_Reqresp_
Rec.Authcode);
            Dbms_Output.Put_Line('X_Reqresp_Rec.Avscode = ' || X_Reqresp_
Rec.Avscode);
            Dbms_Output.Put_Line('----------------------------------------');
            -- Risk Related Response
            If(X_Reqresp_Rec.Riskrespincluded = ‘YES’) Then
                  Dbms_Output.Put_
Line('----------------------------------------');
               Dbms_Output.Put_Line(' X_Reqresp_Rec.Riskresponse.Risk_Score=
'|| X_Reqresp_Rec.Riskresponse.Risk_Score );
                Dbms_Output.Put_Line('X_Reqresp_Rec.Riskresponse.Risk_
Threshold_Val=   '||
                                                    X_Reqresp_
Rec.Riskresponse.Risk_Threshold_Val);
            Endif;
```

```
      Endif;
End If;
```

Security Considerations

Oracle iPayment is architected to send credit card details in the URL. This architecture requires the logging levels on Apache to be lowered from the default to prevent the credit card information from appearing in the log files.

In the httpds.conf file, change:

LogFormat "%h %l %u %t \"%r\" %>s %b" common

to:

LogFormat "%h %l %u %t \"%U\" %>s %b" common

4

Using iPayment with External Payment Systems

This appendix explains about integrating Oracle iPayment with external payment systems. The topics covered include:

- Overview of Payment System Integration Model
- PaymentService APIs
- Routing Engine
- Integration Point Component Types
- Developing a Custom Payment System Integration
- Defining a Payment System
- System Payment Profile
- Formats
- Format Validation
- Extract Generator
- Extract Formatter
- Extract Structure
- Extract Components
- Transmission Functions
- Acknowledgment Parser

Overview of Payment System Integration Model

Oracle iPayment provides a complete payment solution. The Payment System Integration Model allows integration with third party payment systems for credit card, purchase card, PINless debit card, and bank account transfer processing. The payment systems communicate with the payment processors and the acquirers/banks to process payment transactions.

Though the business flow for credit card, purchase card, and bank account transfer transactions is the same, the system flow is significantly different, with the addition of many new modules, both internal and external, such as customizable integration points. The payment system integration model has changed from an API-centered model to a payment instruction file creation and delivery model. This model reflects the various customizable integration points exposed for implementing the custom payment system integration for iPayment.

Flow in gateway and processor model systems

For gateway-model payment systems, since every transaction is online and involves real-time communication with the payment system, the above flow occurs for every transaction operation type, such as authorization, capture, credit, etc.

For processor payment systems, the above flow occurs only for online transaction such as authorization, batch close, and batch query operations. For other operation types, such as capture and credit, the transaction is batched without communication to the payment system. Therefore, the flow completes after format validation occurs.

PaymentService APIs

The PaymentService API represents the Java class containing all payment transaction APIs and acts as the entry point into the iPayment engine. Generic validations are performed here.

Routing Engine

The routing engine associates a payment transaction with a payment system and payment system account. Currently, the routing engine assumes that only a single system payment profile exists per payment system and that each processor-type payment system has a system profile defined for each processor.

For implementing custom payment system integrations, a system payment profile must be defined for the new payment system otherwise the payment system will use the old HTTP name-value pair BEP APIs.

Integration Point Component Types

The table lists the various integration point component types that maybe implemented while integrating with a payment system.

Integration Point Type	Description
Format	Creates the payment instruction file based upon transaction data.
Format Validation Set	Format or payment system-specific validations transactions.
Transmission Function	Implements a transmission protocol for communicating with the payment system.
Acknowledgement Parser	Parses the payment system's acknowledgments.
Payment System Definition	Define attributes for the payment system which the user will provide in the setup UI.
System Payment Profile	The payment processing attributes of a payment system's specification.

Note: Several of the integration component types are developed in Java.

Developing a Custom Payment System Integration

The following sections list the tasks for developing a custom payment system integration for the instrument types. The tasks for an instrument type integration must be performed in an order. If a payment system supports multiple instrument types, combine the tasks for each instrument type.

Developing a Custom Payment System Integration for Credit Cards

The table lists the tasks that must be completed for integrating a credit card payment system.

Task	Component Type	Mandatory	Description
Define the Payment System	Payment System Definition	Yes	Defines the new payment system. If the payment system is a gateway and uses the old HTTP name-value pair BEP APIs, you need not complete any furhter tasks from this list but procees to implement the gateway BEP APIs.
Define payment system account options	Payment System Definition	No	Defines the options for all accounts in the payment system.
Develop an online authorization format template	Format	Yes	Develop a template for XML Publisher.
Seed the online authorization format template	Format	Yes	Define system data attributes for the new format.
Develop an online authorization transmission function	Transmission Function	Yes	Implements a transmission function protocol.
Seed the online authorization transmission protocol	Transmission Function	Yes	Defines a transmission protocol and associates it with the transmission function code-point that implements it. Also defines the protocol parameters.
Develop an online authorization acknowledgement parser	Acknowledgement Parser	Yes	Implements an acknowledgement parser.
Seed the online authorization acknowledgement parser	Acknowledgement Parser	Yes	Define system data attributes for the parser.
Develop a settlement format template	Format	Yes	For gateway model payment system, the format may be identical to the authorization format template.

Task	Component Type	Mandatory	Description
Seed the settlement format template	Format	Yes	
Develop a settlement transmission function	Transmission Function	Yes	For gateway model payment systems, the transmission function may be identical to the authorization transmission function.
Seed the settlement transmission protocol	Transmission Function	Yes	
Develop a settlement acknowledgement parser	Acknowledgement Parser	Yes	For gateway model payment systems, the acknowledgement parser may be identical to the authorization parser.
Seed the settlement acknowledgement parser	Acknowledgement Parser	Yes	
Develop a query format template	Format		The query support is optional for gateway model payment system. For most processor payment systems, an acknowledgement request message is not defined.
Seed the query format template	Format		Optional for gateway model payment systems.
Develop a query transmission function	Transmission Function		Optional for gateway model payment systems.
Seed the query transmission protocol	Transmission Function		Optional for gateway model payment systems.
Develop an query acknowledgement parser	Acknowledgement Parser		Optional for gateway payment systems.
Seed the query acknowledgement parser	Acknowledgement Parser		Optional for gateway model payment systems.
Create credit card system payment profile	System Payment Profile	Yes	Defines the payment processing attributes for the payment system's credit card processing specification.
Create transaction validation set	Format Validation Set	No	Required only for payment system-specific validations.

Task	Component Type	Mandatory	Description
Create batch validation set	Format Validation Set	No	Never for gateway model payment system. For processor model payment system, required only for payment system specific validations.
Seed validation set assignments	Format Validation Set	No	Required only for payment system-specific validations.

Developing a Custom Payment System Integration for Debit Cards

The table lists the tasks that must be completed for integrating a debit card payment system.

Task	Component Type	Mandatory	Description
Define the Payment System	Payment System Definition	Yes	Defines the new payment system.
Define payment system account options	Payment System Definition	No	Defines the options for all accounts in the payment system.
Develop an online debit format template	Format	Yes	Develop a template for XML Publisher.
Seed the online debit format template	Format	Yes	Define system data attributes for the new format.
Develop an online debit transmission function	Transmission Function	Yes	Implements a transmission function protocol.
Seed the online debit transmission protocol	Transmission Function	Yes	Defines a transmission protocol and associates it with the transmission function code-point that implements it. Also defines the protocol parameters.
Develop an online debit acknowledgement parser	Acknowledgement Parser	Yes	Implements an acknowledgement parser.
Seed the online debit acknowledgement parser	Acknowledgement Parser	Yes	Define system data attributes for the parser.
Develop a settlement format template	Format	No	Optional for payment system that do not require debit card settlement.
Seed the settlement format template	Format	No	Optional for payment system that do not require debit card settlement.
Develop a settlement transmission function	Transmission Function	No	Optional for payment system that do not require debit card settlement.
Seed the settlement transmission protocol	Transmission Function	No	Optional for payment system that do not require debit card settlement.

Task	Component Type	Mandatory	Description
Develop a settlement acknowledgement parser	Acknowledgement Parser	No	Optional for payment system that do not require debit card settlement.
Seed the settlement acknowledgement parser	Acknowledgement Parser	No	Optional for payment system that do not require debit card settlement.
Develop a query format template	Format	No	Optional for payment system that do not require debit card settlement.
Seed the query format template	Format	No	Optional for payment system that do not require debit card settlement.
Develop a query transmission function	Transmission Function	No	Optional for payment system that do not require debit card settlement.
Seed the query transmission protocol	Transmission Function	No	Optional for payment system that do not require debit card settlement.
Develop an query acknowledgement parser	Acknowledgement Parser	No	Optional for payment system that do not require debit card settlement.
Seed the query acknowledgement parser	Acknowledgement Parser	No	Optional for payment system that do not require debit card settlement.
Create debit card system payment profile	System Payment Profile	Yes	Defines the payment processing attributes for the payment system's credit card processing specification.
Create transaction validation set	Format Validation Set	No	Required only for payment system-specific validations.
Create batch validation set	Format Validation Set	No	Never for gateway model payment system. For processor model payment system, required only for payment system specific validations.
Seed validation set assignments	Format Validation Set	No	Required only for payment system-specific validations.

Developing a Custom Payment System Integration for Bank Account Cards

The table lists the tasks that must be completed for integrating a bank account payment system.

Task	Component Type	Mandatory	Description
Define the Payment System	Payment System Definition	Yes	Defines the new payment system.
Define payment system account options	Payment System Definition	No	Defines the options for all accounts in the payment system.
Develop an online verification format template	Format	No	Only if online verification supported by payment system.
Seed the online verification format template	Format	No	Define system data attributes for the new format.
Develop an online verification transmission function	Transmission Function	No	Implements a transmission function protocol.
Seed the online verification transmission protocol	Transmission Function	No	Defines a transmission protocol and associates it with the transmission function code-point that implements it. Also defines the protocol parameters.
Develop an online verification acknowledgement parser	Acknowledgement Parser	No	Implements an acknowledgement parser.
Seed the online verification acknowledgement parser	Acknowledgement Parser	No	Define system data attributes for the parser.
Develop a funds transfer format template	Format	Yes	

Task	Component Type	Mandatory	Description
Seed the funds transfer format template	Format	Yes	
Develop a funds transfer transmission function	Transmission Function	Yes	
Seed the funds transfer transmission protocol	Transmission Function	Yes	
Develop a funds transfer acknowledgement parser	Acknowledgement Parser	Yes	
Seed the funds transfer acknowledgement parser	Acknowledgement Parser	Yes	
Develop a query format template	Format	Yes	
Seed the query format template	Format	No	Only if payment system supports bank account transfer acknowledgements.
Develop a query transmission function	Transmission Function	No	
Seed the query transmission protocol	Transmission Function	No	
Develop an query acknowledgement parser	Acknowledgement Parser	No	
Seed the query acknowledgement parser	Acknowledgement Parser	No	
Create bank account system payment profile	System Payment Profile	Yes	Defines the payment processing attributes for the payment system's credit card processing specification.
Create transaction validation set	Format Validation Set	No	Required only for payment system-specific validations.

Task	Component Type	Mandatory	Description
Create batch validation set	Format Validation Set	No	Never for gateway model payment system. For processor model payment system, required only for payment system specific validations.
Seed validation set assignments	Format Validation Set	No	Required only for payment system-specific validations.

Seeding Data

The two important points to be remembered while seeding data include:

- Language-specific data

  Some tables are segmented, with non-language specific columns appearing in a table with the _B suffix and language-specific (i.e. translatable) columns appearing in a parallel table with the _TL suffix. When inserting in the translated column segment table you must use the appropriate language/country code for the data being added, for example, "US" for both columns. It will then be necessary to call some sort of _ADD_ LANGUAGE procedure to fill this table out for all the supported languages used during the installation.

- WHO columns

  Every table has a common set of change-tracking (WHO) columns.

The table lists the the specific functions for the WHO columns.

For the column...	Enter the value...
OBJECT_VERSION_NUMBER	1
CREATION_DATE	SYSDATE()
CREATED_BY	FND_GLOBAL.USER_ID
LAST_UPDATE_DATE	SYSDATE()
LAST_UPDATED_BY	FND_GLOBAL.USER_ID
LAST_UPDATE_LOGIN	FND_GLOBAL.LOGIN_ID

Defining a Payment System

The first task in integrating any payment system is defining a payment system. The definition must include both the payment system's attributes as well as the account options, if any, defined for the payment system accounts the users will establish.

Payment System Attributes

The table lists key attributes of the payment system.

Attribute	Description	Constraints
Payment System	The name of the payment system.	
Supported Instruments	The instrument types supported by the payment system.	Must be one of the following instrument types: ■ Credit Card ■ Purchase Card Level II/Level III ■ PINless debit card
Type	Payment system model type. See Understanding Gateway-Model and Processor-Model Payment Systems in the iPayment Concepts and Procedures guide for a discussion of the differences between the processor and gateway payment system models.	Must be either processor or gateway model payment system.
Suffix	Unique 3-letter identifier for the payment system.	Must not be any of these reserved suffixes seeded in iPayment. The reserved suffixes are: ■ cyb (Cybercash) ■ lop (sample gateway system) ■ efs (Concord EFSnet) ■ ptk (Paymentech) ■ fdb (FDC North) ■ cit (Citibank Merchant Services) ■ cep (Citibank Edifact Bank Transfer)

Attribute	Description	Constraints
Servlet Base URL	The URL of the payment system servlet.	Must be a URL in this form: **http://<host>:<post>/<servlet zone>**.

All of these attributes can be set in the Payment System details page of the iPayment administration user interface. See Creating a New Payment System in the iPayment Concepts and Procedures Guide.

Note: For implementing the custom payment system integrations, you must enter appropriate values while creating a payment system in the Payment System details page.

Account Options

Account options are attributes defined by the payment system for its user accounts. Once an account is created in the Payee Account Information page (see Creating a New Payee in the iPayment Concepts and Procedures Guide), account options may be set if defined for the payment system. Account options are used for payment instruction file generation and are represented in the funds capture extract.

Seeding Account Options Definitions

The table lists the attributes of an account option.

Attribute	Description	Constraints
BEPID	The primary key of the payment system to which the account option belongs.	Must refer to an existing payment system in IBY_ BEPINFO.
ACCOUNT_OPTION_ CODE	The unique code for the account option. This attribute will appear in the Name sub-element of the AccountOption element.	Must be unique per payment system and must have 30 characters.
ACCOUNT_OPTION_ DATATYPE	The datatype of the account option to help in UI validation.	Supported values include: ■ VARCHAR2 ■ NUMBER ■ DATE
DISPLAY_ORDER	The display order of the account option when shown in the UI.	
ACCOUNT_OPTION_ NAME	The description of the account option.	

For implementing the custom payment system integrations, if any attributes of the user's payment system account are required as data when generating the payment instruction file, the account option's definition must be seeded for the payment system

Once the set of account option definitions are determined, the attributes can be seeded in iPayment by creating a SQL script to insert them.

Note: Insert all attributes except translatable attribute ACCOUNT_OPTION_NAME into table IBY_BEP_ACCT_OPT_NAME_B. Insert translatable attribute ACCOUNT_OPTION_NAME into IBY_BEP_ACCT_OPT_NAME_TL and then call function IBY_FNDCPT_MLSUTL_PVT. BEP_OPT_ADD_LANGUAGE.

The attribute BEPID must be the primary key of the payment system owning the account option, which is easily determined by this query:

```
SELECT bepid
FROM iby_bepinfo
WHERE (suffix = :BEP_SUFFIX);
```

When you create a payment system account, The payment system account will automatically establish itself in the corresponding account options in the UI.

For implementing the custom payment system integrations, please make sure to document the values.

System Payment Profile

The system payment profile captures the payment processing "meta-data" for a particular payment system specification. The system payment profile defines attributes such as the formats and transmission protocols the payment system requires. For every major instrument type supported by a payment system specification, such as credit card, bank account, and PINless debit card, a separate system payment profile must be created.

Credit Card System Payment Profile

A credit card profile defines the attributes of a credit card processing specification. A credit card system payment profile attributes may be seeded in iPayment by creating a SQL script to insert them.

> **Note:** Insert all attributes except SYS_CC_PROFILE_NAME into table IBY_FNDCPT_SYS_CC_PF_B. Insert SYS_CC_PROFILE_NAME into IBY_FNDCPT_SYS_CC_PF_TL and call procedure IBY_FNDCPT_MLSUTL_PVT.SYS_CC_PROF_ADD_LANGUAGE.

The attribute PAYMENT_SYSTEM_ID must be the primary key of the payment system owning the profile, which is easily determined by this query:

```
SELECT bepid
FROM iby_bepinfo
WHERE (suffix = :BEP_SUFFIX);
```

This table explains the attributes and allowed values for a credit card system payment profile.

Attribute Name	Description	Constraint
SYS_CC_PROFILE_CODE	Unique identifier for the profile.	Must be unique and less than or equal to 30 characters.
SYS_CC_PROFILE_NAME	Name of the system profile.	
PAYMENT_SYSTEM_ID	Identifier of the payment system the profile belongs to.	Equals the primary key of the payment system in IBY_BEPINFO.
ONLINE_AUTH_FORMAT_CODE	Code of the authorization payment instruction format.	Equals an existing format.
ONLINE_AUTH_TRANS_PRTCL_CODE	Code of the authorization transmission protocol.	Equals an existing protocol.
ONLINE_AUTH_ACK_RDR_CODE	Code of the authorization acknowledgement reader.	Equals an existing acknowledgment reader.
SETTLEMENT_FORMAT_CODE	Code of the settlement payment instruction format.	Equals an existing format.

Attribute Name	Description	Constraint
SETTLEMENT_TRANS_ PRTCL_CODE	Code of the settlement transmission protocol.	Equals an existing protocol.
SETTLEMENT_ACK_ RDR_CODE	Code of the authorization acknowledgement reader.	Equals an existing acknowledgment reader.
QUERY_FORMAT_CODE	Code of the settlement payment instruction format.	Equals an existing format or is NULL.
QUERY_TRANS_PRTCL_ CODE	Code of the settlement transmission protocol.	Equals an existing protocol.
QUERY_ACK_RDR_CODE	Code of the authorization acknowledgement reader.	Equals an existing acknowledgment reader.

Debit Card System Payment Profile

A debit card system profile defines the attributes of a debit card processing specification. A debit card system payment profile attributes may be seeded in iPayment by creating a SQL script to insert them.

> **Note:** Insert all attributes except SYS_DC_PROFILE_NAME into table IBY_FNDCPT_SYS_DC_PF_B. Insert SYS_DC_PROFILE_NAME into IBY_FNDCPT_SYS_DC_PF_TL and call procedure IBY_FNDCPT_MLSUTL_PVT.SYS_DC_PROF_ADD_LANGUAGE.

The attribute PAYMENT_SYSTEM_ID must be the primary key of the payment system owning the profile, which is easily determined by this query:

```
SELECT bepid
FROM iby_bepinfo
WHERE (suffix = :BEP_SUFFIX);
```

This table explains the attributes and allowed values for a debit card system payment profile.

Attribute Name	Description	Constraint
SYS_DC_PROFILE_CODE	Unique identifier for the profile.	Must be unique and less than or equal to 30 characters.
SYS_DC_PROFILE_NAME	Name of the system profile.	
PAYMENT_SYSTEM_ID	Identifier of the payment system the profile belongs to.	Equals the primary key of the payment system in IBY_BEPINFO.
ONLINE_DEB_FORMAT_CODE	Code of the debit payment instruction format.	Equals an existing format.
ONLINE_DEB_TRANS_PRTCL_CODE	Code of the debit transmission protocol.	Equals an existing protocol.
ONLINE_DEB_ACK_RDR_CODE	Code of the debit acknowledgement reader.	Equals an existing acknowledgment reader.
SETTLEMENT_FORMAT_CODE	Code of the settlement payment instruction format.	Equals an existing format.
SETTLEMENT_TRANS_PRTCL_CODE	Code of the settlement transmission protocol.	Equals an existing protocol.

Attribute Name	Description	Constraint
SETTLEMENT_ACK_ RDR_CODE	Code of the authorization acknowledgement reader.	Equals an existing acknowledgment reader.
SETTLE_REQ_FLAG	Settlement required flag. Indicates if a settlement transaction is required after a debit in order to complete the funds transfer	Equals one of the following values: ■ Y (Yes) ■ N (No)
QUERY_FORMAT_CODE	Code of the settlement payment instruction format.	Equals an existing format or is NULL.
QUERY_TRANS_PRTCL_ CODE	Code of the settlement transmission protocol.	Equals an existing protocol.
QUERY_ACK_RDR_CODE	Code of the authorization acknowledgement reader.	Equals an existing acknowledgment reader.

Bank Account Payment Profiles

A bank account system profile defines the attributes of a bank account transfer processing specification. A bank account transfer system payment profile attributes may be seeded in iPayment by creating a SQL script to insert them.

> **Note:** Insert all attributes except SYS_EFT_PROFILE_NAME into table IBY_FNDCPT_SYS_EFT_PF_B. Insert SYS_EFT_PROFILE_NAME into IBY_FNDCPT_SYS_EFT_PF_TL and call procedure IBY_FNDCPT_MLSUTL_PVT.SYS_EFT_PROF_ADD_LANGUAGE.

The attribute PAYMENT_SYSTEM_ID must be the primary key of the payment system owning the profile, which is easily determined by this query:

```
SELECT bepid
FROM iby_bepinfo
WHERE (suffix = :BEP_SUFFIX);
```

This table explains the attributes and allowed values for a bank account payment system profile.

Attribute Name	Description	Constraint
SYS_EFT_PROFILE_CODE	Unique identifier for the profile.	Must be unique and less than or equal to 30 characters.
SYS_EFT_PROFILE_NAME	Name of the system profile.	
PAYMENT_SYSTEM_ID	Identifier of the payment system the profile belongs to.	Equals the primary key of the payment system in IBY_BEPINFO.
VERIFY_FORMAT_CODE	Code of the verify payment message format. Note that not all payment systems support bank account verification.	Equals an existing format or is NULL.
VERIFY_TRANS_PRTCL_CODE	Code of the verification transmission protocol.	Equals an existing protocol or is NULL.
VERIFY_ACK_RDR_CODE	Code of the verification acknowledgement reader.	Equals an existing acknowledgment reader or is NULL.

Attribute Name	Description	Constraint
FUNDS_XFER_FORMAT_CODE	Code of the funds transfer payment instruction format.	Equals an existing format.
FUNDS_XFER_TRANS_PRTCL_CODE	Code of the funds transfer transmission protocol.	Equals an existing protocol.
FUNDS_XFER_ACK_RDR_CODE	Code of the funds transfer acknowledgement reader.	Equals an existing acknowledgment reader.
QUERY_FORMAT_CODE	Code of the settlement payment instruction format.	Equals an existing format or is NULL.
QUERY_TRANS_PRTCL_CODE	Code of the settlement transmission protocol.	Equals an existing protocol or is NULL.
QUERY_ACK_RDR_CODE	Code of the authorization acknowledgement reader.	Equals an existing acknowledgment reader or is NULL.

Formats

A format corresponds to a payment instruction file format defined by the payment system. A payment system uses different payment instruction formats during a transaction processing, such as authorization, settlement. Before creating a format in iPayment, a corresponding XML Publisher template entity must be available.

Developing a Format Template

This table describes the XML Publisher template attributes fixed by iPayment:

Attribute	Description	Constraint
Data Definition	The template's data definition determines the structure of the data to which the template is applied.	Always equal to: IBY_FNDCPT_ INSTRUCTION_1_0

For implementing the custom payment system integrations, a payment instruction template must be developed for XML Publisher using one of the supported XML Publisher template types such as eText (RTF), XSL, etc. This template must be provided to the users of your payment system integration along with instructions regarding the values that should be used when manually defining the template in the XML Publisher Templates UI page.

Seeding a Format Template

Once an XML Publisher template is created, the corresponding formats entity must be created in iPayment.

This table explains the attributes for the format template.

Attribute	Description	Constraint
FORMAT_ CODE	The unique identifier of the format	Must be unique and less than or equal to 30 characters.
FORMAT_ TEMPLATE_ CODE	The code of the XML Publisher template to which this formats corresponds.	Must correspond to an existing XDO template.
EXTRACT_ID	The extract used by the format.	Always equal to:100 (for IBY_FUNDS_ CAPTURE_INSTRUCTION extract version 1.0).
FORMAT_ TYPE_CODE	The type of format.	Always equal to:FUNDS_CAPTURE_ INSTRUCTION

Attribute	Description	Constraint
FORMAT_ NAME	The description of the format.	

These attributes may be seeded in iPayment by creating a SQL script to insert them. All attributes except FORMAT_NAME will be inserted into IBY_FORMATS_B.

Note: Insert translatable attribute FORMAT_NAME into IBY_ FORMATS_TL and then call function IBY_FNDCPT_MLSUTL_PVT. FORMAT_ADD_LANGUAGE.

Format Validation

After routing a transaction to a payment system account, the payment system, system payment profile, and payment instruction format used for the transaction are displayed. Each format can have a set of validations associated with it, which is applied to the transaction to determine if the transaction is valid.

Note: All attributes except FORMAT_NAME will be inserted into IBY_FORMATS_B. Insert translatable attribute FORMAT_NAME into IBY_FORMATS_TL and then call function IBY_FNDCPT_MLSUTL_PVT. FORMAT_ADD_LANGUAGE.

For implementing the custom payment system integrations, you can define format validation sets if the payment system enforces more stringent validations than iPayment.

Format validation sets allow payment system-specific validations to be performed on a transaction. This feature is optional and need be implemented only if the generic validations provided by the iPayment engine are insufficient.

In order to use a validation set with a particular format, these tasks must be performed:

- Developing a Validation Set
- Seeding a Validation Set

Developing a Validation Set

Processor-type payment systems define two sets of payment format specifications, one for online transactions, such as authorizations and another for batched transactions, such as settlements, credits. Therefore, two types of validation set code-points are supported: one for individual transaction operations and another for batch operations.

For implementing the custom payment system integrations, the Java class or PL/SQL package implementing the validation set code-point must be distributed to the user who must then make it accessible to their system by either placing the class file in the CLASSPATH of the application server hosting the iPayment engine, or uploading the package to their database instance.

Batch Validation Sets

A batch validation set must be a PL/SQL procedure.

This table explains the signature of a batch validation:

Name	Data Type	Type	Description
p_api_version	NUMBER	IN	Version of the API called; may be ignored
p_init_msg_list	VARCHAR2	IN	Whether to initialize the message list; may be ignored
p_mbatchid	NUMBER	IN	The identifier of the batch (in table IBY_BATCHES_ALL)
x_return_status	VARCHAR2	OUT	Status of the call.
x_msg_count	NUMBER	OUT	Number of error messages on the stack.
x_msg_data	VARCHAR2	OUT	Message stack of errors.

The important parameters for this signature are p_mbatchid and x_return_status. The first indicates the batch which is being validated and is a foreign key to the MBATCHID primary key column of table IBY_BATCHES_ALL. With this primary key the aggregate values of the batch (e.g. amount totals) stored in IBY_BATCHES_ALL may be validated. The transactions included in the batch may be queried from table IBY_TRXN_SUMMARIES_ALL using column MBATCHID as a filter.

If validation for the batch fails parameter x_returns_status should be set to FND_API.G_RET_STS_SUCCESS. An optional message indicating the cause of the validation set failure may be returned in parameters x_msg_count and x_msg_data as well using the FND_

MSG APIs. If a batch fails then the batch will be rolled back from the database and batch close operation execution is halted. No transactions are removed from the batch, however, even if the transaction are the primary cause for why the batch validation failed. The next batch close attempt will fail unless that transaction is voided by the user. Oracle recommends that all transaction validation is done in a transaction validation set.

Transaction Validation Sets

A transaction validation set validates a single transaction immediately after the transaction is routed and as soon as the payment system and payment system formats for the transaction are known. Oracle recommends that all validations for an individual transaction is done in a transaction validation set rather than a batch validation set, even if the transaction, such as a credit transaction, appears as part of a batch.

A transaction validation set is a Java code-point which implements interface: `oracle.apps.iby.payment.FndCptValidationSet`. This interface has a single function.

This table explains the signature defined by the code-point:

Attribute Name	Type	Description
ecappId	Integer	Electronic commerce application id for the current transaction
payee	oracle.apps.iby.ecapp.Payee	Transaction payee
pmtInstr	oracle.apps.iby.ecapp.PmtInstr	Payment instrument used
order	oracle.apps.iby.ecapp.Tangible	Order
trxn	oracle.apps.iby.ecapp.Transaction	The transaction performed
<return>	oracle.apps.iby.engine.ValidationSetResult	Validation results

Note: The payment instrument, order, and transaction objects are sub-classes of types `oracle.apps.iby.ecapp.PmtInstr`, `oracle.apps.iby.ecapp.Tangible`, `oracle.apps.iby.ecapp.Transaction` respectively, and appropriate for the instrument type and operation type of the transaction being performed.

The validation set must return as its result an object of type `oracle.apps.iby.engine.ValidationSetResult`.

The table list the class attributes.

Attribute Name	Type	Description
Valid	boolean	Validation result. If passed then true, else false.
Message	String	An encoded iPayment error message string of the form:FND_MESSAGE_ CODE#TOKEN_NAME1=TOKEN_ VAL1#.
Code	String	Validation error code, if any.

Seeding a Validation Set

A validation set must be seeded along with the assignment indicating its use in this case an assignment between it and payment format it validates.

The table explains the attributes of a validation set definition:

Attribute Name	Description	Constraint
VALIDATION_SET_CODE	Unique identifier of the validation set.	Unique and must be less than or equal to 30 characters.
VALIDATION_SET_ DISPLAY_NAME	Description of the validation set.	
VALIDATION_LEVEL_ CODE	The level at which the validation id done- transactional (funds capture ORDER) or batch (payment INSTRUCTION).	Equals the following: ■ ORDER (transactional) ■ INSTRUCTION (batch)
VALIDATION_CODE_ LANGUAGE	Language in which the validation set was implemented.	Equals the following: ■ JAVA (only if LEVEL equals ORDER) ■ PLSQL (only if LEVEL equals INSTRUCTION)
VALIDATION_CODE_ PACKAGE	Language-specific package of the code-point.	Equals a fully qualified Java class name or a PL/SQL package name.
VALIDATION_CODE_ ENTRY_POINT	The code-point function/procedure name.	

A validation set and its assignment may be seeded in iPayment by creating a SQL script to insert them.

> **Note:** Insert all but translatable attribute VALIDATION_SET_ DISPLAY_NAME into IBY_VALIDATION_SETS_B. Insert attribute VALIDATION_SET_DISPLAY_NAME into IBY_VALIDATION_ SETS_B and call PL/SQL package procedure IBY_PP_MLSUTL_ PVT.VAL_SET_ADD_LANGUAGE.

The table explains the attributes of the validation set assignment:

Attribute Name	Description	Constraint
VALIDATION_ ASSIGNMENT_ID	Primary key of the assignment	Equals a unique integer
VALIDATION_SET_CODE	Code of the validation set	Equals an existing validation set
VAL_ASSIGNMENT_ ENTITY_TYPE	Type of entity the validation set is assigned to; always a format	Equals: FORMAT
ASSIGNMENT_ENTITY_ ID	Identifier of the entity the validation set is assigned to; always an existing format	Equals the code of an existing format

Validation set assignments may be directly inserted into table IBY_VAL_ASSIGNMENTS, with the primary key ID attribute generated from sequence IBY_VAL_ASSIGNMENTS_S.

Extract Generator

The extract generator produces the payment instruction file extract document, a superset of all data pertaining to the transaction. A format template is applied to the extract to produce a final payment instruction file.

The payment instruction file extract is a XML document whose structure conforms to XML schema as defined in file $IBY_TOP/patch115/publisher/defs/IBY_FCI_1_0.xsd. This XML schema supports transactions for all funds capture instrument types, such as credit card, bank account, PINless debit card and all funds capture transaction operation types, such as authorization, online capture, batch close, etc. For more information, see Funds Capture Extract.

For implementing the custom payment system integrations, the structure of the funds capture extract must be thoroughly understood to create the payment instruction file templates.

Extract Formatter

The extract formatter takes the extract document produced by the extract generator and applies a format template to produce the final payment instruction file. Oracle iPayment uses the Oracle e-Business Suite application XML Publisher (XDO) as its formatting engine.

For implementing the custom payment system integrations, templates must be created for every transaction operation type supported by the payment system. If the integration model for the payment system does not conform to one of formatted payment instruction file delivery, you can use a ordinary formatting template that produces the unchanged extract documents as its output, deliver the extract to a servlet using HTTP, and then extract document to the payment system's native payment request mechanism in the servlet map. An ordinary template is already seeded by iPayment with a template code IBY_IDENTITY.

Extract Structure

The XML Schema is the data source definition for all format templates. This means a single funds capture extract definition supports both bank account and credit card instrument types.

Element definition Table Legend

The table is an example of an element definition table.

Element Name	Datatype	Description	Data Source
Bank Account	Aggregate		
? **BankAccountID**	<Identifier>		
* BankAccountNumber	String		

Each table begins with the root element, followed by a series of indented child elements. The column preceding the child element's name indicates the child element's cardinality.

Cardinality: The values include:

- ' ' (blank) - cardinality of 1
- '?' cardinality of 0..1
- '*' cardinality of 0..n
- '+' cardinality of 1..n
- 'n' cardinality of n

Element Name: If indicated in boldface denotes the element is a complex aggregate; otherwise it is in a normal font.

Datatype: The values include:

- **Aggregate** - Complex type consisting of child elements. If the element is not the table's root element then its structure is defined elsewhere
- **Type** - Element conforms to a custom-defined type described in its own table. Elements that share a type have identical child elements.
- **Scalar** - This can be String, Integer, Real, Date, Boolean and are the same as equivalent SQL types.

Description: Describes the technical or business purpose of the element.

Extract Components

The funds capture extract consists of data elements organized hierarchically. Though the data within most such elements are generated through simple, low-cost data fetches from the iPayment schema, some are the result of complex, high-cost function calls which result in unacceptable performance when creating an extract instance. Therefore, the extract engine supports a series of user-defined rules wherein certain "expensive" elements are not populated if the rule's conditions are met.

In order to support payment formats with unique data requirements, an extensibility element called 'Extend' is present at every level of the extract, allowing the user to provide the required data using their own custom functions.

As the data required to create funds capture instructions change over time, the extract also changes by the addition of new elements. To support easy transition to new extract definitions, each extract has a version number associated with it which will be stored with every user-defined payment format. The product retains the ability to generate all previous extracts versions and, based upon the stored version number, provides each payment format with the appropriate extract instance.

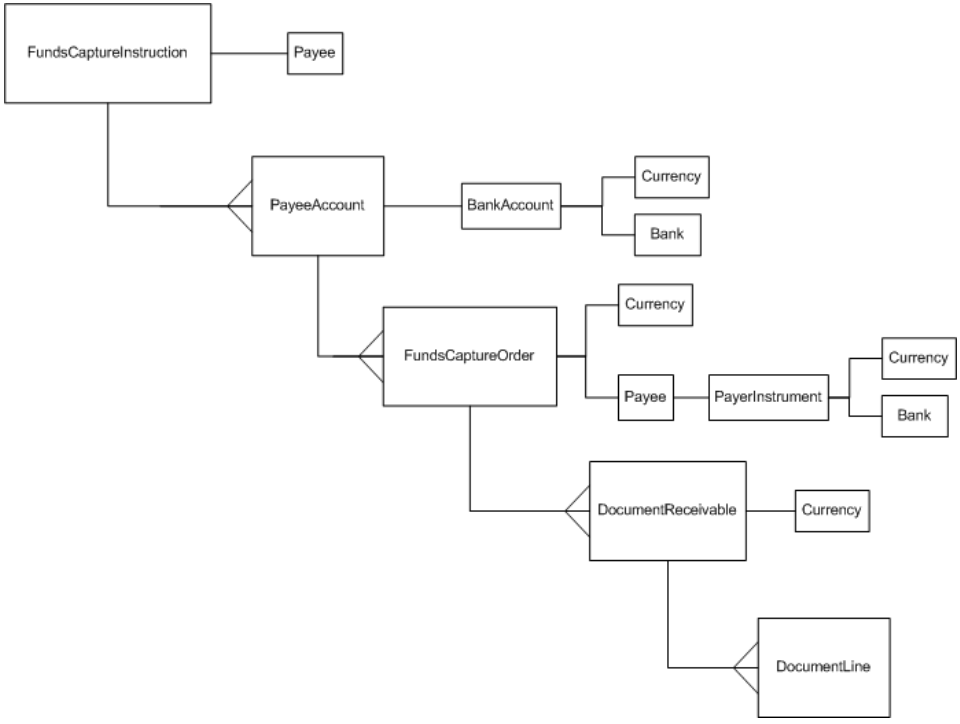
Funds Capture Extract

The funds capture extract has a 5-level structure, where each level except the last contains one or more sub-levels.

The top level is the funds capture instruction level, and represents a single instructions "file" to be delivered to the payment system. The 2nd level includes 1 or more payee accounts, each associated with a single currency and financial institution (bank). The payee accounts act as destinations for a series of funds capture orders.

A funds capture order, located at the 3rd level, is associated with a payment currency, payer information and payer bank account. A funds capture order also acts as a grouping for multiple documents receivable that reside at the 4th level. Each document receivable is associated with a single order and currency and contains multiple document lines, located at the 5th level and representing a line item from the associated order.

This diagram illustrates the logical structure.



Funds Capture Instruction Elements

The root element of an funds capture extract, corresponding to the document or file to be eventually delivered to the external payment system, is **FundsCaptureInstruction**.

Layout

Element Name	Datatype	Description	Data Source
FundsCaptureInstruction			
InstructionInfo	Aggregate	Information about the instruction.	
InstructionSequence	Aggregate	Sequential, possibly periodic, identifier of the instruction.	
InstructionTotals	Aggregate	Totals for the instruction, such as funds capture amount totals and payment instrument counts.	
+ **PayeeAccount**	Aggregate	Account (either a bank account or payment system merchant account) where captured funds will be deposited.	
* **Extend**	<NameValue>	Extensibility element. To be filled with custom user data.	

Element Name	Datatype	Description	Data Source
InstructionInfo			
InstructionInternalID	Integer	Indicates the unique identifier assigned internally to this funds capture instruction	
InstructionName	String	Name of the instruction.	
InstructionCreationDate	Time	Date of the instruction's creation.	IBY_ BA.CREATE_ DATE
InstructionSentDate	Time	Date the instruction was sent.	
InstructionStatus	<Lookup>	Current status of the instruction.	

Element Name	Datatype	Description	Data Source
InstructionSequence			
SequenceName	String	Name/code of the sequence.	
LastValue	Integer	Last/current value of the sequence.	

Element Name	Datatype	Description	Data Source
InstructionTotals			
PayeeAccountCount	Integer	The number of payee accounts in instruction.	

Payee Account Level Elements

The payee account level consists of a BankAccount element, followed by funds capture total elements, and then 1 or more funds capture order elements.

Layout

	Element Name	Datatype	Description	Data Source
PayeeAccount				
	PaymentSystemAccount	Aggregate	(Merchant) account name assigned to the payee by the acting payment system.	
?	**BankAccount**	<BankAccount>	Bank account where funds capture should be deposited.	
	Payee	Aggregate	The payee information.	
	OrderCount	Integer	Number of funds capture order using this payer account/instrument as the funds source.	
+	**FundsCaptureOrder**	Aggregate	Collection of funds capture orders using the current instrument as the funds destination.	
	AccountTotals	Aggregate	Amount totals for the account	
*	**Extend**	<NameValue>	Extensibility element; to be filled with custom user data.	

Element Name	Datatype	Description	Data Source
Payee			
Name	String	Payee business name.	IBY_PY.DBA_ NAME
Address	<Address>	Payee business address.	IBY_PY.*
ContactInfo	<Contact>	Contact points for the payee party.	
MCC	String	Merchant category code.	IBY_PY.MCC_ CODE

Element Name	Datatype	Description	Data Source
AccountTotals			
AuthorizationsTotal	<Amount>	Total of all authorizations.	IBY_BA.
CapturesTotal	<Amount>	Total of all captures/settlements.	IBY_BA. BATCHSALES
CreditsTotal	<Amount>	Total of all credits.	IBY_ BA.BATCH- CREDIT

Element Name	Datatype	Description	Data Source
PaymentSystemAccount			
AccountName	String	(Merchant) account name assigned to the payee by the acting pay- ment system.	
* **AccountOption**	<NameValue>	Account configuration option or value.	

Order Level Elements

Data Sources

Data sources for order-level elements come from tables IBY_TRXN_SUMMARIES_ALL (IBY_TS) , IBY_TRXN_CORE (IBY_TC), and IBY_TANGIBLE (IBY_TG). Joins are performed using column MTRXNID that acts as a primary key for the first 3 tables. IBY_ TG is joined to IBY_TS using column MTANGIBLEID.

Layout

	Element Name	Datatype	Description	Data Source
FundsCaptureOrder				
	OrderSourceInfo	Aggregate	Information about the order requestor.	
	OrderNumber	Aggregate	Number and identifiers associated with the order	
	PayeeOrderRefID	String	Payee-assigned order reference identifier	IBY_TG.REFINFO
	PayeeOrderMemo	String	Payee-assigned order memo.	IBY_TG.MEMO
	OrderMedium	Enumeration	Medium by which the order was received. Values include: ECOMMERCE, RETAIL, etc.	
	OrderAmount	<Amount>	Amount of the order.	IBY_TS.AMOUNT
	Payer	<3rdPartyInfo>	Party information about the funds capture payer.	
a	**PayerBankAccount**	<BankAccount>	The payer's bank account.	
a	**BankAccountTransaction**	Aggregate	The bank account transaction for the funds capture.	
|			Note there is a disjunction between element groups 'a' and 'b'. One of them may appear at the order level.	
b	**PayerCreditCard**	<CreditCard>	The payer's credit card (a source for the funds capture).	
b	**CreditCardTransaction**	Aggregate	The credit card transaction for the funds capture.	
b ?	**OriginalCCTransaction**	Aggregate	Original credit card reques which is a follow-on for the current one. In almost all cases the originating request is an authorization and this element is not present when the request is an authorization.	

Element Name	Datatype	Description	Data Source
c **PayerDebitCard**	<DebitCard>	The payer's debit card.	
c **DebitCardTransaction**	Aggregate	The debit card transaction.	
OriginalDCTransaction	Aggregate	The original debit card request; similar to element OriginalCCTransaction.	
? **DocumentReceivable**	Aggregate	Document receivable associated with the funds capture, if any.	
* **Extend**	<NameValue>	Extensibility element; to be filled with custom user data.	

Element Name	Datatype	Description	Data Source
OrderSourceInfo			
ApplicationInternalID	Integer	Internal identifier of the application originating the order request.	
ApplicationName	String	Name of the application originating the order request.	

Element Name	Datatype	Description	Data Source
OrderNumber			
PayeeOrderNumber	String	Payee-assigned order number associated with the funds capture.	

Element Name	Datatype	Description	Data Source
BankAccountTransaction			
ActionType	Enumeration	Type of EFT transaction attempted. Values include: DEBIT, CREDIT, VERIFY, VALIDATE.	
TransactionDate	Date	Date of the funds capture.	

	Element Name	Datatype	Description	Data Source
	AuthorizationMethod	Enumeration	Method by which the payer authorized the transaction. Values include: WRITTEN, INTERNET_FORM.	
	DeliveryMethod	Enumeration	Method by which the transaction is to be delivered. Values include: ACH, FASCIMILE.	
*	**Extend**	<NameValue>	Extensibility element; to be filled with custom data.	

	Element Name	Datatype	Description	Data Source
CreditCardTransaction				
	ActionType	Enumeration	Type of credit card transaction. Values include: AUTHORIZATION, AUTHCAPTURE, VOICEAUTH, CAPTURE, CREDIT, RETURN, VOID.	IBY_TS.TRXNTYPEID
	TransactionDate	Date	Date of the funds capture.	IBY_TS.CREATION_DATE
?	TraceNumber	String	Payment system-provided trace number.	
?	**POSData**	Aggregate	Point-of-sale data for card-present transactions.	IBY_TC.*
?	AuthCode	String	Authorization code. Present for voice auth transactions.	IBY_TC.AUTHCODE
?	VoiceAuthFlag	Boolean	Indicates whether the transaction was a voice authorization.	IBY_TC.VOICEAUTHFLAG
*	**Extend**	<NameValue>	Extensibility element. To be filled with custom data.	

	Element Name	Datatype	Description	Data Source
OriginalCCTransaction				

	Element Name	Datatype	Description	Data Source
	ActionType	Enumeration	Type of credit card transaction. Values include: AUTHORIZATION, AUTHCAPTURE, VOICEAUTH, CAPTURE, CREDIT, RETURN, VOID.	IBY_TS.TRXNTYPEID
	TransactionDate	Date	Date of the funds capture.	IBY_TS.CREATION_DATE
?	TraceNumber	String	Payment system-provided trace number.	
?	**POSData**	Aggregate	Point-of-sale data for card-present transactions.	IBY_TC.*
?	AuthCode	String	Authorization code provided by the payment system during the initial auth.	IBY_TC.AUTHCODE
?	VoiceAuthFlag	Boolean	Indicates whether the transaction was a voice authorization.	IBY_TC.VOICEAUTHFLAG
	Amount	<Amount>	Transaction Amount.	IBY_TS.AMOUNT
?	AVSCode	String	AVS response from the initial auth.	IBY_TC.AVSCODE
?	ReferenceCode	String	Reference Code	IBY_TC.REFERENCECODE
?	SecurityValueCheck	String	Result of the security value check.	IBY_TC.CVV2RESULT
?	PaymentSystemCode	String	Payment system code returned during the initial authorization.	IBY_TS.BEPCODE
*	**Extend**	<NameValue>	Extensibility element. To be filled with custom data.	IBY_TRXN_EXTENSIBILITY.*

Element Name	Datatype	Description	Data Source
DebitCardTransaction			

	Element Name	Datatype	Description	Data Source
	ActionType	Enumeration	Type of EFT transaction. Values include: DEBIT, CREDIT, VERIFY, VALIDATE.	
	TransactionDate	Date	Date of the funds capture.	
*	**Extend**	<NameValue>	Extensibility element. To be filled with custom data	

	Element Name	Datatype	Description	Data Source
OriginalDCTransaction				
	ActionType	Enumeration	Type of debit card transaction attempted.	IBY_TS.TRXNTYPEID
	TransactionDate	Date	Date of the funds capture.	IBY_TS.CREATION_DATE
?	TraceNumber	String	Payment system-provided trace number.	
?	AuthCode	String	Authorization code provided by the payment system during the initial auth.	IBY_TC.AUTHCODE
?	PaymentSystemCode	String	Payment system code returned during the initial authorization.	IBY_TS.BEPCODE
	DebitNetworkCode	String	Debit network code.	IBY_TC.DEBIT_NETWORK_CODE
*	**Extend**	<NameValue>	Extensibility element. To be filled with custom data	IBY_TRXN_EXTENSIBILITY.*

	Element Name	Datatype	Description	Data Source
POSData				
	ReaderCapability	Enumeration	The card reader capability.	IBY_TC.CARD_READER_CAPABILITY

Element Name	Datatype	Description	Data Source
EntryMode	Enumeration		IBY_TC.CARD_ENTRY_METHOD
CardIdMethod	Enumeration		IBY_TC.CARD_ID_METHOD
AuthSource	Enumeration		IBY_TC.CARD_AUTH_SOURCE
ReaderData	String	Card reader data. Must be in text encoded format if binary	IBY_TC.READER_DATA

Document Level Elements

Layout

Element Name	Datatype	Description	Data Source
DocumentReceivable			
DocumentID	String	Identifier assigned to this document receivable.	
DocumentStatus	<Lookup-Code>	Document Status.	
DocumentDate	Date		
DocumentCreationDate	Date	Document creation date.	
PaymentDueDate	Date	Document payment due date.	
DocumentType	<Lookup>	Document Type.	
DocumentDescription	String	User-provided document description.	
TotalDocumentAmount	<Amount>	Total amount of the document.	
PaymentAmount	<Amount>	Amount paid.	

	Element Name	Datatype	Description	Data Source
+	**Charge**	Aggregate	Charges applied to the document.	
+	**Discount**	Aggregate	Discounts applid to the document.	
+	**Tax**	Aggregate	Taxes applied to the document.	
	ShipmentOrigin	<Address>	Shipping origin of goods provided.	
	ShipmentDestination	<Address>	Shipping destination of goods provided.	
+	**DocumentLine**	Aggregate	Document lines.	
*	**Extend**	<NameValue>	Extensibility element. To be filled with custom user data.	

Document Line Level Elements

Layout

	Element Name	Datatype	Description	Data Source
DocumentLine				
	LineID	String	Identifier for the document line.	
	LineNumber	Integer	Number for the document line.	
	PONumber		Purchase Order Number.	
	LineType	<Lookup>	Document Type.	
	LineDescription	String	Description of the document line.	
	LineAmount	<Amount>	Total amount for the line.	
?	UnitRate	Real	Price per unit of this item.	
?	Quantity	Real	Number of line item units.	
?	**InitOfMeasure**	Real	Unit of measure lookup code.	
+	**Charge**	Aggregate	Charges applied to the document line.	
+	**Discount**	Aggregate	Discounts applid to the document line.	

Element Name	Datatype	Description	Data Source
+ **Tax**	Aggregate	Taxes applied to the document line.	
+ **DocumentLine**	Aggregate	Document lines.	
* **Extend**	<NameValue>	Extensibility element. To be filled with custom user data.	

Common Elements

Generic Elements

Currency information comes from FND_CURRENCIES (FND_C), with a join performed on the curreny code if detailed information is required.

Element Name	Datatype	Description	Data Source
<Currency>			
Code	String	Currency code. Acts as foreign key into table FND_CURRENCIES.	
? Symbol	String		FND_C. SYMBOL
? MinAccountableUnit	Integer		FND_C. MINIMUM_ACCOUNTABLE_UNIT
Precision	Integer		FND_C. PRECISION

Element Name	Datatype	Description	Data Source
<Amount>			
Value	Real	Scale of the amount.	
Currency	Aggregate	Currency of the amount.	

Element Name	Datatype	Description	Data Source
<NameValue>			
Name	String	Name	
Value	String	Value	

Element Name	Datatype	Description	Data Source
<Lookup>			

	Element Name	Datatype	Description	Data Source
	Code	String	Lookup code.	
	Meaning	String	Code meaning.	FND_LOOK-UPS.MEAN-ING
?	FormatValue	String	Value required by the payment format using this lookup.	

Address Elements

	Element Name	Datatype	Description	Data Source
<Address>				
	AddressInternalID	Integer	The data source of the address data.	
	AddressLine1	String		
?	AddressLine2	String		
?	AddressLine3	String		
	City	String		
?	County	String		
	State	String		
	Country	String		
	PostalCode	String		

Contact Information Elements

	Element Name	Datatype	Description	Data Source
<ContactInfo>				
	ContactName	<Person-Name>	The contact's personal name.	
	ContactLocators	<Locators>	Various means by which the contact may be located or reached.	

Element Name	Datatype	Description	Data Source
<PersonName>			
FirstName	String	Person's first name.	
LastName	String	Person's first name.	

Element Name	Datatype	Description	Data Source
<Locators>			
PhoneNumber	String	Contact phone number.	
FaxNumber	String	Contact fax machine number.	
EmailAddress	String	Contact e-mail address.	
? Website	String	Contact website.	

Bank Account Elements

Element Name	Datatype	Description	Data Source
<BankAccount>	Aggregate		
BankAccountInternalID	Integer	Identifies the data source of the parent aggregate.	
BankName	String	Name of the bank.	
BankNumber	String	Number assigned to bank.	
BranchInternalID	Integer	Identifies the data source of all subsequent branch-related elements.	
BranchName	String	Name of the bank branch.	
BranchNumber	String	Number of the bank branch.	
BranchType	<Lookup>	Bank branch type.	
? **FederalBankAccountInfo**	Aggregate	Additional information in case the bank account is owned by a federal agency.	
AccountHolderName	String	Name of the account holder.	

	Element Name	Datatype	Description	Data Source
3	**BankAssignedIdentifier**	Aggregate	Identifier assigned to the account holder by the bank.	
	EFTUserNumber	Aggregate	EFT numbers assigned by the bank to the user.	
	BankAccountName	String	Name of the bank account.	
	BankAccountNumber	String	The bank account number.	
	SwiftCode	String	SWIFT code of the bank account.	
	IBANNumber	String	IBAN of the bank account.	
	CheckDigits	String	Check digits of the bank account number.	
	BankAccountType	<Lookup>	The account type.	
	BankAccountCurrency	<Currency>	Currency by which accounts funds are denominated.	
	BankAddress	<Address>	Address of the bank.	
?	**BankContact**	<ContactInfo>	Contact at the bank.	

Element Name	Datatype	Description	Data Source
BankAssignedIdentifier			
AssignedIdentifier	String	Identifier value assigned by the bank.	
AssignedIdentifierTypeCode	String	Code giving the identifier type.	
AssignedIdentifierType	String	Description of the identifier type.	

Element Name	Datatype	Description	Data Source
EFTUserNumber			
AccountLevelEFTNumber	String	Account-level EFT user number.	
BranchLevelEFTNumber	String	Branch-level EFT user number.	

Element Name	Datatype	Description	Data Source
FederalBankAccountInfo			
FederalRFCIdentifier	String	Identifier of the US Treasury Regional Finance Center (RFC) where disbursement originates for federal agencies.	HZ_CA.CLASS_CODE:select class_codefrom hz_code_assignmentswhere (owner_table_name = 'HZ_PARTIES')and(owner_table_id = <BranchParty.party_id>)and(class_category = 'RFC_IDENTIFIER')andNVL(status, 'A') = 'A'
FederalAgencyLocationCode	String	Agency Location Code used by federal agency.	CE_BA. AGENCY_LOCATION_CODE
FederalAbbreviatedAgencyCode	String		
FederalEmployerIdentificationNumber	String		

Credit Card Elements

Element Name	Datatype	Description	Data Source
<CreditCard>			
CardNumber	String	Credit card number.	
CardExpiration	Date	Card expiration date.	
? SecurityCode	String	CVV2 or similar security value.	
CardIssuer	Enumeration	Credit card issuer type: VISA, MASTERCARD, etc.	
CardHolder	Aggregate	Card holder information.	
CardSubtype	Enumeration	Purchase card subtype. Possible values defined by lookup IBY_PURCHASECARD_SUBTYPE.	IBY_TC.CARD_SUBTYPE_CODE

Element Name	Datatype	Description	Data Source
CardDataLevel	Enumeration	Level of data supported with this instrument; possible values defined by lookup IBY_PCARD_DATA_LEVEL.	

Element Name	Datatype	Description	Data Source
CardHolder			
HolderName	<Person-Name>	Card holder name	
BillingAddress	Aggregate	Billing address of the card holder	
? PhoneNumber	String	Card holder phone number	
? EmailAddress	String	Card holder e-mail address	

Debit Card Elements

Element Name	Datatype	Description	Data Source
<DebitCard>			
CardNumber	String	Credit card number.	
CardExpiration	Date	Card expiration date.	
? SecurityValue	String	Account security code.	
CardHolder	Aggregate	Card holder information.	

Party Elements

Party elements are those which describe a participant in a financial transaction, with varying detailspending on whether they are the 1st party (user) or 3rd party (external).

Element Name	Datatype	Description	Data Source
<1stPartyInfo>			
? PartyInternalID	Integer	Indicates the source of the party data (HZ_PARTIES) with the Identifier element holding PARTY_ID.	

	Element Name	Datatype	Description	Data Source
	PartyNumber	String	User-assigned identifier for this external party.	
	Name	String	Full name of the party.	
	PartyTypeCode	String	Lookup code for the party type.	
	PartyType	String	The party type.	
?	TaxIdentifier	String	Tax number of party.	
	LegalEntityInternalID	Integer	Legal entity data source identifier, with the identifier element holding LEGAL_ENTITY_ID.	
	LegalEntityName	String	Legal name of the party.	
	Address	<Address>	Party's address.	

	Element Name	Datatype	Description	Data Source
	<3rdPartyInfo>			
?	PartyInternalID	Integer	Indicates the source of the party data (HZ_PARTIES) with the Identifier element holding PARTY_ID.	
	PartyNumber	String	User-assigned identifier for this external party.	
	Name	String	Full name of the party.	
	PartyTypeCode	String	Lookup code for the party type.	
	PartyType	String	The party type.	
?	TaxIdentifier	String	Tax number of party.	
	Address	<Address>	Party's address.	
	FirstPartyReference	String	Identifier by which this third party refers to the first.	

Document Line Elements

Element Name	Datatype	Description	Data Source
Discount			
Amount	Aggregate	Amount of the discount.	
RatePercent	Real	Discount rate in percentage.	
DiscountType	String	Type of discount.	

Element Name	Datatype	Description	Data Source
Charge			
Amount	Aggregate	Amount of the charge.	
RatePercent	Real	Charge rate as a percentage.	
ChargeType	String	Type of charge.	

Element Name	Datatype	Description	Data Source
Tax			
Amount	Aggregate	Amount of the tax.	
RatePercent	Real	Tax rate as a percentage.	
Type	String	Type of tax.	
TaxJurisdiction	String	Tax jurisdiction.	

Transmission Functions

The transmission function is responsible for delivering the payment instruction file to the payment system and implements a particular transmission protocol. This function can exist separately from the rest of iPayment, as an independent servlet.

For implementing the custom payment system integrations, transmission function code-points must be created for each transmission protocol for communicating with the payment system.

One component of a payment system specification is the transmission protocol used to deliver payment instruction files to the payment system. A transmission protocol has transmission parameters associated with it that define the required system data when making a communication attempt. A transmission protocol also has a defined transmission function code-point, which is a self-contained unit of code implementing the protocol and conforming to the interface: `oracle.apps.iby.net.TransmitFunction`.

This function can exist separately from the rest of iPayment, as its own servlet. The Base URL parameter in the Payment System details page points to this servlet.

Developing a Transmission Function

A transmission protocol is implemented through a Java class which must implement the interface `oracle.apps.iby.net.TransmitFunction`. To implement the interface, the class must define function transmit().

The table explains the signature of the function:

Argument Type	Type	Description
params	java.util.Dictionary	The map of protocol parameter names and values representing the transmission configuration. The names will be taken from the parameter name definitions for the protocol.
payload	iava.io.InputStream	The message payload being delivered.
<return>	java.io.InputStream	The response to the transmission; maybe null.

On implementing the protocol in function transmit(), the function should handle any system exception that occurs by the exception of type `oracle.apps.iby.exception.PSException` such as:

```
throw new PSException(PSException.COMMUNICATION_ERROR);
```

If a mandatory transmission protocol parameter is not set, an exception using the same class type should be thrown such as:

```
throw new PSException(PSException.CODEPOINT_ARG_ERR,
PSException.CODEPOINT_ARG_ERR_TOKEN_ARG,
      <parameter code>);
```

Once the transmission function class has been created, turn it into a servlet by:

- Placing the transmission function class in the CLASSPATH of the transmission servlet class `oracle.apps.iby.bep.TransmitServlet`.
- Opening the servlet zone properties file of the application server that will host the transmission servlet.
- Creating an alias for the class `oracle.apps.iby.bep.TransmitServlet`, so that it can be accessed as http://<host>:<post>/<servlet zone>/oramipp_<suffix>, where
  - <host> and <port> are the hostname and port of the application servlet
  - <servlet zone> is the servlet zone in which the transmission servlet will run
  - <suffix> is the 3-letter payment system suffix

For implementing the custom payment system integrations, the Java class implementing an employed transmission protocol must be distributed and then placed in the CLASSPATH of the application server that hosts the transmission function (that is, the application server that will host the iPayment transmission servlet).

Seeding a Transmission Protocol

When configuring a payment system account for the payment system in the UI, you will automatically see all transmission protocol parameters defined for the payment system's system payment profile.

The table lists the attributes of a transmission protocol:

Attribute Name	Description	Constraint
TRANSMIT_ PROTOCOL_CODE	The unique identifier for the protocol.	Must be unique and less than or equal to 30 characters.
TRANSMIT_ PROTOCOL_NAME	The description of the protocol.	
TRANSMIT_CODE_ LANGUAGE	The language in which the transmission function code-point for the protocol is implemented.	Must be JAVA.

Attribute Name	Description	Constraint
TRANSMIT_CODE_ PACKAGE	The code-point package. The fully-qualified class name of the transmission function code-point.	A fully qualified class name.
TRANSMIT_CODE_ ENTRY_POINT	The Code-point entry-point: the programming language function name that is called.	Must equal: transmit.

The table defines the attributes of a transmission protocol's parameters, sub-entities of the protocol.

Attribute Name	Description	Constraints
TRANSMIT_ PARAMETER_CODE	The identifier for the parameter.	Must be unique and less than or equal tp 30 characters among all parameters defined for the protocol.
TRANSMIT_ PARAMETER_TYPE	The datatype of the parameter.	Supported values are: ■ VARCHAR2 ■ NUMBER
MANDATORY_ FLAG	Whether the parameter is mandatory; used for UI validation during transmission configuration.	Possible values: ■ Y (Yes) ■ N (No)
TRANSMIT_ PROTOCOL_CODE	The protocol to which this parameter belongs.	
TRANSMIT_ PARAMETER_ NAME	The name of the parameter.	

A transmission protocol and its parameters can be seeded in iPayment by creating a SQL script to insert them.

Note: Insert all but translatable protocol attribute TRANSMIT_PROTOCOL_NAME name into tables IBY_TRANSMIT_PROTOCOLS_B. Insert the translatable attribute TRANSMIT_PROTOCOL_NAME into IBY_TRANSMIT_PROTOCOLS_TL and call the procedure IBY_PP_MLSUTL_PVT.TRANS_PROT_ADD_LANGUAGE.

Insert all but translatable protocol parameter attribute TRANSMIT_PARAMETER_NAME name into tables IBY_TRANSMIT_PARAMETERS_B. Insert the translatable attribute TRANSMIT_PARAMETER_NAME into IBY_TRANSMIT_PARAMETERS_TL and call the procedure IBY_PP_MLSUTL_PVT. TRANS_PARAM_ADD_LANGUAGE.

Acknowledgment Parser

A payment system sends an acknowledgement upon receiving a payment instruction delivery. The task of the acknowledgment parser is to parse the response understandable to iPayment.

For implementing the custom payment system integrations, acknowledgement parsers must be created for every transmission function. If the payment system does not support acknowledgement for a protocol, a default acknowledgement parser must still be created which returns default or trivial values.

Acknowledgement parsers are self-contained code-points that parse responses from the payment system into a form that can be processed by the iPayment engine.

Seeding an Acknowledgement Parser

The table defines the attributes when seeding an acknowledgement parser:

Attribute Name	Description	Constraint
ACK_READER_ CODE	The unique identifier for the parser.	Must be unique and less than or equal to 30 characters.
READER_ CODE_ LANGUAGE	The language that the transmission function code-point for the parser is implemented.	Myst be Java.
READER_ CODE_ PACKAGE	The code-point package- the fully-qualified class name of the acknowledgement parser code-point.	A fully qualified class name.
READER_ CODE_ENTRY_ POINT	The code-point entry-point: the programming language function name that is called.	Must be parse.

After the attributes for an acknowledgement parser are defined, they can be seeded in iPayment by creating a SQL script to insert them into the table IBY_ACK_READERS.

For implementing the custom payment system integrations, the Java class implementing an acknowledgment parser must be distributed to the user and then place it in the CLASSPATH of the application server hosting the iPayment engine.

Developing an Acknowledgement Parser

All acknowledgement parsers must sub-class the interface: `oracle.apps.iby.bep.ACKParser`. The interface has a single function, parse, with these interface:

Argument Name	Type	Description
ackFile	java.io.InputStream	The acknowledgment message or "file".
hints	java.util.Dictionary	Collection of name-values providing information about the transaction the acknowledgement is for. For example, the instrument type used, credit card issuer, etc.
<return>	bep.ACK	A corresponding object for the acknowledgment.

The hints argument to the acknowledgement parser is a collection of name-value pairs providing information about the transaction the response was created for.

The table lists the possible values in this collection:

Hint Key	Description	Value
ACKParser.CARD_ISSUER_HINT	The card issuer; for acknowledgments where the structure of the response varies based upon the card issuer.	One of the following card issuer codes: ■ AMEX ■ DINERS ■ DISCOVER ■ ENROUTE ■ JCB ■ MASTERCARD ■ UNKNOWN ■ VISA
ACKParser.INSTR_TYPE_HINT	The transaction instrument type.	One of the following values: ■ BANKACCOUNT ■ CREDITCARD ■ PINLESSDEBITCARD ■ PURCHASECARD

Hint Key	Description	Value
ACKParser.TRXN_TYPE_ID_HINT	The type of transaction.	One of the following values: ■ 2 (Auth only) ■ 3 (Auth capture) ■ 5 (Return) ■ 8/9 (Capture) ■ 11 (Credit) Value data-source is IBY_TRXN_SUMMARIES_ALL.TRXNTYPEID

Note: The hints are populated only for certain transaction types. For example, the hints will be not be populated for a batch close operation.

The result of an acknowledgement parser returns is an object derived from class: `oracle.apps.iby.bep.ACKParser`. This class is a record intended to hold various response fields mapped from the payment system response.

ACK

The abstract class `oracle.apps.iby.bep.ACK` defines the most basic acknowledgement attributes inherited by all sub-classes. The table describes the structure of this class and all derived sub-classes.

Note: Implicity, for each attribute listed for a particular class there exists get<AttributeName> and set<AttributeName> functions for accessing the attributes. The get<AttributeName> returns an object of the attribute's type and the set<AttributeName> takes an object of the attribute's type as its single argument.

Attribute Name	Type	Description
BEPErrorCode	String	Payment system error code
BEPErrorMessage	String	Payment system error message

TrxnACK

The abstract class `oracle.apps.iby.bep.TrxnACK` defines common attributes for transaction acknowledgements. It is a subclass of `oracle.apps.iby.bep.ACK` and hence inherits its attributes as well.

Attribute Name	Type	Description
OrderId	String	Order identifier for this transaction
TrxnStatus	int	Status of the transaction. The possible values are: ■ 0 (Success) ■ 1 (Communication error) ■ 2 (Duplicate order id) ■ 3 (Duplicate batch id) ■ 4 (Required field missing) ■ 5 (Payment system error) ■ 8 (Operation not supported) ■ 11 (Pending) ■ 20 (Declined)
TrxnDate	java.util.Date	Date the transaction was completed.
TrxnReqType	String	Transaction request type. The possible values are: ■ ORAPMTCAPTURE (Capture) ■ ORAPMTCLOSEBATCH (Batch close) ■ ORAPMTCREDIT (Credit) ■ ORAPMTREQ (Authorization/Auth Capture/Verification/Debit) ■ ORAPMTRETURN (Return) ■ ORAPMTVOID (Void)
ExtensiblitySet	oracle.apps.iby.util.NameValuePair[ ]	The collection of extensibility name-value pairs.

Attribute ExtensiblitySet is typed as an array of `oracle.apps.iby.util.NameValuePair` objects and may be optionally set by the parser if the payment system acknowledgement contains important data that do not correspond to any of the attributes in an appropriate acknowledgements object. Any name values returned by the ExtensiblitySet attributed are stored in table IBY_TRXN_EXTENSIBILITY and are included in the extract document of any follow on transaction

using the series of Extend sub-elements which may appear beneath the OriginalCCTransaction or OriginalDCTransaction elements.

The table explains the attributes of the `oracle.apps.iby.util.NameValuePair` class.

Attribute Name	Type	Description
Name	String	The name/key.
Value	String	The value.

CreditCardTrxnACK

The class `oracle.apps.iby.bep.CreditCardTrxnACK` holds acknowledgment information for a single credit card transaction. The table lists the attributes (not including ones derived from its parent `class oracle.apps.iby.bep.TrxnACK`).

Attribute Name	Type	Description
AuthCode	String	Authorization code.
AVSResponse	String	Address verification system response from the payment system.
SecurityCodeCheck	String	Payment system response for the credit card security code (CVV2) check.
RefCode	String	Reference code.

BankAccountTrxnACK

The `class oracle.apps.iby.bep.BankAccountTrxnACK` holds acknowledgment information for a single bank account transaction. The table lists the attributes (including inherited ones from `oracle.apps.iby.bep.TrxnACK`).

Attribute Name	Type	Description
RefCode	String	Bank reference code.
TrxnAmount	oracle.apps.iby.ecapp.Price	Actual amount of the transfer.
PostDate	java.util.Date	Date funds will be posted.
FundsCommitted	Boolean	Indicates whether funds were committed.

BatchACK

The abstract class `oracle.apps.iby.bep.BatchACK` defines common attributes of batch acknowledgments. The table list the attributes, in addition to those inherited from class `oracle.apps.iby.bep.ACK`.

Attribute Name	Type	Description
BatchId	String	Identifier of the batch.
BatchStatus	int	Status of the batch. The possible values are: ■ 0 (Success) ■ 1 (Communication error) ■ 3 (Duplicate batch id) ■ 5 (Payment system error) ■ 11 (Pending)
BatchDate	java.util.Date	Date of batch submission.
TrxnACKs	bep.TrxnACK[ ]	Collection if acknowledgments for the individual transactions in this batch.
TrxnACKType	String	Enumerated value indicating transaction acknowledgments. The values are: ■ BatchACK.TRXN_ACK_ALL (All transactions in the batch have an acknowledgement) ■ BatchACK.TRXN_ACK_POSITIVE (All transactions missing an acknowledgement assumed failed) ■ BatchACK.TRXN_ACK_NEGATIVE (All transactions missing an acknowledgement assumed successful)

Note: All batch statuses except pending (11) are final. In case a batch acknowledgement does not exist, for example immediately after a batch close, or during a batch query which occurs before the batch has been processed, a batch acknowledgement object with batch status 0 should be created although there is no existing acknowledgement file.

CreditCardBatchACK

The class `oracle.apps.iby.bep.CreditCardBatchACK` extends `oracle.apps.iby.bep.BatchACK` and contains attributes for credit card batch acknowledgments. The table explains the attributes.

Attribute Name	Type	Description
AuthTotal	oracle.apps.iby.ecapp.Price	Total authorizations completed.
CaptureTotal	oracle.apps.iby.ecapp.Price	Total authorizations completed.
CreditTotal	oracle.apps.iby.ecapp.Price	Total credits completed.

BankAccountBatchACK

The class `oracle.apps.iby.bep.BankAccountBatchACK` extends `oracle.apps.iby.bep.BatchACK` and contains attributes for credit card batch acknowledgments. The table explains the attributes.

Attribute Name	Type	Description
CreditTotal	ecapp.Price	Total credits completed.
DebitTotal	ecapp.Price	Total debits completed.

A

Risk Management

This appendix explains risk management functionality. Topics in this section include:

- Utilizing Risk Management
- Risk Management Test Scenarios

Utilizing Risk Management

iPayment supports risk management functionality. Electronic commerce applications can incorporate this feature and detect fraudulent payments. The following information describes how electronic commerce applications can utilize the risk management functionality of iPayment.

Risk Factors and Risk Formulas

iPayment is bundled with a set of risk factors. Payees can configure these factors depending on their business model. The payees can create multiple formulas using different factors and weights depending on their specific requirements. The ability to create multiple formulas provides flexibility to payees to accommodate different business scenarios. Each formula must be set up so that the sum of the weights is equal to 100. If a risk factor value is missing at the time of risk evaluation, the risk for the missing factor is considered very high in the formula.

iPayment also defines an implicit formula for each payee with default factors and weights. Administrators have the flexibility to modify the implicit formula. The following information describes how and where the implicit formula is used.

Process Flow of Risk Evaluation

1. To enable risk analysis during authorization, either set up the explicit risk flag in the input transaction object or check Enabled radio button in the Risk Management Status screen for that payee.
2. When an electronic commerce application makes a Payment Request API call, iPayment first checks the risk flag and depending on its value, decides if the payee involved in the payment request is risk enabled or not. If the risk analysis field indicates that iPayment should perform risk analysis, or if a default value is added in the field and a payee is risk enabled, iPayment evaluates either the risk formula passed in the Payment Request API or the implicit formula associated with that payee.
3. Electronic commerce application can pass a specific risk formula name by calling the overloaded Payment Request API. This API takes PmtRiskInfo object in which electronic commerce application can set up the formula name and additional information. If PmtRiskInfo object is not passed and the payee is risk enabled, iPayment evaluates the implicit formula of that payee.
4. iPayment returns the Risk Response (RiskResp) object as part of the payment response. If risk evaluation is done successfully, Risk Response object contains the risk score obtained after evaluation and the threshold value that is set up with the payee. Based on

the risk score and the threshold value, the electronic commerce application can decide whether a payment can be accepted or not

5. If the risk score is more than the threshold value, the payment request is risky.

Process Flow of Independent Risk APIs

Risk API 1

1. When an electronic commerce application invokes Risk API 1, iPayment evaluates the risk formula sent in the request or the implicit formula associated with that payee.
2. iPayment evaluates all the risk factors that are configured as part of this formula, except the AVS Code risk factor.
3. After evaluation, iPayment returns Risk response (RiskResp) object as a response to this API. This response object contains, the status of the API call, AVSCodeFlag indicating if AVS Code risk factor was part of the formula or not, risk score, and the risk threshold value that is setup for the payee. Depending on the AVSCodeFlag value, it is be decided whether to call Risk API 2 or not.

> **Note:** Partial risk score is returned if AVS Code risk factor is part of the risk formula.

Risk API 2

1. Electronic commerce applications need to call this API with the same PayeeID and formula name that were used to call Risk API 1. The risk score that was returned as part of the Risk API 1 response also needs to be sent. When electronic commerce applications call this API, iPayment checks again if the formula has AVS Code risk factor configured in it or not. If it is configured, iPayment evaluates the AVS Code risk factor.
2. After evaluating the AVS Code risk factor, iPayment calculates the final risk score of the formula using the previous risk score that was sent and the AVS Code risk factor score. This risk score is sent back to the electronic commerce application as part of the response object of this API.

Risk Management Test Scenarios

The following are three business scenarios that describe how a merchant can use the Risk Management functionality.

Merchant Selling Books and Low Priced Goods

In a small business, accepting risky instruments is a critical factor. If a customer is using a stolen credit card, the merchant should consider this transaction risky and assign this risk factor a higher weight than the other risk factors. Ship to/bill to address matching and payment history are also important risk factors. To include AVS Code risk factor, a merchant can set up a formula with weights as shown in Weight B column in the Risk Formula Setup-First Case table. The total of all the weights should be 100. For a formula that a merchant would set up in this case, see Risk Formula Setup for the First Case.

Risk Formula Setup for the First Case

This table shows the risk formula setup for a merchant selling books and low priced goods.

Factors	Weight A	Weight B
Risky Instruments	30	30
Payment Amount Limit	15	15
Transaction Amount	15	15
Ship to/Bill to	20	10
Payment History	20	10
AVS Code	0	20

Risk Factor Setup

- Payment Amount Limit

  This table shows the risk levels and the associated payment amounts.

Risk Levels	Greater than or Equal To
Low	0
Low medium	100
Medium	200

Risk Levels	Greater than or Equal To
Medium high	300
High	400

- Transaction Amount

  A transaction is high risk if the transaction amount exceeds 500 in one week. Otherwise there is no risk.

- Payment History

  This table shows the risk levels and the number of payments made in the last six months by a particular customer.

Risk Levels	Greater than or Equal To
Low	6
Low medium	4
Medium	3
Medium high	2
High	0

- AVS Code

  This table shows the risk levels and the associated AVS Codes. AVS Code risk factor evaluation is useful only for customers in the United States.

Risk Level	AVS Code
No risk	S,Y,U,X,R,E
Low	A,Z,W
Low medium	
Medium	
Medium high	
High	N

- Ship To/bill To and Risky Instruments

These risk factors do not require any setup. The evaluation will be done with the data already existing in the database.

- Risk Score

  A typical threshold value would be between medium and medium high risk score. Risk Management module evaluates the payment request and returns an overall risk score. If an overall risk score exceeds the threshold value set up by the merchant, then the merchant has to decide whether to process the request or to block the request.

Merchant Selling Electronic Goods

Risky instruments is a critical factor in this case. If a customer is using a stolen credit card, the merchant should consider this transaction risky and assign it a higher weight.

Frequency of purchase is the next important risk factor. Usually customers do not buy electronic goods frequently, and if they do, the purchases could be a fraudulent.

In this scenario, time of purchase is also should be considered as an important risk factor. If someone buys many goods after 2:00 AM, it might be a fraudulent purchase.

To include an AVS Code risk factor, a merchant can sets up a formula with weights as shown in column Weight B in Risk Formula Setup-Second Case table. The total of all the weights are 100. The AVS Code risk factor evaluation will be useful only for customers in the United States.

Risk Formula Setup for the Second Case

This table shows the risk formula set up for a merchant selling electronic goods.

Factor	Weight A	Weight B
Risky Instruments	30	30
Ship to/Bill to	15	12
Time of Purchase	15	12
Frequency of Purchase	20	10
Payment Amount	10	8
Transaction Amount Limit	10	8
AVS Code	0	20

Risk Factor Setup

- Payment Amount Limit

This table shows the risk levels and the associated payment amounts.

Risk Levels	Greater Than or Equal To
Low	500
Low medium	1000
Medium	1500
Medium high	2000
High	2500

- Transaction Amount

  This risk factor is considered high risk if the amount exceeds 2,500 in one week. Otherwise there is no risk.

- Frequency of Purchase

  This risk factor is considered high risk if the frequency of purchase exceeds ten times in the previous one week.

- AVS Codes

  This table shows the risk levels and the associated AVS codes. AVS codes risk factor evaluation is only useful for customers in the United States.

Risk Level	AVS Code (Comma Separated)
No risk	S, Y, U, X, R, E
Low	A,Z,W
Low medium	
Medium	
Medium high	
High	N

- Ship To/Bill To and Risky Instruments

  These risk factors do not require any setup. The evaluation is done through the data already existing in the database.

- Risk Score

A typical threshold value is to be between medium and medium high risk score.

The risk management module evaluates the payment request and returns an overall risk score. If an overall risk score exceeds the threshold value set up by the merchant, the merchant has to decide whether to process the request or to block the request.

Business to Business Customer

In a business to business scenario, a merchant has an established relationship with his customer. In this scenario, the Oracle Receivables risk factors take higher precedence. The merchant is interested in the customer's payment history, his credit rating, etc. All Oracle Receivables risk factors are set up through Oracle Receivables interface.

Risk Formula Setup in the Third Case

This table shows a Risk Formula setup for a business to business customer.

Factors	Weight
Overall Credit Limit	30
Transaction Credit Limit	30
Risk Codes	15
Credit Rating Codes	15
Payment History	10

Risk Factor Setup

- Overall Credit Limit: 100,000
- Transaction Credit Limit: 50,000
- Risk Codes are set up through Oracle Receivables codes.

  This table shows the risk codes and the associated risk scores set up through iPayment administration user interface.

Risk Codes	Risk Score
Low	Low
Average	Medium
Excellent	No risk

- Credit Rating Codes are set up through Oracle Receivables interface

  This table shows the set up of credit rating codes and the associated risk scores.

Credit Rating Codes	Risk Score
Low	Low
Average	Medium
Poor	High
Excellent	No risk

- Risk Score

  A typical threshold value is between medium and medium high.

  Risk management module evaluates the payment request and returns an overall risk score. If an overall risk score exceeds the threshold value set up by the merchant, then the merchant decides whether to process or block the request.

B

Error Handling

This appendix explains error handling and describes the most common errors. Topics in this section include:

- Error Handling During Payment Processing

Error Handling During Payment Processing

iPayment returns a response object to each API that an electronic commerce application calls. If the operation fails, then the response object contains status value (IBY_FAILURE), indicating that there was a failure while processing the request. In these cases, the electronic commerce application can get more information about the failure by checking the error code and the error message. Errors can happen in iPayment for various reasons. For example, wrong or duplicate data passed by the electronic commerce application, time out while communicating with Payment Systems, etc. All the errors that can occur in iPayment can be categorized in these groups:

- Common Errors
- Errors Due to Invalid or Duplicate Data
- Communication Errors
- Configuration Errors

Common Errors

This table describes the most common errors.

Error Code	Description
IBY_0001	Communications error. The payment system, the processor, or iPayment electronic commerce servlet is not accessible. You should resubmit the request at a later time.
IBY_0002	Duplicate order identifier.
IBY_0003	Duplicate batch identifier.
IBY_0004	Mandatory fields are required.
IBY_0005	Payment system specific error. Check BEPErrCode and BEPErrMsg in response objects for more information.
IBY_0006	Batch partially succeeded. Some transactions in the batch failed and some were processed correctly.
IBY_0007	The batch failed. You should correct the problem and resubmit the batch.
IBY_0008	Requested action is not supported by the payment system.
IBY_0017	Insufficient funds.
IBY_0019	Invalid credit card or bank account number.

Errors Due to Invalid or Duplicate Data

In each payment request, a payment instrument from which the money is transferred to the payee's account is involved. Generally this information is given by the end user of the electronic commerce application. Sometimes the end user might enter wrong instrument number or an instrument number that does not have enough funds. To detect these errors, iPayment provides two error codes that help electronic commerce applications to prompt the end user for correct information.

The error codes due to invalid or duplicate data and their descriptions are given in this table.

Error Code	Description
IBY_0017	Insufficient funds
IBY_0019	Invalid credit card/bank account number

Communication Errors

Since payment processing requests involve a number of different components connected over networks, time-out errors or communication errors are possible. For example, a processor successfully processes a payment request, but the network connection between the payment system and iPayment, or the network connection between iPayment's PL/SQL API package and iPayment electronic commerce servlet break down, causing the electronic commerce application not to receive the result. In some cases, electronic commerce application might crash before receiving a response. Before the crash, payment processing may have completed. Therefore, when electronic commerce application calls the API with the same information, iPayment considers this a duplicate request and raises an error. To recover from such errors, iPayment provides two approaches.

In the first approach, which is applicable to OraPmtReq and OraPmtCredit, the electronic commerce application can try the request with the retry flag set up to TRUE. This makes iPayment retry the request if it has not processed the request. Otherwise iPayment sends the same response that was sent when this request was first made.

In the second approach, which is applicable to all other operations except OraPmtReq and OraPmtCredit, the electronic commerce application needs to find out if the transactions went through successfully to re-execute any lost transactions. To enable the merchant or business to query the status of a transaction, you need to integrate the Query Transaction Status API in the electronic commerce application. This API returns all existing records for a particular transaction identifier on a payment system.

This table describes the communication error code and its description.

Error Code	Description
IBY_0001	The payment system, the processor, or iPayment's electronic commerce servlet is not accessible. You should resubmit the request at a later time.

Configuration Errors

These errors occur if payees or payment systems are not configured properly. Make sure that the URLs are entered correctly and the payee's payment system identifiers are configured properly.

C

iPayment PL/SQL APIs

This appendix explains the iPayment PL/SQL API's.

Electronic Commerce PL/SQL APIs

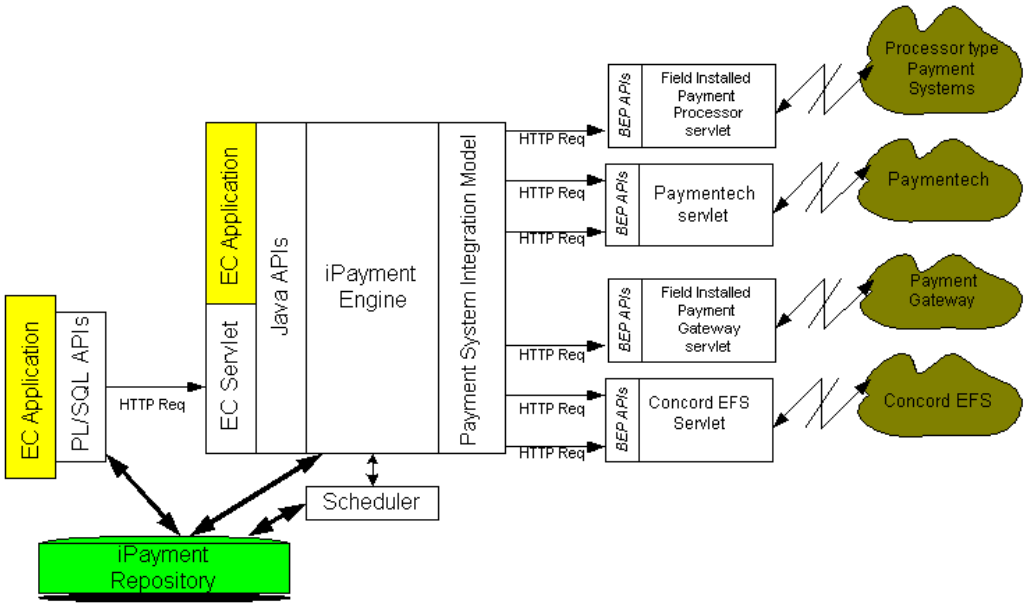
This section describes iPayment 11*i* PL/SQL API specifications for electronic commerce applications (EC-Apps) that require/prefer PL/SQL interfaces for processing credit card, PINless debit card, purchase card, and bank account transfer payment related operations. These APIs could be invoked by EC-Apps with appropriate values to perform payment operations.

The following sections contain architectural overview of iPayment PL/SQL APIs, the signatures of each API, and the definitions for each in/out parameters.

Architectural Overview

The following diagram shows the overall architecture of iPayment 11i and where the PL/SQL APIs fit inside this architecture.

Figure D– 1 iPayment Architecture



PL/SQL based EC-Apps can invoke the PL/SQL APIs which are stored in the applications database. These APIs in turn pass the payment related request, via HTTP, to the iPayment middle tier through iPayment, receives the response and passes this response to the calling application through response records.

EC-Apps can invoke the APIs either in an offline or online mode depending on the requirements of the applications.

For more information on different modes of payment, please see Understanding Offline and Online Payments in the *Oracle iPayment Concepts and Procedures Guide*. For the offline requests, the scheduler is invoked periodically to send appropriate requests to the back end payment systems and the status returned is passed back to the ECApp. For more information on how scheduler and offline operations work, see How the Scheduling System Works in

Oracle iPayment Concepts and Procedures Guide. For more information on how status is updated, please refer to Status Update API.

PL/SQL APIs Procedure Definitions

This section consists of the iPayment PL/SQL APIs which are supported in the 11*i* release. All the procedures described below are declared public and are stored as part of the applications database. All these procedures share some common IN and OUT parameters.

This table describes the common IN parameters.

p_api_version	IN	NUMBER	This parameter is to conform to the Oracle applications API standard. It is the version to be used for the API. The current supported version is 1.0 and so use 1.0
p_init_msg_list	IN	VARCHAR2	This parameter is to conform to the Oracle Applications API standard. Use FND_API.G_FALSE which is also the default value.
p_commit	IN	VARCHAR2	This parameter is to conform to the Oracle Applications API standard and hasn't been implemented for these APIs. Use FND_API.G_ FALSE which is also the default value.
p_validation_level	IN	NUMBER	This parameter is to conform to the Oracle Applications API standard. Use FND_API.G_ VALID_LEVEL_FULL which is also the default value.
p_ecapp_id	IN	NUMBER	The id of EC-App which is invoking the API.

This table describes the common OUT parameters.

x_return_status	OUT	VARCHAR2	Used to indicate the return status of the procedure. This parameter is to conform to the Oracle applications API standard.
x_msg_count	OUT	NUMBER	The error message count holds the number of error messages in the API message list. This parameter is to conform to the Oracle applications API standard
x_msg_data	OUT	VARCHAR2	Contains the error messages. This parameter is to conform to the Oracle applications API standard

Note: These APIs return a single x_return_status as 'S' for overall success, and 'U' for any type of errors (both API internal errors and iPayment processing errors included).

If the value of x_return_status is not 'S', then the calling program needs to check both the API message list parameter x_msg_data and the iPayment response objects to identify whether it is an API implementation error or an iPayment related error. The API message list messages will hold all API implementation errors, while the API response objects will hold iPayment related success/errors.

The error message from iPayment may include messages from the back end payment systems in special response object fields (BEPErrCode, BEPErrMessage, ErrLocation). Hence the error messages from iPayment are not added into the message list, consistent with the Java APIs.

The PL/SQL APIs provided by iPayment are of two types:

- Payment Processing APIs
- Payment Instrument Registration APIs

Payment Processing APIs

These APIs are the transactional APIs that support various payment operations. The electronic commerce applications use these APIs to process various transaction types. For example, authorization of credit cards, PINless debit cards, and purchase cards, transfer of funds from one bank account to another, capture, cancel, return, and others. A list of such APIs are provided below. All the procedures described below are declared public and are stored in the PL/SQL Package IBY_PAYMENT_ADAPTER_PUB as part of the applications database.

The following PL/SQL APIs are described in this section:

- OraPmtReq
- OraPmtMod
- OraPmtCanc
- OraPmtCapture
- OraPmtReturn
- OraPmtVoid
- OraPmtCredit
- OraPmtQryTrxn
- OraPmtCloseBatch
- OraPmtQueryBatch
- OraPmtInq
- OraRiskEval

For more information on Error Codes and their meaning, see Error Handling.

For a description of the PL/SQL records with possible values of all the APIs, see "PL/SQL Record/Table Types Definitions" in this appendix.

OraPmtReq

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API handles new Payment requests from EC-Apps. EC-Apps can make an offline or online payment requests by setting “PmtMode” attribute in “p_pmtreqtrxn_rec” “OFFLINE” or “ONLINE”. If the attribute of the record is not set explicitly then, by default, payment is considered as “ONLINE” request. If “PmtMode” is set to “OFFLINE”, then attribute “Settlement_date” in “p_pmtreqtrxn_rec” must be set to proper value.

This API can be used to validate a bank account before transferring funds from it, and initiate a PINless debit card transaction.

Sometimes credit card processing networks decline transactions with a referral message indicating that the merchant must call the cardholder’s issuing bank to complete the transaction. The payment information in such cases is submitted over the phone. If the transaction is approved, the merchant is provided with an authorization code for the transaction. To facilitate follow-on transactions through iPayment for this voice authorization (for example, capture or void), OraPmtReq API provides voice authorization support.

This API returns a transaction ID if payment request is processed successfully, which can be used later to initiate follow on operation on the payment. For example, to modify a payment or capture the payment, the EC-App will need to pass this transaction ID along with other information that is needed to perform the operation requested.

Response object of the API contains risk response if the payee involved in the payment (on-line) request is risk enabled. EC-Apps can check RiskRespIncluded field in the response to verify if there is a Risk response from iPayment, and if so, check the RiskResponse record for details. This API also accepts additional OPTIONAL risk-related input parameters for evaluating risk of an on-line payment request.

For more information on using Risk Management, see Utilizing Risk Management.

In summary, this API can be used to:

- Validate a bank account
- Authorize credit transactions
- Transfer funds from a bank account
- Do risk analysis
- Schedule payments to be made in future (Offline payments)

Note: This API is also available in an overloaded form, without the Risk related input parameter to enable EC-Apps that may not need risk evaluation functionality to call the OraPmtReq API directly without any Risk related input. All the other inputs and outputs are identical to the above API. Only the input parameter p_riskinfo_rec is absent in the overloaded API's signature definition.

Signature

```
Procedure OraPmtReq (p_api_version        IN    NUMBER,
                     p_init_msg_list      IN    VARCHAR2:=FND_API.G_FALSE
                     p_commit             IN    VARCHAR2:=FND_API.G_FALSE
                     p_validation_level   IN    NUMBER:=FND_API.G_VALID
                                                LEVEL_FULL
                     p_ecapp_id           IN    NUMBER,
                     p_payee_rec          IN    Payee_rec_type,
                     p_payer_rec          IN    Payer_rec_type,
                     p_pmtinstr_rec       IN    PmtInstr_rec_type,
                     p_tangible_rec       IN    Tangible_rec_type,
                     p_pmtreqtrxn_rec     IN    PmtReqTrxn_rec_type,
                     p_riskinfo_rec       IN    RiskInfo_rec_type,
                     x_return_status      OUT   VARCHAR2,
                     x_msg_count          OUT   NUMBER,
                     x_msg_data           OUT   VARCHAR2,
                     x_reqresp_rec        OUT   ReqResp_rec_type)
```

Overloaded API Signature (without risk objects):

```
Procedure OraPmtReq (p_api_version        IN    NUMBER,
                     p_init_msg_list      IN    VARCHAR2:=FND_API.G_FALSE,
                     p_commit             IN    VARCHAR2:=FND_API.G_FALSE,
                     p_validation_level   IN    NUMBER:=FND_API.G_VALID_
```

```
                                    LEVEL_FULL,
p_ecapp_id          IN              NUMBER,
p_payee_rec         IN              Payee_rec_type,
p_payer_rec         IN              Payer_rec_type,
p_pmtinstr_rec      IN              PmtInstr_rec_type,
p_tangible_rec      IN              Tangible_rec_type,
p_pmtreqtrxn_rec    IN              PmtReqTrxn_rec_type,
x_return_status     OUT             VARCHAR2,
x_msg_count         OUT             NUMBER,
x_msg_data          OUT             VARCHAR2,
x_reqresp_rec       OUT             ReqResp_rec_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_payee_rec	IN	Payee_rec_type		Required
		Payee_ID	VARCHAR2	Required
p_payer_rec	IN	Payer_rec_type	-	Optional
		Payer_ID	VARCHAR2	Optional
p_pmtinstr_rec	IN	PmtInstr_rec_type	-	Required
		1. PmtInstr_ID	NUMBER	Mandatory if 2, 3, 4 and 5 are null

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		2. CreditCardInstr	CreditCardInstr_rec_type	Mandatory if 1, 3, 4 and 5 are null
			CC_Num	Required
			CC_ExpDate	Required
			1.CC_ BillingAddr.Address 1	Optional*
			2.CC_ BillingAddr.City	Optional*
			3.CC_ BillingAddr.State	Optional*
			4.CC_ BillingAddr.PostalCode	Optional*
			5.CC_ BillingAddr.Address 2	Optional
			6. CC_ BillingAddr.Address 3	Optional
			7.CC_ BillingAddr.County	Optional
			8.CC_ BillingAddr.Country	Optional
			9. CC_Type	Optional
			10.CC_HolderName	Optional
			11. FIName	Optional

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		3.PurchasetCardInstr	PurchaseCardInstr_ rec_type	Mandatory if 1, 2, 4 and 5 are null
			PC_Num	Required
			PC_ExpDate	Required
			1.PC_ BillingAddr.Address 1	Optional*
			2.PC_ BillingAddr.City	Optional*
			3.PC_ BillingAddr.State	Optional*
			4.PC_ BillingAddr.PostalC ode	Optional*
			5.PC_ BillingAddr.Address 2	Optional
			6. PC_ BillingAddr.Address 3	Optional
			7.PC_ BillingAddr.County	Optional
			8.PC_ BillingAddr.Country	Optional
			9. PC_Type	Optional
			10.PC_HolderName	Optional
			11. FIName	Optional
			12. PC_SubType	Mandatory

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		4. DebitCardInstr	DebitCardInstr_rec_ type	Mandatory if 1, 2, 3 and 5 are null
			DC_Num	Required
			DC_ExpDate	Required
			1.DC_ BillingAddr.Address 1	Optional*
			2.DC_ BillingAddr.City	Optional*
			3.DC_ BillingAddr.State	Optional*
			4.DC_ BillingAddr.PostalC ode	Optional*
			5.DC_ BillingAddr.Address 2	Optional
			6. DC_ BillingAddr.Address 3	Optional
			7.DC_ BillingAddr.County	Optional
			8.DC_ BillingAddr.Country	Optional
			9. DC_Type	Optional
			10.DC_HolderName	Optional
			11. FIName	Optional
		5. DualPaymentInstr	DualPaymentInstr_ rec_type	Mandatory if 1, 2, 3 and 4 are null

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
			1. PmtInstr_ID	Mandatory
			2. PmtInstr_ ShortName	Optional
			3. BnfPMTInstr_ID	Optional
			4. BnfPmtInstr_ ShortName	Optional
		6. PmtInstr_ShortName	VARCHAR2	Optional
p_tangible_rec	IN	Tangible_rec_type		Required
		1. Tangible_ID	VARCHAR2	Required
		2. Tangible_Amount	NUMBER	Required
		3. Currency_Code	VARCHAR2	Required
		4. RefInfo	VARCHAR2	Optional
		5. Memo	VARCHAR2	Optional
		6. Acct_Num	VARCHAR2	Optional
		7. OrderMedium	VARCHAR2	Optional
		8. EFTAuthMethod	VARCHAR2	Optional
p_pmtreqtrxn_rec	IN	PmtReqTrxn_rec_type		Required
	IN	PmtMode	VARCHAR2	Required
	IN	CVV2	VARCHAR2	Optional
	IN	Settlement_Date	DATE	Mandatory for PmtMode = OFFLINE
	IN	Check_Flag	VARCHAR2	Optional with default value = ‘TRUE’ for PmtMode = OFFLINE

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
	IN	Auth_Type	VARCHAR2	Mandatory for Credit Card where value is AUTHONLY, AUTHCAPTURE, or AUTHANDCAPTURE Mandatory for Electronic Funds Transfer Online Validation where value is VALIDATE
	IN	Retry_Flag	VARCHAR2	Optional
	IN	Org_ID	NUMBER	Optional
	IN	NLS_LANG	VARCHAR2	Optional
	IN	PONum	NUMBER	Mandatory for Purchase Card
	IN	TaxAmount	NUMBER	Optional
	IN	ShipFromZip	VARCHAR2	Optional
	IN	ShipToZip	VARCHAR2	Optional
	IN	AnalyzeRisk	VARCHAR2	Optional
	IN	Retail Data_rec_type		Optional
p_riskinfo_rec	IN	RiskInfo_rec_type		Optional
		Formula_Name	VARCHAR2	Optional
		ShipToBillTo_Flag	VARCHAR2	Optional
		Time_Of_Purchase	VARCHAR2	Optional
		Customer_Acct_Num	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
x_reqresp_rec	OUT	ReqResp_rec_type		
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response:	Response_rec_type:	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	
	OUT	Trxn_ID	NUMBER	
	OUT	RefCode	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
(OPERATION RELATED RESPONSE)				
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	Authcode	VARCHAR2	
	OUT	AVSCode	VARCHAR2	
	OUT	PmtInstr_Type	VARCHAR2	
	OUT	Acquirer	VARCHAR2	
	OUT	VpsBatch_ID	VARCHAR2	
	OUT	AuxMsg	VARCHAR2	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
(RISK RELATED RESPONSE)	OUT	RiskRespIncluded	VARCHAR2	
		RiskResponse	RiskResp_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		Additional_ ErrMessage	VARCHAR2	
		Risk_Score	NUMBER	
		Risk_Threshold_ Val	NUMBER	
		Risk_Flag	VARCHAR2	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp		
		EarliestSettlement _Date	DATE	
		Scheduled_Date	DATE	

OraPmtMod

API type: Public

Prerequisites for calling the API: Existing scheduled Off-line payment request

Function(s) performed by the API:

This API handles modifications to existing Payment request. This API can be used to modify a payment requested by an EC-App. Payment modification is relevant in case of Scheduled (i.e., OFFLINE) payments. Users may decide to modify a payment before it is sent to the payment system.

The payee and tangible_id cannot be modified. The payment instrument can be modified, but the modified/new payment instrument should be of the same type as the original request. (If original instrument is a credit card, the modified instrument should be a credit card.)

Signature

Procedure OraPmtMod (p_api_version IN NUMBER,

```
p_init_msg_list      IN    VARCHAR2 := FND_API.G_FALSE,
p_commit             IN    VARCHAR2 := FND_API.G_FALSE,
p_validation_level   IN    NUMBER := FND_API.G_VALID_
                           LEVEL_FULL,
p_ecapp_id           IN    NUMBER,
p_payee_rec          IN    Payee_rec_type,
p_payer_rec          IN    Payer_rec_type,
p_pmtinstr_rec       IN    PmtInstr_rec_type,
p_tangible_rec       IN    Tangible_rec_type,
p_modtrxn_rec        IN    ModTxn_rec_type,
x_return_status      OUT   VARCHAR2,
x_msg_count          OUT   NUMBER,
x_msg_data           OUT   VARCHAR2,
x_modresp_rec        OUT   ModResp_rec_type)
```

Parameters

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_payee_rec	IN	Payee_rec_type		Required
		Payee_ID	VARCHAR2	Required
p_payer_rec	IN	Payer_rec_type		Optional
		Payer_ID	VARCHAR2	Optional
		Payer_Name	VARCHAR2	Optional
p_pmtinstr_rec	IN	PmtInstr_rec_type		Required

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
		1. PmtInstr_ID	NUMBER	Mandatory if 2, 3, 4 and 5 are null
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory. .		2. CreditCardInstr	CreditCardInstr_ rec_type	Mandatory if 1, 3, 4 and 5 are null
			CC_Num	Required
			CC_ExpDate	Required
			1.CC_ BillingAddr.Address1	Optional*
			2.CC_ BillingAddr.City	Optional*
			3.CC_ BillingAddr.State	Optional*
			4.CC_ BillingAddr.PostalCode	Optional*
			5.CC_ BillingAddr.Address2	Optional
			6. CC_ BillingAddr.Address3	Optional
			7.CC_ BillingAddr.County	Optional
			8.CC_ BillingAddr.Country	Optional

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
			9. CC_Type	Optional
			10.CC_ HolderName	Optional
			11. FIName	Optional
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		3.PurchasetCardInstr	PurchaseCardInstr_rec_type	Mandatory if 1, 2, 4 and 5 are null
			PC_Num	Required
			PC_ExpDate	Required
			1.PC_ BillingAddr.Address1	Optional*
			2.PC_ BillingAddr.City	Optional*
			3.PC_ BillingAddr.State	Optional*
			4.PC_ BillingAddr.PostalCode	Optional*
			5.PC_ BillingAddr.Address2	Optional
			6. PC_ BillingAddr.Address3	Optional
			7.PC_ BillingAddr.County	Optional

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
			8.PC_ BillingAddr.Country	Optional
			9. PC_Type	Optional
			10.PC_ HolderName	Optional
			11. FIName	Optional
			12. PC_SubType	Mandatory
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		4. DebitCardInstr	DebitCardInstr_ rec_type	Mandatory if 1, 2, 3 and 5 are null
			DC_Num	Required
			DC_ExpDate	Required
			1.DC_ BillingAddr.Address1	Optional*
			2.DC_ BillingAddr.City	Optional*
			3.DC_ BillingAddr.State	Optional*
			4.DC_ BillingAddr.PostalCode	Optional*
			5.DC_ BillingAddr.Address2	Optional
			6. DC_ BillingAddr.Address3	Optional

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
			7.DC_ BillingAddr.County	Optional
			8.DC_ BillingAddr.Country	Optional
			9. DC_Type	Optional
			10.DC_ HolderName	Optional
			11. FIName	Optional
		5. DualPaymentInstr	DualPaymentInstr_rec_type	Mandatory if 1, 2, 3 and 4 are null
			1. PmtInstr_ID	Mandatory
			2. PmtInstr_ ShortName	Optional
			3. BnfPMTInstr_ ID	Optional
			4. BnfPmtInstr_ ShortName	Optional
		6. PmtInstr_ShortName	VARCHAR2	Optional
p_tangible_rec	IN	Tangible_rec_type		Required
		1.Tangible_ID	VARCHAR2	Required
		2 Tangible_Amount	NUMBER	Required
		3.Currency_Code	VARCHAR2	Required
		4.RefInfo	VARCHAR2	Optional
		5. Memo	VARCHAR2	Optional
		6. Acct_Num	VARCHAR2	Optional
		7. OrderMedium	VARCHAR2	Optional
		8. EFTAuthMethod	VARCHAR2	Optional
p_modtrxn_rec	IN	ModTrxn_rec_type		Required

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
		PmtMode	VARCHAR2	Required
		Trxn_ID	NUMBER	Required
		Auth_Type	VARCHAR2	Mandatory for CreditCard
				Mandatory for Electronic Funds Transfer Online Validation where value is VALIDATE
		Settlement_Date	DATE	Mandatory for PmtMode= OFFLINE
		Check_Flag	VARCHAR2	Optional with default value = 'TRUE' for PmtMode = OFFLINE
	IN	PONum	NUMBER	Mandatory for Purchase Card
	IN	TaxAmount	NUMBER	Optional
	IN	ShipFromZip	VARCHAR2	Optional
	IN	ShipToZip	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_modresp_rec	OUT	ModResp_rec_type		

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_ type	
		Status	NUMBER	
		ErrCode	VARCHA R2	
		ErrMessage	VARCHA R2	
		NLS_LANG	VARCHA R2	
	OUT	Trxn_ID	NUMBER	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp		
		EarliestSettlement_Date	DATE	
		Scheduled_Date	DATE	

OraPmtCanc

API type: Public

Prerequisites for calling the API: Existing scheduled Offline payment operation that should be canceled. The payment operations that can be canceled are payment request, capture etc.

Function(s) performed by the API:

This API handles cancellations of offline payment operations. This API can cancel the entire operation before it reaches the payment system for offline operations, since the operation information is maintained in the database. The cancellation will not happen if the payment operation is already submitted to the payment system.

Signature

```
Procedure OraPmtCanc (p_api_version      IN    NUMBER,
                      p_init_msg_list    IN    VARCHAR2 := FND_API.G_FALSE,
                      p_commit           IN    VARCHAR2 := FND_API.G_FALSE,
                      p_validation_level IN    NUMBER := FND_API.G_VALID_
```

```
                                        LEVEL_FULL,
    p_ecapp_id          IN              NUMBER,
    p_canctrxn_rec      IN              CancelTrxn_rec_type,
    x_return_status     OUT             VARCHAR2,
    x_msg_count         OUT             NUMBER,
    x_msg_data          OUT             VARCHAR2,
    x_cancresp_rec      OUT             CancelResp_rec_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_canctrxn_rec	IN	CancelTrxn_rec_type		Required
	IN	Trxn_ID	NUMBER	Required
		Req_Type	VARCHAR2	Required
	IN	NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_cancresp_rec	OUT	CancelResp_rec_type		
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCodeErr	VARCHAR2	
		Message	VARCHAR2	
		NLS_LANG	VARCHAR2	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
(CANCEL OPERATION RELATED RESPONSE)				
	OUT	Trxn_ID	NUMBER	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	

OraPmtCapture

API type: Public

Prerequisites for calling the API: Previously authorized payment request operation.

Function(s) performed by the API:

The Capture API is invoked by the EC-App to perform a capture of a previously authorized operation. The captured amount may or may not be the same as the authorized amount. An authorized operation can only be captured once.

Each authorization operation is valid for a limited time until expiration (3-30 days depending on the cardholder's bank). If capture cannot be performed before the authorization expires, the merchant must reauthorize the payment, with a different tangible_ id.

Signature

```
Procedure OraPmtCapture (p_api_version          IN   NUMBER,
                         p_init_msg_list        IN   VARCHAR2 := FND_API.G_FALSE,
                         p_commit               IN   VARCHAR2 := FND_API.G_FALSE,
                         p_validation_levelIN        NUMBER :=FND_API.G_VALID_
                                                     LEVEL_FULL,
                         p_ecapp_id             IN   NUMBER,
                         p_capturetrxn_rec      IN   CaptureTrxn_rec_type,
                         x_return_status        OUT  VARCHAR2,
```

```
                    x_msg_count     OUT NUMBER,
                    x_msg_data      OUT VARCHAR2,
                    x_capresp_rec   OUT CaptureResp_rec_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_capturetrxn_rec	IN	CaptureTrxn_rec_type		Required
		Trxn_ID	NUMBER	Required
		PmtMode	VARCHAR2	Required
		Settlement_Date	DATE	Mandatory if PmtMode is OFFLINE
		Currency	VARCHAR2	Required
		Price	NUMBER	Required
		NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_capresp_rec	OUT	CaptureResp_rec_type		
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
(CAPTURE OPERATION RELATED RESPONSE)				
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	PmtInstr_Type	VARCHAR2	
	OUT	RefCode	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp		
		EarliestSettlement_Date	DATE	
		Scheduled_Date	DATE	

OraPmtReturn

API type: Public

Prerequisites for calling the API: Previous payment capture operation

Function(s) performed by the API:

This API is invoked by the EC-App to credit a customer account when a customer returns goods purchased through a previously captured payment operation. Only one return can be applied against each order, subsequent returns must be treated as standalone credits. The operation takes in the transaction ID of the initial payment operation, and returns the same transaction ID as part of the output.

Signature

Procedure OraPmtReturn (	p_api_version	IN	NUMBER,
	p_init_msg_list	IN	VARCHAR2 := FND_API.G_FALSE,
	p_commit	IN	VARCHAR2 := FND_API.G_FALSE,
	p_validation_level	IN	NUMBER := FND_API.G_VALID_

```
                                        LEVEL_FULL,
    p_ecapp_id          IN  NUMBER,
    p_returntrxn_rec    IN  ReturnTrxn_rec_type,
    x_return_status     OUT VARCHAR2,
    x_msg_count         OUT NUMBER,
    x_msg_data          OUT VARCHAR2,
    x_retresp_rec       OUT ReturnResp_rec_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_returntrxn_rec	IN	ReturnTrxn_rec_type		Required
		Trxn_ID	NUMBER	Required
		PmtMode	VARCHAR2	Required
		Settlement_Date	DATE	Mandatory if PmtMode is OFFLINE
		Currency	VARCHAR2	Required
		Price	NUMBER	Required
		NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_returnresp_rec	OUT	ReturnResp_rec_type		

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	
(RETURN OPERATION RELATED RESPONSE)				
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	PmtInstr_Type	VARCHAR2	
	OUT	RefCode	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp		
		EarliestSettlement_Date	DATE	
		Scheduled_Date	DATE	

OraPmtVoid

API type: Public

Prerequisites for calling the API:Existing payment operations

Function(s) performed by the API:

The Void API voids a capture or return operation for an order before the operation is settled. It takes in the transaction ID of the initial payment request and returns the same transaction ID as part of the output. Void Operations can be performed on “Capture”, “Return” and “Credit” Operations for all back-end Payment Systems, and on “Authorization” operations for certain back-end payment systems.

The Void operation has to be used to void the most recent operation for the designated Order ID. For example, you perform a capture and then a return operation for a particular Order ID, if you try to void the capture, it'll result in an error.

Signature

```
Procedure OraPmtVoid (p_api_version          IN    NUMBER,
                      p_init_msg_list        IN    VARCHAR2 := FND_API.G_FALSE,
                      p_commit               IN    VARCHAR2 := FND_API.G_FALSE,
                      p_validation_level     IN    NUMBER :=  FND_API.G_VALID_
                                                   LEVEL_FULL,
                      p_ecapp_id             IN    NUMBER,
                      p_voidtrxn_rec         IN    VoidTrxn_rec_type,
                      x_return_status        OUT   VARCHAR2,
                      x_msg_count            OUT   NUMBER,
                      x_msg_data             OUT   VARCHAR2,
                      x_voidresp_rec         OUT   VoidResp_rec_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_voidtrxn_rec	IN	VoidTrxn_rec_type		Required
		Trxn_ID	NUMBER	Required
		PmtMode	VARCHAR2	Required
		Settlement_Date	DATE	Mandatory if PmtMode is OFFLINE

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
		Trxn_Type	VARCHAR2	Required
		NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_voidresp_rec	OUT	VoicResp_rec_type		
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	
(VOID OPERATION ONLINE MODE RELATED RESPONSE)				
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	PmtInstr_Type	VARCHAR2	
	OUT	RefCode	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp		
		EarliestSettlement_Date	DATE	
		Scheduled_Date	DATE	

OraPmtCredit

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API is invoked by the EC-App to credit a customer account in the case that the merchant wants to issue a “standalone credit” (i.e., a credit not associated with any previous order). It returns the transaction ID as part of the output.

The OraPmtCredit API is also invoked by EC-App during an EFT transaction.

Signature

```
Procedure OraPmtCredit (p_api_version          IN    NUMBER,
                        p_init_msg_list        IN    VARCHAR2 := FND_API.G_FALSE,
                        p_commit               IN    VARCHAR2 := FND_API.G_FALSE,
                        p_validation_level     IN    NUMBER := FND_API.G_VALID_
                                                     LEVEL_FULL,
                        p_ecapp_id             IN    NUMBER,
                        p_payee_rec            IN    Payee_rec_type,
                        p_pmtinstr_rec         IN    PmtInstr_rec_type,
                        p_tangible_rec         IN    Tangible_rec_type,
                        p_credittrxn_rec       IN    CreditTrxn_rec_type,
                        x_return_status        OUT   VARCHAR2,
                        x_msg_count            OUT   NUMBER,
                        x_msg_data             OUT   VARCHAR2,
                        x_creditresp_rec       OUT   CreditResp_rec_type)
```

Parameters

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_ level	IN	NUMBER	-	Optional

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
p_ecapp_id	IN	NUMBER	-	Required
p_payee_rec	IN	Payee_rec_type		Required
		Payee_ID	VARCHAR2	Required
p_pmtinstr_rec	IN	PmtInstr_rec_type		Required
		1. PmtInstr_ID	NUMBER	Mandatory if 2, 3, and 4 are null
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		2. CreditCardInstr	CreditCardInstr_rec_type	Mandatory if 1, 3 and 4 are null
			CC_Num	Required
			CC_ExpDate	Required
			1.CC_BillingAddr.Address1	Optional*
			2.CC_BillingAddr.City	Optional*
			3.CC_BillingAddr.State	Optional*
			4.CC_BillingAddr.PostalCode	Optional*
			5.CC_BillingAddr.Address2	Optional
			6. CC_BillingAddr.Address3	Optional
			7.CC_BillingAddr.County	Optional
			8.CC_BillingAddr.Country	Optional
			9. CC_Type	Optional
			10.CC_HolderName	Optional
			11. FIName	Optional

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
Note: Address record is optional overall, but if passed, then the 4 fields Addr1, City, State, Postal Code (1,2,3,4)* are together Mandatory.		3.PurchasetCardInstr	PurchaseCardInstr_rec_type	Mandatory if 1 and 2 are null
			PC_Num	Required
			PC_ExpDate	Required
			1.PC_BillingAddr.Address1	Optional*
			2.PC_BillingAddr.City	Optional*
			3.PC_BillingAddr.State	Optional*
			4.PC_BillingAddr.PostalCode	Optional*
			5.PC_BillingAddr.Address2	Optional
			6. PC_BillingAddr.Address3	Optional
			7.PC_BillingAddr.County	Optional
			8.PC_BillingAddr.Country	Optional
			9. PC_Type	Optional
			10.PC_HolderName	Optional
			11. FIName	Optional
			12. PC_SubType	Mandatory
		4. PmtInstr_ShortName	VARCHAR2	Optional
p_tangible_rec	IN	Tangible_rec_type		Required
		1.Tangible_ID	VARCHAR2	Required
		2 Tangible_Amount	NUMBER	Required
		3.Currency_Code	VARCHAR2	Required
		4.RefInfo	VARCHAR2	Optional
		5. Memo	VARCHAR2	Optional

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
		6. Acct_Num	VARCHAR2	Optional
		7. OrderMedium	VARCHAR2	Optional
		8. EFTAuthMethod	VARCHAR2	Optional
p_credittrxn_rec	IN	CreditTrxn_rec_type		Required
	IN	PmtMode	VARCHAR2	Required
		Settlement_Date	DATE	Mandatory for PmtMode= OFFLINE
		Org_ID	NUMBER	Optional
		NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_creditresp_rec	OUT	CreditResp_rec_type		
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	
(CREDIT OPERATION RELATED RESPONSE)				
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	PmtInstr_Type	VARCHAR2	
	OUT	RefCode	VARCHAR2	

Parameter	IN/ OUT	Data Type	Sub Type	Required/ Optional
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp		
		EarliestSettlement_Date	DATE	
		Scheduled_Date	DATE	

OraPmtQryTrxn

API type: Public

Prerequisites for calling the API:None

Function(s) performed by the API:

This API provides an interface for querying payment operations details. This API will return either all the operations performed on the queried transaction id or the latest operation, based on the value of the History_Flag which is one of the input parameters. Payment Mode is always ‘ONLINE’ for this operation.

Signature

```
Procedure OraPmtQryTrxn (p_api_version IN        NUMBER,
                         p_init_msg_list   IN    VARCHAR2 := FND_API.G_FALSE,
                         p_commit          IN    VARCHAR2 := FND_API.G_FALSE,
                         p_validation_levelIN    NUMBER :=  FND_API.G_VALID_
                                                 LEVEL_FULL,
                         p_ecapp_id        IN    NUMBER,
                         p_querytrxn_rec   IN    QueryTrxn_rec_type,
                         x_return_status   OUT   VARCHAR2,
                         x_msg_count       OUT   NUMBER,
                         x_msg_data        OUT   VARCHAR2,
```

x_qrytrxnrespsum_rec OUT QryTrxnRespSum_rec_type,
x_qrytrxnrespdet_tbl OUT QryTrxnRespDet_tbl_type)

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_querytrxn_rec	IN	QueryTrxn_rec_type		Required
		Trxn_ID	NUMBER	Required
		History_Flag	VARCHAR2	Required
		NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_qrytrxnrespsum_rec	OUT	QryTrxnRespSum_rec_ type		
	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessag	VARCHAR2	
		NLS_LANG	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
x_qrytrxnrespdet_tbl N.B.: All detail records name-value pairs will have '-n' suffixed to show the index value 'n'	OUT	QryTrxnRespDet_tbl_type		
	OUT	Status	NUMBER	
	OUT	StatusMsg	VARCHAR2	
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	PmtInstr_Type	VARCHAR2	
	OUT	Currency	VARCHAR2	
	OUT	Price	NUMBER	
	OUT	RefCode	VARCHAR2	
	OUT	AuthCode	VARCHAR2	
	OUT	AVSCode	VARCHAR2	
	OUT	Acquirer	VARCHAR2	
	OUT	VpsBatch_ID	VARCHAR2	
	OUT	AuxMsg	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	

OraPmtCloseBatch

API type: Public

Prerequisites for calling the API: Existing current batch of operations

Function(s) performed by the API:

This API allows a merchant or business to close a batch of previously performed operations. The operation types that can be included in a batch are capture, return, and credit. This

operation is mandatory for a terminal-based merchant; a host-based merchant may not need to explicitly close the batch since the batch is generally closed at predetermined intervals automatically by the processor.

For more information on terminal-based merchant, please refer to "Understanding Terminal Based Merchant" in the *Oracle iPayment Concepts and Procedures Guide*.

Signature

```
Procedure OraPmtCloseBatch (p_api_version          IN   NUMBER,
                            p_init_msg_list        IN   VARCHAR2 := FND_API.G_
                                                        FALSE,
                            p_commit               IN   VARCHAR2 := FND_API.G_
                                                        FALSE,
                            p_validation_level     IN   NUMBER :=  FND_API.G_VALID_
                                                        LEVEL_FULL,
                            p_ecapp_id             IN   NUMBER,
                            p_batchtrxn_rec        IN   BatchTrxn_rec_type,
                            x_return_status        OUT  VARCHAR2,
                            x_msg_count            OUT  NUMBER,
                            x_msg_data             OUT  VARCHAR2,
                  x_closebatchrespsum_rec          OUT  BatchRespSum_rec_type,
                  x_closebatchrespdet_tbl          OUT  BatchRespDet_tbl_type
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_batchtrxn_rec	IN	BatchTrxn_rec_type		Required
	IN	PmtMode	VARCHAR2	Required
		PmtType	VARCHAR2	Optional
	IN	Settlement_Date	DATE	Required if PmtMode is OFFLINE
	IN	Payee_ID	VARCHAR2	Required
	IN	MerchBatch_ID	VARCHAR2	Required
	IN	BEP_Suffix	VARCHAR2	Required
	IN	BEP_Account	VARCHAR2	Required
	IN	NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_closebatchrespsum_rec	OUT	BatchRespSum_rec_type		
	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	
(OFFLINE MODE RELATED RESPONSE)	OUT	OffLineResp	OffLineResp_rec_ type	
	OUT	EarliestSettlement_Date	DATE	
		Scheduled_Date	DATE	
	OUT	NumTrxns	NUMBER	
	OUT	MerchBatch_ID	VARCHAR2	
	OUT	BatchState	NUMBER	
	OUT	BatchDate	DATE	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
	OUT	Payee_ID	VARCHAR2	
	OUT	Credit_Amount	NUMBER	
	OUT	Sales_Amount	NUMBER	
	OUT	Batch_Total	NUMBER	
	OUT	Currency	VARCHAR2	
	OUT	VpsBatch_ID	VARCHAR2	
	OUT	GWBatch_ID	VARCHAR2	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
x_closebatchrespdet_ tblN.B.: All detail records name-value pairs will have '-n' suffixed to show the index value 'n'	OUT	BatchRespDet_tbl_type		
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	Status	NUMBER	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
	OUT	NLSLANG	VARCHAR2	

OraPmtQueryBatch

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API provides an interface to query the status of any previous batch of operations by providing the Batch ID (that is, MerchBatch_ID) as part of the input. Payment Mode is always 'ONLINE' for this operation.

Signature

```
Procedure OraPmtQueryBatch (p_api_version          IN    NUMBER,
                            p_init_msg_list        IN    VARCHAR2 := FND_API.G_
                                                         FALSE,
                            p_commit               IN    VARCHAR2 := FND_API.G_
                                                         FALSE,
                            p_validation_level     IN    NUMBER :=FND_API.G_
                                                         VALID_LEVEL_FULL,
                            p_ecapp_id             IN    NUMBER,
                            p_batchtrxn_rec        IN    BatchTrxn_rec_type,
                            x_return_status        OUT   VARCHAR2,
                            x_msg_count            OUT   NUMBER,
                            x_msg_data             OUT   VARCHAR2,
                    x_qrybatchrespsum_rec          OUT   BatchRespSum_rec_type,
                    x_qrybatchrespdet_tbl          OUT   BatchRespDet_tbl_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_batchtrxn_rec	IN	BatchTrxn_rec_type		Required
	IN	PmtMode	VARCHAR2	Required

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
(will be NULL since always PmtMode =‘ONLINE’)	IN	Settlement_Date	DATE	Mandatory if PmtMode is OFFLINE
	IN	Payee_ID	VARCHAR2	Required
	IN	MerchBatch_ID	VARCHAR2	Required
	IN	BEP_Suffix	VARCHAR2	Required
	IN	BEP_Account	VARCHAR2	Required
	IN	NLS_LANG	VARCHAR2	Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_qrybatchrespsum_rec	OUT	BatchRespSum_rec_type		
	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	
	OUT	NumTrxns	NUMBER	
		MerchBatch_ID	VARCHAR2	
		BatchState	NUMBER	
		BatchDate	DATE	
		Payee_ID	VARCHAR2	
		Credit_Amount	NUMBER	
		Sales_Amount	NUMBER	
		Batch_Total	NUMBER	
		Currency	VARCHAR2	
		VpsBatch_ID	VARCHAR2	
		GWBatch_ID	VARCHAR2	
		ErrorLocation	NUMBER	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
		BEPErrCode	VARCHAR2	
		BEPErrMessage	VARCHAR2	
x_qrybatchrespdet_ tblN.B.: All detail records name-value pairs will have '-n' suffixed to show the index value 'n'	OUT	BatchRespDet_tbl_type		
	OUT	Trxn_ID	NUMBER	
	OUT	Trxn_Type	NUMBER	
	OUT	Trxn_Date	DATE	
	OUT	Status	NUMBER	
	OUT	ErrorLocation	NUMBER	
	OUT	BEPErrCode	VARCHAR2	
	OUT	BEPErrMessage	VARCHAR2	
	OUT	NLS_LANG	VARCHAR2	

OraPmtInq

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API provides high-level payment information such as Payee, Payer, Instrument, and Tangible related information. It can be used when all the information regarding a payment is needed. So an EC-App which does not store all the payment related information locally, can invoke this API to find all the information pertaining to the payment operation. Typically used to display the information to the end user for editing in case of OFFLINE operation in an application like internet payments.

It takes in the ECApp ID and the transaction ID as input parameters.

Signature

```
Procedure OraPmtInq(p_api_version          IN    NUMBER,
                    p_init_msg_list        IN    VARCHAR2 := FND_API.G_FALSE,
```

p_commit	IN	VARCHAR2 := FND_API.G_FALSE,
p_validation_level	IN	NUMBER := FND_API.G_
		VALID_LEVEL_FULL,
p_ecapp_id	IN	NUMBER,
p_tid	IN	NUMBER,
x_return_status	OUT	VARCHAR2,
x_msg_count	OUT	NUMBER,
x_msg_data	OUT	VARCHAR2,
x_inqresp_rec	OUT	InqResp_rec_type)

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
p_api_version	IN	NUMBER	-	Required
p_init_msg_list	IN	VARCHAR2	-	Optional
p_commit	IN	VARCHAR2	-	Optional
p_validation_level	IN	NUMBER	-	Optional
p_ecapp_id	IN	NUMBER	-	Required
p_tid	IN	NUMBER	-	Required
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_inqresp_rec	OUT	InqResp_rec_type		
(GENERIC PAYMENT SERVER RESPONSE)	OUT	Response	Response_rec_type	
		Status	NUMBER	
		ErrCode	VARCHAR2	
		ErrMessage	VARCHAR2	
		NLS_LANG	VARCHAR2	

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
(INQUIRY OPERATION RELATED RESPONSE)				
	OUT	Payer	Payer_rec_type	
		Payer_ID	VARCHAR2	
		Payer_Name	VARCHAR2	
	OUT	Payee	Payee_rec_type	
		Payee_ID	VARCHAR2	
	OUT	Tangible	Tangible_rec_type	Required
		Tangible_ID	VARCHAR2	Required
		Tangible_Amount	NUMBER	Required
		Currency_Code	VARCHAR2	Required
		RefInfo	VARCHAR2	Optional
		Memo	VARCHAR2	Optional
		Acct_Num	VARCHAR2	Optional
		OrderMedium	VARCHAR2	Optional
		EFTAuthMethod	VARCHAR2	Optional

Parameter	IN/ OUT	DataType	SubType	Required/ Optional
	OUT	PmtInstr	PmtInstr_rec_type	
		PmtInstr_ID		
		PmtInstr_ShortName		
		CreditCardInstr	CreditCardInstr_rec_type	
			CC_Num	
			CC_ExpDate	
			CC_BillingAddr.Address1	
			CC_BillingAddr.Address2	
			CC_BillingAddr.Address3	
			CC_BillingAddr.City	
			CC_BillingAddr.County	
			CC_BillingAddr.State	
			CC_BillingAddr.Country	
			CC_BillingAddr.PostalCode	
			CC_Type	
			CC_HolderName	
			FIName	
		BankAcctInstr	BankAcctInstr_rec_type	
			Bank_ID	
			BankAcct_Num	
			BankAcct_Type	
			Branch_ID	
			FIName	
			BankAcct_HolderName	

OraRiskEval

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API performs risk evaluation without using transactions. For more information on using this API for evaluating risk, please refer to Appendix A: Risk Management.

Note: This API is also available in an overloaded form, with Address Verification System (AVS). The AVS version of the API includes an additional input parameter, p_avs_risk_info. All the other inputs and outputs are identical to the API without AVS.

Signature

Procedure OraRiskEval (p_api_version IN NUMBER,

p_init_msg_list IN VARCHAR2 := FND_API.G_FALSE,

p_commit IN VARCHAR2 := FND_API.G_FALSE,

p_validation_level IN NUMBER := FND_API.G_VALID_LEVEL_FULL,

p_ecapp_id IN NUMBER,

p_payment_risk_info IN PaymentRiskInfo_rec_type,

x_return_status OUT VARCHAR2,

x_msg_count OUT NUMBER,

x_msg_data OUT VARCHAR2,

x_risk_resp OUT RiskResp_rec_type)

Overloaded API Signature (with AVS information):

Procedure OraRiskEval (p_api_version IN NUMBER,

p_init_msg_list IN VARCHAR2 := FND_API.G_FALSE,

p_commit IN VARCHAR2 := FND_API.G_FALSE,

p_validation_level IN NUMBER := FND_API.G_VALID_LEVEL_FULL,

p_ecapp_id IN NUMBER,

p_avs_risk_info IN AVSRiskInfo_rec_type,

x_return_status OUT VARCHAR2,

x_msg_count OUT NUMBER,

x_msg_data OUT VARCHAR2,

x_risk_resp OUT RiskResp_rec_type)

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_ecapp_id	IN	NUMBER		Required
p_avs_risk_info	IN	AVSRiskInfo_rec_type		Required for Overloaded API
		Formula Name	VARCHAR2	Optional
		Payee_ID	NUMBER	Required
		Previous_Risk_Score	VARCHAR2	Required
		AVSCode	VARCHAR2	Required
p_payment_risk_ info	IN	PaymentRiskInfo_rec_ type		Required
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_risk_resp	OUT	RiskResp_rec_type		

OraPmtBankPayBatchReq

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

The OraPmtBankPayBatchReq API handles new bank payment batch requests from Oracle Payables. The Oracle Payables application passes the payment batch details as a PmtBankPayBatchReq_Rec_Type record, transaction details as a PmtBankPayBatchTrxn_ Tbl_Type table structure and invoice details as a PmtBankPayInvoice_Tbl_Type table structure to iPayment. This is an OFFLINE batch submission wherein the batch information is stored in the iPayment database.

If the batch payment information is saved in the database successfully, then the OUT status field x_return_status returns a value of FND_API.G_RET_STS_SUCCESS.

Signature

PROCEDURE OraPmtBankPayBatchReq(

p_api_version IN NUMBER,

p_init_msg_list IN VARCHAR2 DEFAULT FND_API.G_FALSE,

p_commit IN VARCHAR2 DEFAULT FND_API.G_FALSE,

p_validation_level IN NUMBER DEFAULT FND_API.G_VALID_LEVEL_FULL,

p_ecapp_id IN NUMBER,

x_return_status OUT VARCHAR2,

x_msg_count OUT NUMBER,

x_msg_data OUT VARCHAR2,

p_pmt_bs_req_rec IN PmtBankPayBSReq_Rec_Type,

p_pmt_batch_req_rec IN PmtBankPayBatchReq_Rec_Type,

p_pmt_batch_trxn_tbl IN PmtBankPayBatchTrxn_Tbl_Type,

p_pmt_invoice_tbl IN PmtBankPayInvoice_Tbl_Type)

Parameters

Parameter	IN/OUT	DataType	Sub Type	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_ecapp_id	IN	NUMBER		Required
p_pmt_bs_req_rec	IN	PmtBankPayBSReq_Rec_Type		Required
p_pmt_batch_req_rec	IN	PmtBankPayBatchReq_Rec_Type		Required

Parameter	IN/OUT	DataType	Sub Type	Required/Optional
p_pmt_batch_trxn_tbl	IN	PmtBankPayBatchTrxn_Tbl_Type		Required
p_pmt_invoice_tbl	IN	PmtBankPayInvoice_Tbl_Type		Required
x_return_status	OUT	VARCHAR2		Required
x_msg_count	OUT	NUMBER		Required
x_msg_data	OUT	VARCHAR2		Required

Payment Instrument Registration APIs

Instrument registration APIs provide the functionality to register a payor's bank, credit card, PINless debit card, or purchase card. All the procedures described below are declared public and are stored in the PL/SQL Package IBY_INSTRREG_PUB as part of the applications database.

The following PL/SQL APIs are described in this section:

- OraInstrAdd
- OraInstrmMod
- OraInstrDel
- OraInstrInq

OraInstrAdd

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API can be used to add an instrument to the iPayment. Only one of Credit Card, PINless Debit Card, Purchase Card or Bank Account can be registered at a time.

If the registration is successful, an Instrument Id is returned. This Instrument Id may be used to submit a payment transaction. For Bank Account transfers, you need to have a registered instrument id to submit a transaction. This APIs will internally call IBY_BANKACCT_ PKG. createBankAcct orIBY_CREDITCARD_PKG.createCard to register a new instrument.

Signature

```
Procedure OraInstrAdd (p_api_version      IN    NUMBER,
                       p_init_msg_list    IN    VARCHAR2:=FND_API.G_FALSE
                       p_commit           IN    VARCHAR2:=FND_API.G_FALSE
                       p_validation_level IN    NUMBER:=FND_API.G_VALID
                                                LEVEL_FULL
                       p_payer_id         IN    VARCHAR2(80),
                       p_pmtInstrRec      IN    PmtInstr_rec_type,
```

x_return_status	OUT	VARCHAR2,
x_msg_count	OUT	NUMBER,
x_msg_data	OUT	VARCHAR2,
x_instr_id	OUT	NUMBER(15))

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_payer_id	IN	VARCHAR2		Required
p_pmtInstrRec	IN	PmtInstr_rec_type		Required
		1. CreditCardInstr	CreditCardInstr_ rec_type	Mandatory if 2, 3 and 4 are not passed.
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3.CC_Type	Optional
			4.CC_Num	Required
			5.CC_ExpDate	Required
			6.CC_ HolderName	Optional
			7.Billing_ Address1	Optional*
			8. Billing_ Address2	Optional
			9. Billing_ Address3	Optional
			10.Billing_City	Optional*
			11. Billing_ County	Optional

Parameter	IN/ OUT	DataType	SubType	Required/Optional
			12. Billing_State	Optional
			13. Billing_ Country	Optional*
Note: Address record is optional overall, but if passed, then the 3 fields Addr1, City, Country (7,10,13)* are together mandatory.			14. Billing_ PostalCode	Optional
			15. CC_Desc	Optional
		2.PurchaseCardInstr	PurchaseCardInstr_rec_type	Mandatory if 1, 3 and 4 are null
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3.PC_Type	Optional
			4.PC_Num	Required
			5.PC_ExpDate	Required
			6.PC_ HolderName	Optional
			7.Billing_ Address1	Optional*
			8. Billing_ Address2	Optional
			9. Billing_ Address3	Optional
			10.Billing_City	Optional*
			11. Billing_ County	Optional
			12. Billing_State	Optional
			13. Billing_ Country	Optional*

Parameter	IN/ OUT	DataType	SubType	Required/Optional
Note: Address record is optional overall, but if passed, then the 3 fields Addr1, City, Country (7,10,13)* are together mandatory.			14. Billing_ PostalCode	Optional
			15, PC_Subtype	Required
			16. PC_Desc	Optional
		3.DebitCardInstr	DebitCardInstr_ rec_type	Mandatory if 1, 2 and 4 are null
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3.DC_Type	Optional
			4.DC_Num	Required
			5.DC_ExpDate	Required
			6.DC_ HolderName	Optional
			7.Billing_ Address1	Optional*
			8. Billing_ Address2	Optional
			9. Billing_ Address3	Optional
			10.Billing_City	Optional*
			11. Billing_ County	Optional
			12. Billing_State	Optional
			13. Billing_ Country	Optional*

Parameter	IN/ OUT	DataType	SubType	Required/Optional
Note: Address record is optional overall, but if passed, then the 3 fields Addr1, City, Country (7,10,13)* are together mandatory.			14. Billing_ PostalCode	Optional
			15. DC_Subtype	Required
			16. DC_Desc	Optional
		4. BankAcctInstr	BankAcctInstr_ rec_type	Mandatory if 1, 2 and 3 are both null
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3. Bank_Id	Required
			4. Branch_ID	Optional
			5. BankAcct_ Type	Required
			6. BankAcct_ Num	Required
			7. BankAcct_ HolderName	Required
			8. Bank_Desc	Optional
		5. InstrumentType	VARCHAR2	Required.
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_instr_id	OUT	NUMBER		

OraInstrmMod

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API can be used to modify an instrument in the iPayment. Only one instrument of type Credit Card, PINless Debit Card, Purchase Card or Bank Account can be modified at a time. This APIs will internally call IBY_BANKACCT_PKG. modifyBankAcc or IBY_CREDITCARD_PKG.modifyCard to modify an existing instrument.

Note: The instrument record in the database is updated with the input parameters on an "as is" basis. Since the default value for all the input parameters (or record type members) is NULL, the record will be updated with null values for parameters (or members) not assigned a value. This means that each time an instrument's information is modified, it is REPLACED with all the information passed in the modification request. That is, all the prior information is overwritten by the data in the modification request, assuming that the data passed is the newest.

Signature

Procedure OraInstrMod (p_api_version		IN	NUMBER,
	p_init_msg_list	IN	VARCHAR2:=FND_API.G_FALSE
	p_commit	IN	VARCHAR2:=FND_API.G_FALSE
	p_validation_level	IN	NUMBER:=FND_API.G_VALID
			LEVEL_FULL
	p_payer_id	IN	VARCHAR2(80),
	p_pmtInstrRec	IN	PmtInstr_rec_type,
	x_return_status	OUT	VARCHAR2,
	x_msg_count	OUT	NUMBER,
	x_msg_data	OUT	VARCHAR2)

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_payer_id	IN	VARCHAR2		Required
p_pmtInstrRec	IN	PmtInstr_rec_type		Required
		1. CreditCardInstr	CreditCardInstr_ rec_type	Mandatory if 2, 3 and 4 are not passed.
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3.CC_Type	Optional
			4.CC_Num	Required
			5.CC_ExpDate	Required
			6.CC_ HolderName	Optional
			7.Billing_ Address1	Optional*
			8. Billing_ Address2	Optional
			9. Billing_ Address3	Optional
			10.Billing_City	Optional*
			11. Billing_ County	Optional
			12. Billing_State	Optional
			13. Billing_ Country	Optional*

Parameter	IN/ OUT	DataType	SubType	Required/Optional
Note: Address record is optional overall, but if passed, then the 3 fields Addr1, City, Country (7,10,13)* are together mandatory.			14. Billing_ PostalCode	Optional
			15. CC_Desc	Optional
		2.PurchaseCardInstr	PurchaseCardInstr_rec_type	Mandatory if 1, 3 and 4 are null
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3.PC_Type	Optional
			4.PC_Num	Required
			5.PC_ExpDate	Required
			6.PC_ HolderName	Optional
			7.Billing_ Address1	Optional*
			8. Billing_ Address2	Optional
			9. Billing_ Address3	Optional
			10.Billing_City	Optional*
			11. Billing_ County	Optional
			12. Billing_State	Optional
			13. Billing_ Country	Optional*

Parameter	IN/ OUT	DataType	SubType	Required/Optional
Note: Address record is optional overall, but if passed, then the 3 fields Addr1, City, Country (7,10,13)* are together mandatory.			14. Billing_ PostalCode	Optional
			15, PC_Subtype	Required
			16. PC_Desc	Optional
		3.DebitCardInstr	DebitCardInstr_ rec_type	Mandatory if 1, 2 and 4 are null
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3.DC_Type	Optional
			4.DC_Num	Required
			5.DC_ExpDate	Required
			6.DC_ HolderName	Optional
			7.Billing_ Address1	Optional*
			8. Billing_ Address2	Optional
			9. Billing_ Address3	Optional
			10.Billing_City	Optional*
			11. Billing_ County	Optional
			12. Billing_State	Optional
			13. Billing_ Country	Optional*

Parameter	IN/ OUT	DataType	SubType	Required/Optional
Note: Address record is optional overall, but if passed, then the 3 fields Addr1, City, Country (7,10,13)* are together mandatory.			14. Billing_ PostalCode	Optional
			15. DC_Subtype	Required
			16. DC_Desc	Optional
		4. BankAcctInstr	BankAcctInstr_ rec_type	Mandatory if 1, 2 and 3 are both null
			1.Instr_Id	Should NOT be passed.
			2.FIName	Optional
			3. Bank_Id	Required
			4. Branch_ID	Optional
			5. BankAcct_ Type	Required
			6. BankAcct_ Num	Required
			7. BankAcct_ HolderName	Required
			8. Bank_Desc	Optional
		5. InstrumentType	VARCHAR2	Required.
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		

OraInstrDel

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API can be used to delete an instrument from the iPayment. Only one instrument of type Credit Card, PINless Debit Card, Purchase Card or Bank Account can be deleted at a time. This APIs will finally call IBY_BANKACCT_PKG. deleteBankAcct orIBY_ CREDITCARD_PKG.deleteCreditCard to delete an existing instrument.

Note: This is a soft delete. The record is not removed or deleted physically from the database, the instrument status is made inactive.

Signature

```
Procedure OraInstrDel (p_api_version        IN    NUMBER,
                       p_init_msg_list      IN    VARCHAR2:=FND_API.G_FALSE
                       p_commit             IN    VARCHAR2:=FND_API.G_FALSE
                       p_validation_level   IN    NUMBER:=FND_API.G_VALID
                                                  LEVEL_FULL
                       p_payer_id           IN    VARCHAR2(80),
                       p_instr_id           IN    NUMBER(15),
                       x_return_status      OUT   VARCHAR2,
                       x_msg_count          OUT   NUMBER,
                       x_msg_data           OUT   VARCHAR2)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_payer_id	IN	VARCHAR2		Required
p_instr_id	IN	NUMBER		Required

Parameter	IN/ OUT	DataType	SubType	Required/Optional
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		

OraInstrInq

API type: Public

Prerequisites for calling the API: None

Function(s) performed by the API:

This API can be used to inquire about an instrument in the iPayment. This API will have 2 overloaded procedures. The provides flexibility to the calling applications. The two available flavours are:

a. This inquiry is based on the payer Id and will return all the instruments that are registered for that payer. Three tables, each containing instruments of the same type will be returned as output.

b. This inquiry is based on the Instrument Id and will return details for the instrument that is registered for that Instrument Id and the instrument type. UNREGISTERED is returned when the instrument does not exist for the given payer_id and instr_id.

Signature (with only payer id)

```
Procedure OraInstrInq (p_api_version        IN    NUMBER,
                       p_init_msg_list      IN    VARCHAR2:=FND_API.G_FALSE,
                       p_commit             IN    VARCHAR2:=FND_API.G_FALSE,
                       p_validation_level         IN NUMBER:=FND_API.G_VALID
                                                             LEVEL_FULL,
                       p_payer_id                 IN      VARCHAR2(80),
                       x_return_status            OUT     VARCHAR2,
                       x_msg_count                OUT     NUMBER,
                       x_msg_data                 OUT     VARCHAR2,
                       x_creditcard_tbl           OUT     CreditCard_tbl_type,
```

x_purchasecard_tbl OUT PurchaseCard_tbl_type,

x_bankacct_tbl OUT BankAcct_tbl_type)

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_payer_id	IN	VARCHAR2		Required
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_creditcard_tbl	OUT	CreditCard_tbl_type	CreditCardInstr_ rec_type 1.Instr_Id 2.FIName 3.CC_Type 4.CC_Num 5.CC_ExpDate 6.CC_ HolderName 7.Billing_ Address1 8. Billing_ Address2 9. Billing_ Address3 10.Billing_City 11. Billing_ County	

Parameter	IN/ OUT	DataType	SubType	Required/Optional
			12. Billing_State	
			13. Billing_ Country	
			14. Billing_ PostalCode	
			15. CC_Desc	
x_purchasecard_tbl	OUT	PurchaseCard_tbl_type	PurchaseCardInstr_rec_type	
			1.Instr_Id	
			2.FIName	
			3.PC_Type	
			4.PC_Num	
			5.PC_ExpDate	
			6.PC_ HolderName	
			7.Billing_ Address1	
			8. Billing_ Address2	
			9. Billing_ Address3	
			10.Billing_City	
			11. Billing_ County	
			12. Billing_State	
			13. Billing_ Country	
			14. Billing_ PostalCode	
			15, PC_Subtype	
			16. PC_Desc	

Parameter	IN/ OUT	DataType	SubType	Required/Optional
x_bankacct_tbl	OUT	BankAcct_tbl_type	BankAcctInstr_ rec_type	
			1.Instr_Id	
			2.FIName	
			3. Bank_Id	
			4. Branch_ID	
			5. BankAcct_ Type	
			6. BankAcct_ Num	
			7. BankAcct_ HolderName	
			8. Bank_Desc	

Overloaded API Signature (using instrument id)

```
Procedure OraInstrInq (p_api_version          IN    NUMBER,
                       p_init_msg_list        IN    VARCHAR2:=FND_API.G_
                                                    FALSE,
                       p_commit               IN    VARCHAR2:=FND_API.G
                                                    _FALSE,
                       p_validation_level     IN    NUMBER:=FND_API.G
                                                    VALID_LEVEL_FULL,
                       p_payer_id             IN    VARCHAR2(80),
                       p_instr_id             IN    VARCHAR2,
                       x_return_status        OUT   VARCHAR2,
                       x_msg_count            OUT   NUMBER,
                       x_msg_data             OUT   VARCHAR2,
                       x_pmtInstrRec          OUT   PmtInstr_rec_type)
```

Parameters

Parameter	IN/ OUT	DataType	SubType	Required/Optional
p_api_version	IN	NUMBER		Required
p_init_msg_list	IN	VARCHAR2		Optional
p_commit	IN	VARCHAR2		Optional
p_validation_level	IN	NUMBER		Optional
p_payer_id	IN	VARCHAR2		Required
p-instr_id	IN	NUMBER		
x_return_status	OUT	VARCHAR2		
x_msg_count	OUT	NUMBER		
x_msg_data	OUT	VARCHAR2		
x_pmtInstrRec	OUT	PmtInstr_rec_type		
		1. CreditCardInstr	CreditCardInstr_ rec_type	
			1.Instr_Id	
			2.FIName	
			3.CC_Type	
			4.CC_Num	
			5.CC_ExpDate	
			6.CC_ HolderName	
			7.Billing_ Address1	
			8. Billing_ Address2	
			9. Billing_ Address3	
			10.Billing_City	

Parameter	IN/ OUT	DataType	SubType	Required/Optional
			11. Billing_ County	
			12. Billing_State	
			13. Billing_ Country	
			14. Billing_ PostalCode	
			15. CC_Desc	
		2. PurchaseCardInstr	PurchaseCardIns tr_rec_type	
			1.Instr_Id	
			2.FIName	
			3.PC_Type	
			4.PC_Num	
			5.PC_ExpDate	
			6.PC_ HolderName	
			7.Billing_ Address1	
			8. Billing_ Address2	
			9. Billing_ Address3	
			10.Billing_City	
			11. Billing_ County	
			12. Billing_State	
			13. Billing_ Country	
			14. Billing_ PostalCode	
			15, PC_Subtype	

Parameter	IN/ OUT	DataType	SubType	Required/Optional
			16. PC_Desc	
		3.DebitCardInstr	DebitCardInstr_ rec_type	
			1.Instr_Id	
			2.FIName	
			3.DC_Type	
			4.DC_Num	
			5.DC_ExpDate	
			6.DC_ HolderName	
			7.Billing_ Address1	
			8. Billing_ Address2	
			9. Billing_ Address3	
			10.Billing_City	
			11. Billing_ County	
			12. Billing_State	
			13. Billing_ Country	
			14. Billing_ PostalCode	
			15. DC_Subtype	
			16. DC_Desc	
		4. BankAcct_tbl_type	BankAcctInstr_ rec_type	
			1.Instr_Id	
			2.FIName	
			3. Bank_Id	

Parameter	IN/ OUT	DataType	SubType	Required/Optional
			4. Branch_ID	
			5. BankAcct_ Type	
			6. BankAcct_ Num	
			7. BankAcct_ HolderName	
			8. Bank_Desc	
		5. Instrument Type	VARCHAR2. Can have following values defined as constants	
			C_ INSTRTYPE_ UNREG,	
			C_ INSTRTYPE_ BANKACCT,	
			C_ INSTRTYPE_ CREDITCARD,	
			C_ INSTRTYPE_ PURCHASECARD	

PL/SQL Record/Table Types Definitions

The following PL/SQL record/table types are defined to store the objects (entities) necessary for the ECApp PL/SQL APIs. For information on Mandatory, Conditionally Mandatory, and Optional fields in these records/tables, please refer to the ensuing API descriptions, where these requirements are tabulated.

Payments Related Generic Record Types

1. TYPE **Payee_rec_type** IS RECORD (

Payee_ID	VARCHAR2(80)

);

Payee_ID: ID of the payee

2. TYPE **Payer_rec_type** IS RECORD (

Payer_ID	VARCHAR2(80),
Payer_Name	VARCHAR2(80)

);

Payer_ID: ID of the payer

Payee_Name: Name of the payer

3. TYPE **Address_rec_type** IS RECORD (

Address1	VARCHAR2(80),
Address2	VARCHAR2(80),
Address3	VARCHAR2(80),
City	VARCHAR2(80),
County	VARCHAR2(80),
State	VARCHAR2(80),
Country	VARCHAR2(80),
PostalCode	VARCHAR2(40),
Phone	VARCHAR2(40),
Email	VARCHAR2(40)

);

Address1: The first line of the street address.

Address2: The second line of the street address.

Address3: The third line of the street address.

City: City in the address.

State: State in the address.

County: County in the address.

Country: Country code in the address.

Postalcode: Postal code for the address.

Phone: Phone for that address. It is for informational purposes only.

Email: It is not supported right now.

4. TYPE **CreditCardInstr_rec_type** IS RECORD (

FIName	VARCHAR2(80),
CC_Type	VARCHAR2(80),
CC_Num	VARCHAR2(80),
CC_ExpDate	DATE,
CC_HolderName	VARCHAR2(80),
CC_BillingAddr	Address_rec_type

);

Financial Institution Name (FIName): *Optional*, should be at least of non-trivial length 3.

CC_Type: Type of credit card (MASTERCARD, VISA, AMEX, …).

CC_Num: For *credit card number*, it should be numeric other than dashes and spaces. However, it will be stored without any spaces or dashes.

CC_ExpDate: Credit Card expiration date.

CC_HolderName: Credit card holder name.

CC_BillingAddr: Address type record for the billing address of the credit card.

5. TYPE **PurchaseCardInstr_rec_type** IS RECORD (

FIName	VARCHAR2(80),
PC_Type	VARCHAR2(80),
PC_Num	VARCHAR2(80),
PC_ExpDate	DATE,
PC_HolderName	VARCHAR2(80),
PC_BillingAddr	Address_rec_type,

PC_Subtype VARCHAR2(80)

);

Financial Institution Name (FIName): *Optional*, should be at least of non-trivial length 3.

PC_Type: Type of purchase card (MASTERCARD, VISA, AMEX, …).

PC_Num: For *purchase card number*, it should be numeric other than dashes and spaces. However, it will be stored without any spaces or dashes.

PC_ExpDate: Purchase Card expiration date.

PC_HolderName: Purchase card holder name.

PC_BillingAddr: Address type record for the billing address of the purchase card.

PC_Subtype: The subtype for purchase card. Possible values are ('B'/'C'/'P'/'U') which are for BUSINESS / CORPORATE / PURCHASE / UNKNOWN.

Financial Institution Name (FIName): *Optional*.

Bank_ID: Routing number of the bank. Should be at least of non-trivial length 2.

Branch_ID: ID of the branch.

BankAcct_Type: Should be of at least non-trivial length 3. Such as CHECKING.

BankAcct_Num: For *bank account number*, should be at least of non-trivial length 3.

BankAcct_HolderName: Name of the bank account holder.

6. TYPE DebitCardInstr_rec_type IS RECORD (

FIName VARCHAR2(80),

DC_Type VARCHAR2(80),

DC_Num VARCHAR2(80),

DC_ExpDate Date,

DC_HolderName VARCHAR2(80),

DC_BillingAddr Address_rec_type,

DC_Subtype VARCHAR2(80)

);

Financial Institution Name (FIName): *Optional*, should be at least of non-trivial length 3.

DC_Type: Type of debit card (MASTERCARD, VISA, AMEX, …).

DC_Num: For *debit card number*, it should be numeric other than dashes and spaces. However, it will be stored without any spaces or dashes.

DC_ExpDate: Debit Card expiration date.

DC_HolderName: Debit card holder name.

DC_BillingAddr: Address type record for the billing address of the purchase card.

DC_Subtype: The subtype for debit card. Possible values are ('B'/'C'/'P'/'U') which are for BUSINESS / CORPORATE / PURCHASE / UNKNOWN.

7. TYPE **PmtInstr_rec_type** IS RECORD (

PmtInstr_ID	NUMBER,
PmtInstr_ShortName	VARCHAR2(80),
CreditCardInstrCredit	CardInstr_rec_type,
PurchaseCardInstr	PurchaseCardInstr_rec_type,
DualPaymentInstr	DualPaymentInstr_rec_type,
DebitCardInstr	DebitCardInstr_rec_type

);

PmtInstr_ID: The payment instrument ID of an already registered payment instrument.

PmtInstr_ShortName: Short name for the payment instrument.

CreditCardInstr: Credit card instrument type record. Refer #4 for details.

PurchaseCardInstr: Purchase card instrument type record. Refer #5 for details.

DualPaymentInstr: Payment instrument type record.

DebitCardInstr: Debit card instrument type record. Refer #6 for details.

Note: The Payment Instrument Type (i.e., CREDITCARD / DEBIT CARD / PURCHASECARD / BANKACCOUNT / UNREGISTERED) is derived from the input data, by verifying which of the input instrument records (i.e., CreditCardInstr, PurchaseCardInstr, DebitCardInstr, BankAcctInstr, PmtInstr_ID) are provided with input values. That particular instrument type and its component fields are then passed to the iPayment11i EC-Servlet. So, either PmtInstr_ID alone is provided for registered instruments, or one of the other three (CreditCardInstr, PurchaseCardInstr, BankAcctInstr) is provided as part of payment instrument input.

8. TYPE **DualPaymentInstr_rec_type** IS RECORD (

PmtInstr_ID	NUMBER,

```
    PmtInstr_ShortName      VARCHAR2(80),
    BnfPmtInstr_ID          NUMBER,
    BnfPmtInstr_ShortName   VARCHAR2(80),
);
```

PmtInstr_ID: The payment instrument ID of an already registered payment instrument.

PmtInstr_ShortName: Short name for the payment instrument.

BnfPmtInstr_ID: The payment instrument ID of a registered bank account instrument that is the beneficiary of the transaction.

BnfPmtInstr_ShortName: Short name for the a registered bank account instrument.

9. TYPE **Tangible_rec_type** IS RECORD (

```
    Tangible_ID         VARCHAR2(80),
    Tangible_Amount     NUMBER,
    Currency_Code       VARCHAR2(80),
    RefInfo             VARCHAR2(80),
    Memo                VARCHAR2(80),
    Acct_Num            VARCHAR2(80),
    OrderMedium         VARCHAR2(80),
    EFTAuthMethod       VARCHAR2(80),
);
```

Tangible_ID: It is the order id or bill id. It should be unique for a given payee.

Tangible_Amount: Should be a positive number.

Currency_Code: The 3 letter currency code.

RefInfo: Reference information for this bill/order.

Memo: Memo for this bill/order.

Acct_Num: Account number of the customer, if applicable.

OrderMedium: This parameter indicates the medium or channel through which a transaction was created. It is used for credit card, purchase card, and debit card transactions to obtain an improved interchange rate.

EFTAuthMethod: This parameter indicates the method used to receive authorization to perform an electronic funds transfer to debit a payer's bank account.

10. SUBTYPE **RetailData_Enum** IS VARCHAR2(10);

11. TYPE **RetailData_rec_type** IS RECORD (

Tangible_ID	VARCHAR2(80),
IsRetail	VARCHAR2(1),
POSEntryMode	RetailData_Enum,
POSCapability	RetailData_Enum,
POSAuthSource	RetailData_Enum,
POSCardIdMethod	RetailData_Enum,
POSSwipeData	VARCHAR2(300)

Tangible_ID: It is the order id or bill id. It should be unique for a given payee.

IsRetail: Value 'Y' indicates the current transaction is a retail transaction; value 'N' that it is not.

POSEntryMode: Gives the credit card entry mode at the point-of-sale (POS). The following constants are have been enumerated for this field:

C_ENTRYMODE_KEYED: Manual/keyed entry.

C_ENTRYMODE_MAGTRACK1: Magnetic reader track 1.

C_ENTRYMODE_MAGTRACK2: Magnetic reader track 2.

C_ENTRYMODE_MAGTRACKALL: Magnetic reader all tracks (track 1 & 2).

C_ENTRYMODE_SMARTCARD_RDR: Smart card reader/chip reader.

C_ENTRYMODE_UNKNOWN: Unknown entry mode.

POSCapability: The card reading capabilities at the point-of-sale. This field takes the following enumerated values:

C_CAPABILITY_KEY: Keyed/manual entry-only capability.

C_CAPABILITY_MAG_RDR: Magnetic reader capability.

C_CAPABILITY_CHIP_RDR: Chip reader capability.

C_CAPABILITY_UNKNOWN: Unknown capability.

POSAuthSource: The authorization source. This field takes the following enumerated values:

C_AUTHSRC_ISSUER_PROVIDED: Issuer provided authorization source.

C_AUTHSRC_REFERRAL: Referral authorization source.

C_AUTHSRC_OFFLINE: Off-line authorization.

C_AUTHSRC_NONAPPROVED: Non-approved.

POSCardIdMethod: The card identification method used at the point-of-sale. The field can have the following enumerated values:

C_CARDID_SIGNATURE: Signature identification.

C_CARDID_PIN: PIN-entry identification.

C_CARDID_UNATTEND_TERM: Unattended terminal identification.

C_CARDID_MAILORDER: Mail order identification.

C_CARDID_NONE: No identification.

POSSwipeData: Swipe data read by a magnetic or chip reader at the point-of-sale. A calling application that can interface with such a reader may pass this data to iPayment as a (possibly encoded) string.

Inbound Payment Operations Related Record Types

1. TYPE **PmtReqTrxn_rec_type** IS RECORD (

PmtMode	VARCHAR2(30),
CVV2	VARCHAR2(10) := NULL,
Settlement _Date	Date:=,
Auth_Type	VARCHAR2(80),
Check_Flag	VARCHAR2(30),
Retry_Flag	VARCHAR2(30),
Org_ID	NUMBER,
NLS_LANG	VARCHAR2(80),
PONum	NUMBER,
TaxAmount	NUMBER,
ShipFromZip	VARCHAR2(80),
ShipToZip	VARCHAR2(80),
AnalyzeRisk	VARCHAR2(80)
AuthCode	VARCHAR2(255)
VoiceAuthFlag	VARCHAR2(30)

);

PmtMode: Its value should be either ONLINE or OFFLINE.

CVV2: The Visa CVV2, Mastercard CVC2, or American Express CIP value associated with the credit card is used for this transaction.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

Check flag: Ignored for ONLINE requests, optional for OFFLINE requests. It is meaningful only for OFFLINE Bank Account transfer operations when the user requested settle date is earlier THAN the earliest date it can be settled by the system. When check flag is set to true, the operation will be rejected if it cannot be settled by user specified settle date, otherwise, the operation will get scheduled with the earliest settle date available by the system, and a warning message will be returned saying unable to meet user specified date.

Retry flag: Should be either 'Y' or 'N'.

Applicable for ONLINE Credit Card Request and Credit operations.

You should set this flag to 'Y' when previous request when the same operation may have been processed by the back payment system. For example, when first request returns with a time out status, or when OraPmtQryTrxn failed to retrieve the information. This flag is passed as is to the backend payment system. Check with individual backend payment system for further details.

Org_ID: The identifier for the organization submitting the request.

Applicable for new operations (Request, Modify, Credit). Should be a positive integer.

Auth_Type: Applicable for credit card authorization (request), modify, and credit operation only. Also applicable for electronic funds transfer online validations. Takes one of the following values:

AUTHONLY: terminal-based/host-based authorization only.

AUTHCAPTURE: host-based authorization and capture together.

VALIDATE: EFT online validations.

NLSLang: The NLS language code.

PONum: Purchase order number for this transaction.

TaxAmount: Amount of transaction that is tax.

ShipFromZip: The ZIP code from which merchandise will be shipped.

ShipToZip: The ZIP code to which merchandise will be shipped.

AnalyzeRisk: The flag that allows the calling application to request risk analysis. Values are True, False, and Neutral. "True" causes risk analysis to be done and "False" keeps risk analysis from being done. "Neutral" causes iPayment to use the payee level risk enablement setting. This defaults to "neutral" if no value is given.

AuthCode: The authorization Code that the financial institution issues after doing a voice authorization. This field is required if the VoiceAuthFlag is set to ‘Y’.

VoiceAuthFlag: Should be set up to either Y or N. This indicates whether the current transaction refers to a voice authorization (where the financial institution has already been contacted directly). If this field is set up as ‘Y’, then the AuthCode field is required to have the same value.

2. TYPE **ModTrxn_rec_type** IS RECORD (

Trxn_ID NUMBER,

PmtMode	VARCHAR2(30),
Settlement_Date	DATE,
Check_Flag	VARCHAR2(30),
Auth_Type	VARCHAR2(80),
PONum	NUMBER,
TaxAmount	NUMBER,
ShipFromZip	VARCHAR2(80),
ShipToZip	VARCHAR2(80)

);

Trxn_ID: The transaction id for the operation which has to be modified.

PmtMode: Its value should be either ONLINE or OFFLINE.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

Check flag: Ignored for ONLINE requests, optional for OFFLINE requests. It is meaningful only for OFFLINE operations when the user requested settle date is *earlier* than the earliest date it can be settled by the system. When check flag is set to true, the operation will be rejected if it cannot be settled by user specified settle date, otherwise, the operation will get scheduled with the earliest settle date available by the system, and a warning message will be returned saying unable to meet user specified date.

Auth_Type: Applicable for credit card authorization (request), modify, and credit operation only. Also applicable for electronic funds transfer online validations. Takes one of the following values:

AUTHONLY: terminal-based/host-based authorization only.

AUTHCAPTURE: host-based authorization and capture together.

VALIDATE: EFT online validations.

PONum: Purchase order number for this transaction.

TaxAmount: Amount of transaction that is tax.

ShipFromZip: The ZIP code from which merchandise will be shipped.

ShipToZip: The ZIP code to which merchandise will be shipped.

3. TYPE **CaptureTrxn_rec_type** IS RECORD (

Trxn_ID	NUMBER,
PmtMode	VARCHAR2(30),
Settlement_Date	DATE,
Currency	VARCHAR2(80),
Price	NUMBER,
NLS_LANG	VARCHAR2(80)

);

Trxn_ID: The transaction id for the operation which has to be captured.

PmtMode: Its value should be either ONLINE or OFFLINE.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

Currency: Should be a 3-letter code.

Price: Should be a positive amount. The amount of money to be captured.

NLSLang: The NLS language code

4. TYPE **ReturnTrxn_rec_type** IS RECORD (

Trxn_ID	NUMBER,
PmtMode	VARCHAR2(30),
Settlement_Date	DATE,
Currency	VARCHAR2(80),
Price	NUMBER,
NLS_LANG	VARCHAR2(80)

);

Trxn_ID: The transaction id for the operation which has to be returned.

PmtMode: Its value should be either ONLINE or OFFLINE.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

Currency: Should be a 3-letter code.

Price: Should be a positive amount. The amount of money to be captured.

NLSLang: The NLS language code

5. TYPE **CancelTrxn_rec_type** IS RECORD (

Trxn_ID	NUMBER,
Req_Type	VARCHAR2,
NLS_LANG	VARCHAR2(80)

);

Trxn_ID: The transaction id for the operation which has to be returned.

Req_Type: optional field provides the option of canceling other operations (such as Void, Return, etc.), in addition to scheduled payment requests. By Default, this Req_Type field is set to 'ORAPMTREQ' to cancel the authorization operation.

NLSLang: The NLS language code

6. TYPE **QueryTrxn_rec_type** IS RECORD (

Trxn_ID	NUMBER,
History_Flag	VARCHAR2(30),
NLS_LANG	VARCHAR2(80)

);

Trxn_ID: The transaction id for the operation which has to be queried.

History_Flag: takes in values => 'TRUE' or 'FALSE'. When set to TRUE, it retrieves the entire history, otherwise it retrieves the latest one only.

NLSLang: The NLS language code

7. TYPE **VoidTrxn_rec_type** IS RECORD (

Trxn_ID	NUMBER,
PmtMode	VARCHAR2(30),
Settlement_Date	DATE,
Trxn_Type	NUMBER,
NLS_LANG	VARCHAR2(80)

);

Trxn_ID: The transaction id for the operation which has to be voided. The type of the operation will be specified in Trxn_Type.

PmtMode: Its value should be either ONLINE or OFFLINE.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

NLSLang: The NLS language code.

Trxn_Type: takes the following numeric values:

Lookup Code	Meaning	Description
2	AuthOnly	Online authorization requested for an order
3	AuthCapture	Online authorization & capture for an order
4	VoidAuthOnly	Void an order authorized but not captured
5	Return	Return on an order which is authorized & captured
6	ECRefund	Refund on a purchase done using EC cash/coin
7	VoidAuthCapture	Voids a previously authorized & captured trxn
8	Capture	Capture funds for previously authorized trxn.
9	MarkCapture	Marked for capture by terminal based system
10	MarkReturn	Marked for return by terminal based system
11	Credit	Refund money to customer
13	VoidCapture	Void operation captured by host based system
14	VoidMarkCapture	Void operation marked for capture by terminal based system
17	VoidReturn	Void return operation for host based system
18	VoidMarkReturn	Void operation marked for return by terminal based system
102	Batch Admin	Used for open, purge, query, and close batch operations

8. TYPE **CreditTrxn_rec_type** IS RECORD (

PmtMode	VARCHAR2(30),
Settlement_Date	DATE,
Retry_Flag	VARCHAR2(30),
Org_ID	NUMBER,

NLS_LANG	VARCHAR2(80)

);

PmtMode: Its value should be either ONLINE or OFFLINE.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

Retry flag: Should be either 'Y' or 'N'.

Applicable for ONLINE Credit Card Request and Credit operations.

You should set this flag to 'Y' when previous request with the same operation may have been processed by the back payment system. For example, when first request returns with a time out status, or when OraPmtQryTrxn failed to retrieve the information. This flag is passed as is to the backend payment system. Check with individual backend payment system for further details.

Org_ID: The identifier for the organization submitting the request.

NLSLang: The NLS language code

9. TYPE **BatchTrxn_rec_type** IS RECORD (

PmtMode	VARCHAR2(30),
PmtType	VARCHAR2(30),
Settlement_Date	DATE,
Payee_ID	NUMBER,
MerchBatch_ID	VARCHAR2(80),
BEP_Suffix	VARCHAR2(80),
BEP_Account	VARCHAR2(80),
NLS_LANG	VARCHAR2(80)

);

PmtMode: Its value should be either ONLINE or OFFLINE.

PmtType: Optional, defaulted to empty string. You need specify it if you wish to operate on a back end payment system rather than the default one.

Settlement_Date: Ignored for all ONLINE requests, required for OFFLINE requests. It is the date by which you wish the operation to be settled.

Payee_ID: It's the payee identifier for whom the batch operation is performed.

MerchBatch_ID: It's the user selected identifier for this operation. Should be a non-empty string, and should be unique across all merchant batch ids from a particular payee.

BEP_Suffix: The 3-letter suffix of the payment system that is associated with this batch.

BEP_Account: The merchant account the batch is associated with. This is the same value as the payee's payment system identifier for the given back end payment system.

NLSLang: The NLS language code.

Outbound Bank Payment Batch Related Record Types

```
TYPE PmtBankPayInvoice_Rec_Type IS RECORD
(
  pmt_batchrequestid NUMBER,
  pmt_trxnid NUMBER,
  inv_number VARCHAR2(50),
  inv_date DATE,
  inv_amount NUMBER,
  pmt_amount NUMBER);
```

pmt_batchrequestid: Payment-batch Request ID.

pmt_trxnid: Transaction ID.

inv_number: Invoice Number.

inv_date: Invoice Date.

inv_amount: Invoice Amount.

pmt_amount: Actual payment amount after discount deduction.

```
TYPE PmtBankPayInvoice_Tbl_Type IS TABLE OF
PmtBankPayInvoice_Rec_Type INDEX BY BINARY_INTEGER;
TYPE PmtBankPayBatchTrxn_Rec_Type IS RECORD (
 pmt_batchrequestid NUMBER,
 pmt_trxnid NUMBER,
 pmt_loc_country VARCHAR2(10),
 pmt_priority VARCHAR2(10),
 debit_acctno VARCHAR2(30),
 fax_mail_to_name VARCHAR2(80),
 mail_address1 VARCHAR2(35),
 mail_address2 VARCHAR2(35),
```

mail_address3 VARCHAR2(35),
mail_addr_city VARCHAR2(35),
mail_addr_state VARCHAR2(35),
mail_addr_zip VARCHAR2(35),
value_amount NUMBER,
ref_amount NUMBER,
value_date DATE,
charge_details VARCHAR2(6),
vat_amount NUMBER,
wht_amount NUMBER,
pmt_details_1 VARCHAR2(20),
pmt_details_2 VARCHAR2(20),
pmt_details_3 VARCHAR2(20),
pmt_details_4 VARCHAR2(20),
trxn_code_clrid VARCHAR2(6),
bnf_name VARCHAR2(80),
bnf_id VARCHAR2(30),
bnf_address_line1 VARCHAR2(30),
bnf_addr_city VARCHAR2(35),
bnf_addr_state VARCHAR2(35),
bnf_addr_zip VARCHAR2(35),
bnf_addr_phone VARCHAR2(20),
bnf_remarks VARCHAR2(80),
bnf_charge_amt NUMBER,
bnf_adjustment_amt VARCHAR2(30),
bnf_bank_name VARCHAR2(30),
bnf_bank_address VARCHAR2(60),
bnf_bank_addr_state VARCHAR2(35),

bnf_bank_addr_city VARCHAR2(35),

bnf_bank_addr_zip VARCHAR2(20),

bnf_bank_branch_no VARCHAR2(50),

bnf_bank_branch_name VARCHAR2(30),

bnf_bank_branch_type VARCHAR2(25),

bnf_acctno VARCHAR2(30),

bnf_bank_account_name VARCHAR2(80),

bnf_bank_acct_type VARCHAR2(25),

bnf_acct_type VARCHAR2(25),

bnf_faxno_cableaddr VARCHAR2(35),

bnf_site_code VARCHAR2(15),

bnf_swift_code VARCHAR2(30),

bnf_bank_clearing_mtd VARCHAR2(60),

bnf_taxpayer_id VARCHAR2(30),

future_pay_due_date DATE,

cust_ref NUMBER,

intrm_bankcode VARCHAR2(30),

intrm_swiftcode VARCHAR2(30),

intrm_bankname VARCHAR2(60),

intrm_bank_faxno_cbl VARCHAR2(35)}

pmt_batchrequestid: Payment-batch Request ID.

pmt_trxnid: Transaction ID.

pmt_loc_country: Payment Location ISO Country Code.

pmt_priority: Payment Priority.

debit_acctno: Account number to be debited.

fax_mail_to_name: Fax/Mail to Name - Only required if fax beneficiary advice is required.

mail_address1: Mail to Address 1 - Only required if fax beneficiary advice is required.

mail_address2: Mail to Address 2 - Only required if fax beneficiary advice is required.

mail_address3: Mail to Address 3 - Only required if fax beneficiary advice is required.

mail_addr_city: Mail to City - Only required if fax beneficiary advice is required.

mail_addr_state: Mail to State - Only required if fax beneficiary advice is required.

mail_addr_zip: Mail to Postal Code - Only required if fax beneficiary advice is required.

value_amount: Amount in the payment document.

ref_amount: Amount in the reference currency.

value_date: Date in the payment document.

charge_details: Details of any charges on the invoice.

vat_amount: Amount of the value added tax.

wht_amount: Withholding tax amount.

pmt_details_1: Payment Details 1.

pmt_details_2: Payment Details 2.

pmt_details_3: Payment Details 3.

pmt_details_4: Payment Details 4.

trxn_code_clrid: Transaction Code/Clearing ID.

bnf_name: Beneficiary Name.

bnf_id: Beneficiary ID.

bnf_address_line1: Beneficiary Address.

bnf_addr_city: Beneficiary City.

bnf_addr_state: Beneficiary State.

bnf_addr_zip: Beneficiary Zip.

bnf_addr_phone: Beneficiary Phone.

bnf_remarks: Remarks.

bnf_charge_amt: Beneficiary Charges Amount.

bnf_adjustment_amt: Adjustment Amount.

bnf_bank_name: Beneficiary Bank Name.

bnf_bank_address: Beneficiary Bank Address.

bnf_bank_addr_state: Beneficiary Bank State.

bnf_bank_addr_city: Beneficiary Bank City.

bnf_bank_addr_zip: Beneficiary Bank Postal Code.

bnf_bank_branch_no: Beneficiary Bank/Branch No.

bnf_bank_branch_name: Beneficiary Bank Branch Name.

bnf_bank_branch_type: Beneficiary Bank Branch Type.

bnf_acctno: Beneficiary Bank Account Number.

bnf_bank_account_name: Beneficiary Bank Account Name.

bnf_bank_acct_type: Beneficiary Bank Account Type i.e Checking or Saving.

bnf_acct_type: Bank account type code. Possible values are: INTERNAL or SUPPLIER for banks defined for Oracle Payables.

bnf_faxno_cableaddr: Beneficiary Fax No/Cable Address - Only required if fax beneficiary advice is required.

bnf_site_code: Beneficiary Site Code.

bnf_swift_code: Beneficiary Bank (SWIFT Code).

bnf_bank_clearing_mtd: Beneficiary Bank Clearing Method.

bnf_taxpayer_id: Beneficiary Tax ID.

future_pay_due_date: Future pay date.

cust_ref: Customer Reference.

intrm_bankcode: Intermediary Bank Code.

intrm_swiftcode: Intermediary Bank (SWIFT Code).

intrm_bankname: Intermediary Bank Name.

intrm_bank_faxno_cbl: Intermediary Bank Fax No/Cable Address.

TYPE PmtBankPayBatchTrxn_Tbl_Type IS TABLE OF

```
PmtBankPayBatchTrxn_Rec_Type INDEX BY BINARY_INTEGER;
TYPE PmtBankPayBatchReq_Rec_Type IS RECORD(
  pmt_batchrequestid NUMBER,
  pmt_batch_name VARCHAR2(80),
  cust_id   NUMBER,
  cust_name VARCHAR2(60),
  cust_addr1 VARCHAR2(60),
  cust_addr2 VARCHAR2(60),
  cust_addr3 VARCHAR2(60),
  bank_name VARCHAR2(60),
  bank_branch_name VARCHAR2(60),
  bank_acct_name VARCHAR2(60),
  bank_acct_number VARCHAR2(30),
  bank_acct_type VARCHAR2(30),
  orig_country_code VARCHAR2(25),
  pmt_method VARCHAR2(10),
  currency_code VARCHAR2(15),
  ref_currency_code VARCHAR2(15),
  doc_order_lookup_code VARCHAR2(25),
  no_of_trxns NUMBER,
  batch_total NUMBER,
  request_date DATE
);
```

pmt_batchrequestid: Payment-batch Request ID.

pmt_batch_name: Payment-batch name.

cust_id: Customer ID.

cust_name: Customer Name.

cust_addr1: Customer Address line1.

cust_addr2: Customer Address line2.

cust_addr3: Customer Address line3.

bank_name: Originating Bank Name.

bank_branch_name: Branch name of the originating bank.

bank_acct_name: Bank account name specific to the bank and branch.

bank_acct_number: Bank account number specific to the bank and branch.

bank_acct_type: Bank account type, i.e whether checking or saving.

orig_country_code: Originating ISO Country Code.

pmt_method: Payment Method - CHECK/ELECTRONIC/WIRE.

currency_code: Currency Code.

ref_currency_code: Reference Currency Code.

doc_order_lookup_code: Type of payment ordering in a batch.

no_of_trxns: Total number of transactions in the batch.

batch_total: Sum total of all the transaction amounts in the batch.

request_date: Date when the batch has been submitted.

```
TYPE PmtBankPayBSReq_Rec_Type IS RECORD(
  pmt_batchset_id       NUMBER,
  no_of_batches         NUMBER
);
```

pmt_batchset_id: Payment batch-set request ID.

no_of_batches: Total number of payment-batches in the batch-set.

Risk Management Record Types

1. TYPE **RiskInfo_rec_type** IS RECORD (

Formula_Name	VARCHAR2(80),
ShipToBillTo_Flag	VARCHAR2(255),
Time_Of_Purchase	VARCHAR2(80),
Customer_Acct_Num	NUMBER

);

Formula_Name: Name of the formula to be used.

ShipToBillTo_Flag: used to notify whether the “Ship_To” and the “Bill_To” addresses match or not (‘TRUE’/’FALSE’).

Time_Of_Purchase: Represents the time duration passed in ‘HH:MI’ format in 24 Hours notation. For example, 11 pm will be denoted as ‘23:00’.

Customer_Acct_Num: Represents the payer’s account number in Oracle Accounts Receivables. This field is needed in AR - risk factors evaluation.

Note: For more information on using Risk Management, please refer to the documentation for the “Integrating Risk Management” under the section “Implementing iPayment”.

Inbound Payment Operations Response Record/Table Types

1. TYPE **Response_rec_type** IS RECORD (

Status	NUMBER,
ErrCode	VARCHAR2(80),
ErrMessage	VARCHAR2(255),
NLS_LANG	VARCHAR2(80)

);

Status: The status for the request. Possible values are (0,1,2 or 3).

ErrCode: The IBY_XXXX error code for the error, if any.

ErrMessage: The error message associated with the error.

NLS_LANG: The NLS code.

NOTE: This record is included in all the responses and the status of the operation can be found by looking at the value of status. Possible values for Status are: (0 => 'Success', 1=> 'Information', 2=> 'Warning', 3=> 'Error').

For more information on Error Codes and their meaning, please refer to "Error Handling during Payment Processing" in this document.

2. TYPE **OffLineResp_rec_type** IS RECORD (

EarliestSettlement_Date	DATE,
Scheduled_Date	DATE

);

If the payment operation cannot be settled by the settlement date specified in input, due to lead time of the back end payment system, then

EarliestSettlement_Date: Specifies the earliest date by which the operation can be settled

Scheduled_Date: Specifies the date on which scheduler will pick up the operation.

The **OffLineResp_rec_type** record outputs can be looked into for payment operations sent in OFFLINE Mode.

For more information on how the status values are propagated back to the ECApp, please refer to "Status Update API for Offline Request" in this document.

3. TYPE **RiskResp_rec_type** IS RECORD (

Status	NUMBER,
ErrCode	VARCHAR2(80),
ErrMessage	VARCHAR2(255),
Additional_ErrMessage	VARCHAR2(255),
Risk_Score	NUMBER,
Risk_Threshold_Val	NUMBER,
Risky_Flag	VARCHAR2(30)

);

Status: The status for the request. Possible values are (0,1,2 or 3).

ErrCode: The IBY_XXXX error code for the error, if any.

ErrMessage: The error message associated with the error.

Additional_ErrMessage: If multiple factors have failed, this field contains additional messages about why the factors failed.

Risk_Score: Represents the overall risk score of the payment request.

Risk_Threshold_Val: The threshold value that is set for the payee involved in the payment request.

Risky_Flag: Indicates whether payment is risky or not.

4. TYPE **ReqResp_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
RiskRespIncluded	VARCHAR2(30),
RiskResponseRisk	Resp_rec_type,
Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
Authcode	VARCHAR2(80),
RefCode	VARCHAR2(80),
AVSCode	VARCHAR2(80),

```
    PmtInstr_Type       VARCHAR2(80),
    Acquirer            VARCHAR2(80),
    VpsBatch_ID         VARCHAR2(80),
    AuxMsg              VARCHAR2(255),
    ErrorLocation       NUMBER,
    BEPErrCode          VARCHAR2(80),
    BEPErrMessage       VARCHAR2(255)
);
```

Response: The response record. Refer #1 for details.

OffLineResp: The offline response record. Refer to #2 for details.

RiskRespIncluded: Flag used to indicate whether risk response included or not. Possible values ('YES'/'NO')/.

RiskResponse: The risk response record. Refer to #3 for details.

Trxn_ID: The new id generated for this request.

Trxn_Type: The type of the capture operation. Back-end system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

AuthCode: Authorization code that is returned by back end payment system.

RefCode: Reference code that is returned by back end payment system.

AVSCode: AVS code that is returned by back end payment system.

PmtInstr_Type: Credit card type of the operation, such as 'Visa'.

Acquirer: Acquirer information that is returned by back end payment system.

VPSBatch_ID: VPSBatchId that is returned by back end payment system.

AuxMsg: Auxiliary message that is returned by back end payment system.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

Note: RiskRespIncluded is a flag ('YES'/'NO') that tells the ECAPP that the RiskResponse Record contains some valid Risk response information.

5. TYPE **ModResp_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
Trxn_ID	NUMBER

);

Response: The response record. Refer to #1 for details.

OffLineResp: The offline response record. Refer to #2 for details.

Trxn_ID: The new id generated for this request.

6. TYPE **VoidResp_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
RefCode	VARCHAR2(80),
PmtInstr_Type	VARCHAR2(80),
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer to #1 for details.

OffLineResp: The offline response record. Refer to #2 for details.

Trxn_ID: The transaction id for this request.

Trxn_Type: The type of the capture operation. The Back-end system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

RefCode: Reference code that is returned by back end payment system.

PmtInstr_Type: Credit card type of the operation, such as 'Visa'.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

7. TYPE **CancelResp_rec_type** IS RECORD (

Response	Response_rec_type,
Trxn_ID	NUMBER,
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer #1 for details.

Trxn_ID: The transaction id for this request.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

8. TYPE **CaptureResp_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
PmtInstr_Type	VARCHAR2(80),
RefCode	VARCHAR2(80),

ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer to #1 for details.

OffLineResp: The offline response record. Refer to #2 for details.

Trxn_ID: The transaction id for this request.

Trxn_Type: The type of the capture operation. Backend system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

PmtInstr_Type: Credit card type of the operation, such as 'Visa'.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

9. TYPE **ReturnResp_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
PmtInstr_TypeV	ARCHAR2(80),
RefCode	VARCHAR2(80),
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer #1 for details.

OffLineResp: The offline response record. Refer #2 for details.

Trxn_ID: The transaction id for this request.

Trxn_Type: The type of the capture operation. Backend system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

PmtInstr_Type: Credit card type of the operation, such as 'Visa'.

RefCode: Reference code that is returned by the back end payment system.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

10. TYPE **CreditResp_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
PmtInstr_Type	VARCHAR2(80),
RefCode	VARCHAR2(80),
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer #1 for details.

OffLineResp: The offline response record. Refer #2 for details.

Trxn_ID: The transaction id for this request.

Trxn_Type: The type of the capture operation. Backend system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

PmtInstr_Type: Credit card type of the operation, such as 'Visa'.

RefCode: Reference code that is returned by the back end payment system.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

11. TYPE **InqResp_rec_type** IS RECORD (

Response	Response_rec_type,
Payer	Payer_rec_type,
Payee	Payee_rec_type,
Tangible	Tangible_rec_type,
PmtInstr	PmtInstr_rec_type

);

Response: The response record. Refer to C.4.4.#1 for details.

Payer: The payer record. Refer to C.4.4.#2 for details.

Payee: The payee record. Refer to C.4.4.#1 for details.

Tangible: The tangible record. Refer to C.4.4.#8 for details.

PmtInstr: The pmtinstr record. Refer to C.4.4.#7 for details.

12. TYPE **QryTrxnRespSum_rec_type** IS RECORD (

Response	Response_rec_type,
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer #1 for details.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

13. TYPE **QryTrxnRespDet_rec_type** IS RECORD (

Status	NUMBER,
StatusMsg	VARCHAR2(255),
Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
PmtInstr_Type	VARCHAR2(80),
Currency	VARCHAR2(80),
Price	NUMBER,
RefCode	VARCHAR2(80),
AuthCode	VARCHAR2(80),
AVSCode	VARCHAR2(80),
Acquirer	VARCHAR2(80),
VpsBatch_ID	VARCHAR2(80),
AuxMsg	VARCHAR2(255),
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Status: The status for this request

StatusMsg: The status message for this request.

Trxn_ID: The transaction id for this request.

Trxn_Type: The type of the capture operation. Backend system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

PmtInstr_Type: Credit card type of the operation, such as 'Visa'.

Currency: Should be a 3-letter code.

Price: Should be a positive amount. The amount of money to be captured.

RefCode: Reference code that is returned by back end payment system.

AuthCode: Authorization code that is returned by back end payment system.

AVSCode: AVS code that is returned by back end payment system.

Acquirer: Acquirer information that is returned by back end payment system.

VPSBatch_ID: VPSBatchId that is returned by back end payment system.

AuxMsg: Auxiliary message that is returned by back end payment system.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

14. TYPE **QryTrxnRespDet_tbl_type** IS TABLE OF QryTrxnRespDet_rec_type INDEX BY BINARY_INTEGER;

Inbound Batch Payment Operations Response Record/Table Types

1. TYPE **BatchRespSum_rec_type** IS RECORD (

Response	Response_rec_type,
OffLineResp	OffLineResp_rec_type,
NumTrxns	NUMBER,
MerchBatch_ID	VARCHAR2(80),
BatchState	NUMBER,
BatchDate	DATE,
Credit_Amount	NUMBER,
Sales_Amount	NUMBER,
Batch_Total	NUMBER,
Payee_ID	VARCHAR2(80),
VpsBatch_ID	VARCHAR2(80),
GWBatch_ID	VARCHAR2(80),
Currency	VARCHAR2(80),
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255)

);

Response: The response record. Refer #1 for details.

OffLineResp: The offline response record. Refer #2 for details.

NumTrxns: Total number of individual operations in this batch.

Merch Batch_ID: Merchant-specified unique batch id for this batch operation

BatchState: The state of the batch operation.

BatchDate: The date of the batch operation.

Credit_Amount: Total amount of credits.

Sales_Amount: Total amount of charges.

Batch_Total: Total amount of the entire batch.

VPSBatch_ID: VPSBatchId returned by the backend payment system.

GWBatch_ID: GWBatchId returned by the backend payment system.

Currency: The currency code used.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system.

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

2. TYPE **BatchRespDet_rec_type** IS RECORD (

Trxn_ID	NUMBER,
Trxn_Type	NUMBER,
Trxn_Date	DATE,
Status	NUMBER,
ErrorLocation	NUMBER,
BEPErrCode	VARCHAR2(80),
BEPErrMessage	VARCHAR2(255),
NLS_LANG	VARCHAR2(80)

);

Trxn_ID: The transaction id for this request.

Trxn_Type: The type of the capture operation. The Back-end system may distinguish between Capture and MarkCapture.

Trxn_Date: The date of the operation.

Status: The status for this request.

ErrorLocation: The error location, if applicable. It is a number which indicates what place the error has occurred, like middle tier or the back end payment system.

BEPErrCode: The error code, if applicable, returned by the back end payment system

BEPErrMessage: The error message, if applicable, returned by the back end payment system.

NLSLang: The NLS language code

3. TYPE **BatchRespDet_tbl_type** IS TABLE OF **BatchRespDet_rec_type** INDEX BY BINARY_INTEGER;

Instrument Registration Related Record Types

This section describes the record/table definitions used in the Instrument Registration API.

Note: **CreditCardInstr_rec_type** and **PurchaseCardInstr_rec_type** defined in this section are different than ones defined in IBY_PAYMENT_ADAPTER_PUB. The record types defined in this section do NOT have Address_rec_type as a member.

---

1. TYPE **CreditCardInstr_rec_type** IS RECORD (

Instr_Id	NUMBER(15),
FIName	VARCHAR2(80),
CC_Type	VARCHAR2(80),
CC_Num	VARCHAR2(80),
CC_ExpDate	DATE,
CC_HolderName	VARCHAR2(80),
CC_Desc	VARCHAR2(240),
Billing_Address1	VARCHAR2(80),
Billing_Address2	VARCHAR2(80),
Billing_Address3	VARCHAR2(80),
Billing_City	VARCHAR2(80),
Billing_County	VARCHAR2(80),
Billing_State	VARCHAR2(80),
Billing_Country	VARCHAR2(80),
Billing_PostalCode	VARCHAR2(40));

Instr_Id: Unique identifier for the instrument.

Financial Institution Name (FIName): Optional, should be at least of non-trivial length 3.

CC_Type: Type of credit card (MASTERCARD, VISA, AMEX, …)

CC_Num: This should be numeric other than dashes and spaces.

CC_ExpDate: Credit Card expiration date.

CC_HolderName: Credit card holder name.

CC_Desc: Descriptions/Comments, if any.

Billing_Address1: The first line of the street address.

Billing_Address2: The second line of the street address.

Billing_Address3: The third line of the street address.

Billing_City: City in the address.

Billing_State: State in the address.

Billing_County: County in the address.

Billing_Country: Country code in the address.

Billing_Postalcode: Postal code for the address.

2. TYPE **PurchaseCardInstr_rec_type** IS RECORD (

Instr_Id	NUMBER(15),
FIName	VARCHAR2(80),
PC_Type	VARCHAR2(80),
PC_Num	VARCHAR2(80),
PC_ExpDate	DATE,
PC_HolderName	VARCHAR2(80),
PC_Subtype	VARCHAR2(80),
PC_Desc	VARCHAR2(240),
Billing_Address1	VARCHAR2(80),
Billing_Address2	VARCHAR2(80),
Billing_Address3	VARCHAR2(80),
Billing_City	VARCHAR2(80),
Billing_County	VARCHAR2(80),
Billing_State	VARCHAR2(80),
Billing_Country	VARCHAR2(80),
Billing_PostalCode	VARCHAR2(40));

Instr_Id: Unique identifier for the instrument.

Financial Institution Name (FIName): Optional, should be at least of non-trivial length 3.

PC_Type: Type of credit card (MASTERCARD, VISA, AMEX, …)

PC_Num: This should be numeric other than dashes and spaces.

PC_ExpDate: Credit Card expiration date.

PC_HolderName: Credit card holder name.

PC_Subtype: The subtype for purchase card. Possible values are ('B'/'C'/'P'/'U') which are for BUSINESS / CORPORATE / PURCHASE / UNKNOWN.

PC_Desc: Descriptions/Comments, if any.

Billing_Address1: The first line of the street address.

Billing_Address2: The second line of the street address.

Billing_Address3: The third line of the street address.

Billing_City: City in the address.

Billing_State: State in the address.

Billing_County: County in the address.

Billing_Country: Country code in the address.

Billing_Postalcode: Postal code for the address.

3. TYPE **BankAcctInstr_rec_type** IS RECORD (

Instr_Id	NUMBER(15),
FIName	VARCHAR2(80),
Bank_Id	VARCHAR2(25),
Branch_Id	VARCHAR2(30),
BankAcct_Type	VARCHAR2(80),
BankAcct_Num	VARCHAR2(80),
BankAcct_HolderName	VARCHAR2(80),
Bank_Desc	VARCHAR2(240));

Instr_Id: Unique identifier for the instrument.

Financial Institution Name (FIName): Optional, should be at least of non-trivial length 3.

Bank_Id: Routing number of the bank. Should be at least of non-trivial length2. Typically the international bank identification number.

Branch_Id: Branch Number of the bank. Typically a national Branch Identification Code (BIC) number.

BankAcct_Type: Should be at least of non-trivial length3. Typical values could be ’CHECKING’, ’SAVING’.

BankAcct_Num: Account number in the branch of the bank.

BankAcct_HolderName: Name of the account holder.

Bank_Desc: Descriptions/Comments, if any.

4. TYPE **PmtInstr_rec_type** IS RECORD (

InstrumentType	VARCHAR2(80):C_INSTRTYPE_UNREG,
CreditCardInstr	CreditCardInstr_rec_type,
BankAcctInstr	BankAcctInstr_rec_type,
PurchaseCardInstr	PurchaseCardInstr_rec_type)

I**nstrumentType**: This holds the type of instrument that is passed in the PmtInstr_rec_type. It should have one of values - 'CREDITCARD', 'PURCHASECARD' and 'BANKACCOUNT', when being passed as input. When this is passed as an output parameter, it may also have the value 'UNREGISTERED’ (when the instrument is not registered in iPayment). Use the constants defined to assign values to this.

CreditCardInstr: Credit card instrument type record. This is described above.

BankAcctInstr: Bank account instrument type record. This is described above.

PurchaseCardInstr: Purchase card instrument type. This is described above.

5. TYPE **CreditCard_tbl_type** IS TABLE OF CreditCardInstr_rec_type INDEX BY BINARY_INTEGER.
6. TYPE **PurchaseCard_tbl_type** IS TABLE OF PurchaseCardInstr_rec_type INDEX BY BINARY_INTEGER.
7. TYPE **BankAcct_tbl_type** IS TABLE OF BankAcctInstr_rec_type INDEX BY BINARY_INTEGER.

Sample PL/SQL Code

The following PL/SQL code helps you in understanding how iPayment PL/SQL APIs can be invoked. This example code invokes the Payment Request API using a credit card. It also passes risk related information for risk evaluation. After invoking the PL/SQL API, it prints out all the elements in the response objects.

```
DECLARE
p_api_versionNUMBER := 1.0;
--To initialize message list.
p_init_msg_listVARCHAR2(2000) := FND_API.G_TRUE;
p_commitVARCHAR2(2000) := FND_API.G_FALSE;
p_validation_levelNUMBER := FND_API.G_VALID_LEVEL_FULL;
p_ecapp_idNUMBER := 0;
p_payee_recIBY_PAYMENT_ADAPTER_PUB.Payee_rec_type;
p_payer_recIBY_PAYMENT_ADAPTER_PUB.Payer_rec_type;
p_pmtinstr_recIBY_PAYMENT_ADAPTER_PUB.PmtInstr_rec_type;
p_tangible_recIBY_PAYMENT_ADAPTER_PUB.Tangible_rec_type;
p_pmtreqtrxn_recIBY_PAYMENT_ADAPTER_PUB.PmtReqTrxn_rec_
type;
p_riskinfo_recIBY_PAYMENT_ADAPTER_PUB.RiskInfo_rec_type;
x_return_statusVARCHAR2(2000);-- output/return status
x_msg_countNUMBER;-- output message count
x_msg_dataVARCHAR2(2000);-- reference string for output
message text
x_reqresp_recIBY_PAYMENT_ADAPTER_PUB.ReqResp_rec_type;
-- request specific output
-- response object
l_msg_countNUMBER;
l_msg_dataVARCHAR2(2000);
BEGIN
-- Common inputs
p_ecapp_id := 66;-- iPayment generated ECAppID
-- Payee related inputs
p_payee_rec.Payee_ID := 'ipay-payee1';-- payee's ID
-- Payer related inputs
p_payer_rec.Payer_ID := 'ipay-cust1';-- payer's ID
p_payer_rec.Payer_Name := 'Cust1';-- Payer's (Customer's name)
-- Payment request operation related input
p_pmtreqtrxn_rec.PmtMode := 'ONLINE';-- Payment mode (Can be
--ONLINE/OFFLINE)
-- Tangible/Bill related inputs
p_tangible_rec.Tangible_ID := 'tangibleid1';-- Tangible ID / orderID
p_tangible_rec.Tangible_Amount := 25.50; -- Amount for the operation
```

```
p_tangible_rec.Currency_code := 'USD'; -- Currency for the operation
p_tangible_rec.RefInfo := 'test_refinfo3';
p_pmtreqtrxn_rec.Auth_Type := upper('authonly');-- request type
-- Payment instrument related inputs
p_pmtinstr_rec.CreditCardInstr.CC_Type := 'Visa';
-- payment instrument type
p_pmtinstr_rec.CreditCardInstr.CC_Num := '4111111111111111';
-- payment instrument number
p_pmtinstr_rec.CreditCardInstr.CC_ExpDate := to_char(sysdate+300);
-- payment instr. Expiration date
-- Risk related inputs
p_riskinfo_rec.Formula_Name := 'test3';-- Risk formula name
p_riskinfo_rec.ShipToBillTo_Flag := 'TRUE';
-- Flag showing if ship to address same as Bill to address
p_riskinfo_rec.Time_Of_Purchase := '08:45'-- Time of purchase
-- invoking the API
IBY_PAYMENT_ADAPTER_PUB.OraPmtReq(
p_api_version,
p_init_msg_list,
p_commit,
p_validation_level,
p_ecapp_id,
p_payee_rec,
p_payer_rec,
p_pmtinstr_rec,
p_tangible_rec,
p_pmtreqtrxn_rec,
p_riskinfo_rec,
x_return_status,
x_msg_count,
x_msg_data,
x_reqresp_rec);
END;
-- After invoking the API, printing/interpreting the results
-- API status response
-- The status for the API. The value of this status has to be used to
-- find out whether the call was successful or not.
dbms_output.put_line('x_return_status = ' || x_return_status);
-- Payment Request Related Response. Printing Only If Status Is Success
If(Char(X_Reqresp_Rec.Response.Status = ‘S’) Then
-- Offline Mode Related Response
If P_Pmtreqtrxn_Rec.Pmtmode = 'OFFLINE' Then
dbms_output.put_line('Transaction ID = ' || To_Char(X_Reqresp_Rec.Trxn_ID));
dbms_output.put_line (‘X_Reqresp_Rec.Offlineresp.Earliestsettlement_Date = ' ||
To_Char(X_Reqresp_Rec.Offlineresp.Earliestsettlement_Date));
```

```
dbms_output.put_line( 'X_Reqresp_Rec.Offlineresp.Scheduled_Date = ' ||To_Char(X_
Reqresp_Rec.Offlineresp.Scheduled_Date));
Else
dbms_output.put_line('Transaction ID = ' || To_Char(X_Reqresp_Rec.Trxn_ID));
dbms_output.put_line('X_Reqresp_Rec.Authcode = ' || X_Reqresp_Rec.Authcode);
dbms_output.put_line('X_Reqresp_Rec.Avscode = ' || X_Reqresp_Rec.Avscode);
dbms_output.put_line('---------------------------------');
-- Risk Related Response
If(X_Reqresp_Rec.Riskrespincluded = 'YES') Then
dbms_output.put_line('--------------------------');
dbms_output.put_line(' X_Reqresp_Rec.Riskresponse.Risk_Score=   '|| X_Reqresp_
Rec.Riskresponse.Risk_Score );
dbms_output.put_line(  'X_Reqresp_Rec.Riskresponse.Risk_Threshold_Val=   '||
Reqresp_Rec.Riskresponse.Risk_Threshold_Val);
Endif;
Endif;
End If;
-- printing the error messages, if any from the API message list.
for i in 1..x_msg_count loop
dbms_output.put('msg # '||to_char(i)|| fnd_msg_pub.get(i) );
dbms_output.new_line();
end loop;
EXCEPTION
when others then
dbms_output.put_line('In When others Exception');
dbms_output.put_line('SQlerr is :'||substr(SQLERRM,1,200));
end;
/
```

D

Back-End APIs for Gateways

This appendix explains the back-end APIs used by gateway servlets. Topics in this section include:

- Gateway Model Payment System Integration Model Overview
- Payment System Servlet Operations
- Authorization API
- Purchase Card Authorization API
- Voice Authorization API
- Authorization API Output Name-Value Pairs
- Capture API
- Void API
- Return/Credit API
- Close Batch API
- Query Transaction Status API
- Query Batch Status API
- Transaction Status and Messages
- Transaction Types and Transaction States

Gateway Model Payment System Integration Model Overview

iPayment provides a set of APIs for interfacing with the payment system servlets, including APIs for authorization, capture, return, void, close batch, query batch status, and query transaction status. iPayment makes requests to these APIs using HTTP.

This section provides information to enable SSL payment system servlet developers (those who perform traditional credit-card processing) to create an interface for communication between iPayment and their payment systems. Also provided is the information that iPayment sends to payment system servlets, and the format and method of passing the data.

Payment System Servlet Development Prerequisites

Before you build a payment system servlet, you will need a basic understanding of iPayment. For additional information, see *Oracle iPayment Concepts and Procedures Guide* to get an understanding of iPayment and its architecture.

Test Payment System Servlet

After building a payment system servlet, complete the following steps:

1. Add the payment system to iPayment by following the steps of Creating a New Payment System in the *Oracle iPayment Concepts and Procedures Guide.*
2. Test and refine your servlet.

Payment System Servlet Operations

To perform the Payment System Servlet API operations, iPayment passes data to the payment system servlet in the form of HTTP name-value pairs.

Servlet Virtual Path Mapping

The following example shows the name-value pair format:

```
http://host name:port/servlet virtual path
?name-value pair(1)
&name-value pair(2)
&name-value pair(n)
&name-value pair(n+1)
...
```

where:

host name	The name of the computer where the payment system is located, for example, payment.com.
port	The listener's port number
servlet virtual path	The virtual path to the payment system servlet. This must always end in `oramipp_xxx`, where `xxx` is the three letter suffix chosen for this payment system.

Authorization API

When the payment system servlet receives the authorization request from iPayment, it formats the request into the payment system's native format and requests that the payment system perform an online authorization. When the payment system returns the authorization result, the payment system servlet will reformat the response into the iPayment's format.

Authorization API Input Name-Value Pairs

This table describes the authorization API input name-value pairs. To perform the Authorization operation, use the name value pairs listed in this table:

Name	Value
OapfAction	Value=oraauth
OapfOrderId	Order number for the transaction. OapfOrderId can contain only letters, numbers, dashes, underlines, and dots.
OapfCurr	ISO 4217 three-letter currency code. For example, usd (US Dollar).
OapfPrice	Transaction amount in the format prescribed for the three-letter ISO 4217 currency code
OapfAuthType	The authorization type for the transaction: AuthOnly or AuthCapture. n Use AuthOnly transactions when customers purchase "hard goods." The funds for these transactions are not captured until after the goods are shipped. n Use AuthCapture transactions when customers purchase "soft goods" such as software "downloadable" from a Web page. The funds for these transactions are authorized and captured at the same time.
OapfPmtInstrID	Identification (card) number for the selected OapfPmtType
OapfPmtInstrExp	Expiration date for the selected OapfPmtType in the format MM/YY or MM/YYYY. The payment system servlet should be able to accept both formats.
OapfStoreId	Merchant or business identification. The maximum length is 80 characters. It may consist of an Id and a password in the following format: <StoreId>:<Password>
In addition to the values above, the following name-value pairs are also required if AVS is required (except for OapfPhone, OapfEmail, and OapfCnty):	
OapfCustName	The customer's name

Name	Value
OapfAddr1	The customer's billing address (1st line). The portion of the address before city, state, and zip code.
OapfAddr2	The customer's billing address (2nd line). The portion of the address before city, state, and zip code.
OapfAddr3	The customer's billing address (3rd line). The portion of the address before city, state, and zip code.
OapfCity	The customer's city name for billing
OapfCnty	The customer's county name for billing
OapfState	The customer's state for billing
OapfCntry	The customer's country for billing
OapfPostalCode	The customer's zip code for billing
OapfPhone	The customer's telephone number
OapfEmail	The customer's e-mail address
OapfRetry	Specifies if this operation is a retry. Values include yes or no. If this flag is incorrectly turned on, then the servlet should attempt this transaction a second time as a non-retry transaction.
OapfNlsLang	(Optional) Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Purchase Card Authorization API

The Purchase Card Authorization API is the same as the Authorization API, with the addition of a few parameters. To perform the Purchase Card Authorization operation, use name value pairs defined by the Authorization API, and the name value pairs described in this table:

Name	Value
OapfCommCard	The type of card being used for the transaction. Possible values are: n P for Purchase cards n C for Corporate cards n B for Business cards
OapfPONum	Purchase Order number
OapfTaxAmount	Tax amount
OapfShipToZip	The ZIP code to which merchandise is to be shipped
OapfShipFromZip	The ZIP code from which merchandise is to be shipped

Voice Authorization API

The Voice Authorization API is the same as the Authorization API or Purchase Card Authorization API, except that the value for OapfAction should be ‘oravoiceauth’ and a new field, OapfAuthCode is mandatory.

This table lists the voice authorization input name-value pairs. To perform a Voice Authorization operation, use name value pairs defined in the Authorization API or Purchase Card Authorization API, with the following changes and additions:

Name	Value
OapfAction	Value= oravoiceauth
OapfAuthCode	Authorization Code issued by the financial institution, when the voice authorization is done over the phone.

Authorization API Output Name-Value Pairs

Output served by the payment system to iPayment returns in the form of HTTP headers consisting of the name-value pairs listed in this table:

Name	Value
OapfOrderId	Order number for the transaction. OapfOrderId can contain only letters, numbers, dashes, underlines, and dots.
OapfTrxnType	The transaction type from the payment system. See "Transaction Types and Transaction States" for a list of values.
OapfStatus	The transaction status. See "OapfStatus" for more information.
OapfAuthcode	The string for the authorization (approval) code.
OapfTrxnDate	The time stamp showing when the transaction is processed in YYYYMMDDHHMMSS format.
OapfPmtInstrType	The payment instrument type. For example, Visa or MasterCard.
OapfErrLocation	The error location. See "OapfErrLocation" for more information.
OapfVendErrCode	The payment system error code. See the payment system documentation for more information.
OapfVendErrmsg	The payment system error message. See the payment system documentation for more information.
The following name-value pairs are optional:	
OapfAcquirer	Name of the acquirer or bank
OapfRefcode	The retrieval reference number
OapfAVScode	The AVS code
OapfAuxMsg	Additional message from the processor
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Note: If an optional field does not have a value, do not include the optional field in the header.

Capture API

iPayment invokes the Capture API to perform online capture of previously authorized transactions.

Capture API Input Name-Value Pairs

To perform the Capture operation, use the name-value pairs listed in this table:

Name	Value
OapfAction	Value = oracapture.
OapfOrderId	Order number for the transaction. OapfOrderId can contain only letters, numbers, dashes, underlines, and dots.
OapfPrice	Transaction amount in the format prescribed for the three-letter ISO 4217 currency code.
OapfCurr	ISO 4217 three-letter currency code. For example, usd (US Dollar).
OapfStoreId	Merchant or business identification. The maximum length is 26 characters.
The following name-value pairs are optional:	
OapfRetry	Specifies if this operation is a retry. Values include Yes or No.
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_ LANG environment variable.

Capture API Output Name-Value Pairs

Output served by the payment system to iPayment returns in the form of HTTP headers consisting of the name-value pairs listed in this table:

Name	Value
OapfStatus	The transaction status. See "OapfStatus" for more information.
OapfTrxnType	The transaction type from the payment system. See "Transaction Types and Transaction States" for a list of values.
OapfTrxnDate	The time stamp for the time when the transaction is processed. This is in YYYYMMDDHHMMSS format.
OapfErrLocation	The error location. See "OapfErrLocation" for more information.

Name	Value
OapfVendErrCode	The payment system error code. See the payment system documentation for more information.
OapfVendErrmsg	The payment system error message. See the payment system documentation for more information.
The following name-value pairs are optional:	
OapfRefcode	The retrieval reference number.
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Capture API for Terminal-Based Merchant

For a terminal-based merchant, the Capture operation marks the transaction for capture in the local batch. If the operation completes successfully, it returns the following parameters:

OapfStatus Set to `0000`.

OapfTrxnType Set to `MarkCapture, 9`

OapfTrxnDate Set to the appropriate transaction date.

If the operation fails, it returns the following parameters:

- OapfStatus
- OapfTrxnType
- OapfTrxnDate
- OapfErrLocation
- OapfVendErrCode
- OapfVendErrmsg

Capture API for Host-Based Merchant

For a host-based merchant, the Capture operation communicates with the processor to capture the transaction. If the operation completes successfully, it returns the following parameters:

OapfStatus	Set to `0000`.
OapfTrxnType	Set to `MarkCapture, 8`
OapfTrxnDate	Set to the appropriate transaction date.
OapfRefcode	Set to the appropriate retrieval reference number

If the operation fails, it returns:

- OapfStatus
- OapfTrxnType
- OapfTrxnDate
- OapfErrLocation
- OapfVendErrCode

Void API

The Void API allows the merchant or business to void the following transaction types:

- Credit transactions
- Return transactions
- Capture transactions

The Void API voids the most recent transaction type for an order. For example, the merchant or business performs authorization--and later capture-- for a transaction. If the merchant or business performs a void on this order, the capture transaction is voided.

Void API Input Name-Value Pairs

To perform the Void operation, use the name-value pairs listed in this table:

Name	Value
OapfAction	Value = oravoid.
OapfTrxnType	The transaction type to void from the payment system. See "Transaction Types and Transaction States" for a list of values.
OapfOrderId	Order number for the transaction. OapfOrderId can contain only letters, numbers, dashes, underlines, and dots.
OapfStoreId	Merchant or business identification. The maximum length is 26 characters.
The following name-value pairs are optional:	
OapfRetry	Specifies if this operation is a retry. Values include Yes or No.
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Note: For a terminal-based merchant, the OapfTrxnType should be set to `MarkCapture (9)` or `MarkReturn (10)`. For a host-based merchant, the OapfTrxnType should be set to `Capture (8)` or `Return (5)`.

Void API Output Name-Value Pairs

Output served by the payment system to iPayment returns in the form of HTTP headers and consists of the name-value pairs listed in this table:

Name	Value
OapfStatus	The transaction status. See "OapfStatus" for more information.
OapfTrxnDate	The time stamp for the time when the transaction is processed. This is in YYYYMMDDHHMMSS format.
OapfTrxnType	The transaction type from the payment system. See "Transaction Types and Transaction States" for a list of values.
OapfErrLocation	The error location. See "OapfErrLocation" for more information.
OapfVendErrCode	The payment system error code. See the payment system documentation for more information.
OapfVendErrmsg	The payment system error message. See the payment system documentation for more information.
The following name-value pairs are optional:	
OapfRefcode	The retrieval reference number.
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Void API for Terminal-Based Merchant

For a terminal-based merchant, the Void operation voids the transaction in the local batch. If the Void operation completes successfully, it returns the following parameters:

OapfStatus Set to `0000`.

OapfTrxnType Set to `VoidMarkCapture, 14` or `VoidMarkReturn, 18`

OapfTrxnDate Set to the appropriate transaction date.

If the operation fails, it returns the following parameters:

- OapfStatus
- OapfTrxnType
- OapfTrxnDate
- OapfErrLocation

- OapfVendErrCode
- OapfVendErrmsg

Void API for Host-Based Merchant

For a host-based merchant, the Void operation communicates with the processor to void the specified transaction. If the Void operation completes successfully, it returns the following parameters:

OapfStatus	Set to `0000`.
OapfTrxnType	Set to `VoidCapture, 13` or `VoidReturn, 17`
OapfTrxnDate	Set to the appropriate transaction date.
OapfRefcode	(Optional) Set to the appropriate retrieval reference number.

If the operation fails, it returns:

- OapfStatus
- OapfTrxnType
- OapfTrxnDate
- OapfErrLocation
- OapfVendErrCode
- OapfVendErrMsg

Return/Credit API

The electronic commerce application invokes the Return/Credit API when goods are returned. If the authorization and capture transaction records still exist, the merchant or business will use the existing Order ID to perform a return. If there is no previous authorization or capture records, the merchant or business will create a new Order ID and provide the credit card information.

Return/Credit API Input Name-Value Pairs

To perform the Return/Credit operation, use the name-value pairs listed in this table:

Name	Value
OapfAction	Value = orareturn
OapfOrderId	Order number for the transaction. OapfOrderId can contain only letters, numbers, dashes, underlines, and dots.
OapfPrice	Transaction amount in the format prescribed for the three-letter ISO 4217 currency code.
OapfCurr	ISO 4217 three-letter currency code. For example usd (US Dollar).
OapfPmtInstrID	Identification number (card number). OapfPmtInstrID will be supplied only for credits.
OapfPmtInstrExp	Expiration date for the selected OapfPmtType in the format MM/YY or MM/YYYY. OapfPmtInstrExp will be supplied only for credits.
OapfStoreId	Merchant or business identification. The maximum length is 26 characters.
The following name-value pairs are optional:	
OapfRetry	Specifies if this operation is a retry. Values include Yes or No. If this flag is incorrectly turned on for a stand-alone retry (i.e., one which includes payment instrument information) the servlet should attempt this transaction a second time as a non-retry transaction.
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_ LANG environment variable.

Return/Credit API Output Name-Value Pairs

Output served by the payment system to iPayment returns in the form of HTTP headers and consists of the name-value pairs listed in this table:

Name	Value
OapfStatus	The transaction status. See "OapfStatus" for more information.
OapfTrxnType	The transaction type from the payment system. See "Transaction Types and Transaction States" for a list of values.
OapfTrxnDate	The time stamp of when the transaction is processed. This is in YYYYMMDDHHMMSS format.
OapfPmtInstrType	The payment instrument type such as Visa or MasterCard
OapfErrLocation	The error location. See "OapfErrLocation" for more information.
OapfVendErrCode	The payment system error code. See the payment system documentation for more information.
OapfVendErrmsg	The payment system error message. See the payment system documentation for more information.
The following name-value pairs are optional:	
OapfRefcode	The retrieval reference number
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Return/Credit API for Terminal-Based Merchant

For a terminal-based merchant, the Return/Credit operation marks the transaction for return in the local batch. If the operation completes successfully, it returns the following parameters:

OapfStatus	Set to `0000`.
OapfTrxnType	Set to `MarkReturn, 10`
OapfTrxnDate	Set to the appropriate transaction date

If the operation fails, it returns the following parameters:

- OapfStatus
- OapfTrxnType
- OapfTrxnDate
- OapfErrLocation
- OapfVendErrCode

- OapfVendErrmsg

Return/Credit API for Host-Based Merchant

For a host-based merchant, the Return/Credit operation communicates with the processor to return/credit the transaction. If the operation completes successfully, it returns the following parameters:

OapfStatus	Set to `0000`.
OapfTrxnType	Set to `Return, 5.`
OapfTrxnDate	Set to the appropriate transaction date.
OapfPmtInstrType	(Optional) Set to the appropriate payment instrument type.
OapfRefcode	(Optional) Set to the appropriate retrieval reference number.

If the operation fails, it returns the following parameters:

- OapfStatus
- OapfTrxnType
- OapfTrxnDate
- OapfErrLocation
- OapfVendErrCode
- OapfVendErrmsg

Close Batch API

The merchant or business uses the Close Batch API to close a batch of previously performed transactions. The transaction types that can be included in a close batch are:

- Capture transactions
- Return/Credit transactions

Close Batch API Input Name-Value Pairs

To perform this operation you need the parameters (name-value pairs) listed in this table:

Name	Value
OapfAction	Value = oraclosebatch
OapfStoreId	Merchant or business identification. The maximum length is 26 characters.
The following name-value pairs are optional:	
OapfRetry	Specifies if this operation is a retry. Values include Yes or No.
OapfVpsBatchID	The payment system batch identification
OapfNlsLang	Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_ LANG environment variable.

Close Batch API Output Name-Value Pairs

Output served by the payment system to iPayment returns in the form of HTTP headers and consists of the name-value pairs listed in this table:

Name	Value
OapfStatus	The transaction status. See "OapfStatus" for more information.
OapfBatchDate	The date for this batch
OapfCreditAmount	The credit amount. This is the total outflow including return/credit and void.
OapfSalesAmount	The total amount captured
OapfBatchTotal	The total amount in this batch
OapfCurr	ISO 4217 three-letter currency code. For example, usd (US Dollar).

Name	Value
OapfNumTrxns	The number of transactions in this batch
OapfStoreID	Merchant or business identification. The maximum length is 26 characters.
OapfVpsBatchID	The payment system batch identification
OapfGWBatchID	The gateway batch identification
OapfBatchState	State of the batch. For example, sent, queued, accept, etc. See "OapfBatchState" for more information.
OapfErrLocation	The error location. See "OapfErrLocation" for more information.
OapfVendErrCode	The payment system error code. See the payment system documentation for more information.
OapfVendErrmsg	The payment system error message. See the payment system documentation for more information.
OapfNlsLang	(Optional) Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Close Batch API Additional Output

Additional output for the Close Batch API includes the status of individual transactions. This output differs based on transaction type. The Capture and Return/Credit transaction types return the following parameters:

- OapfOrderId-count=<>
- OapfTrxnType-count=<>
- OapfStatus-count=<>
- OapfErrLocation-count=<>
- OapfVendCode-count=<>
- OapfVendErrmsg-count=<>

Note: OapfErrLocation, OapfVendCode, and OapfVendErrmsg are only returned if the OapfStatus field is non-zero. They are returned when there is some failure for the Order ID during batch close.

The OapfNumTrxns field indicates the number of transactions included in the batch. Each output name-value pair should be appended with a counter to indicate to which transaction it belongs. The counter should start from 0. For example, assume there are two transactions in a batch. The output of this batch is:

```
OapfVpsBatchID: 1234
OapfStatus: PMT-0000
OapfBatchDate: 19970918091000
OapfCreditAmount: 10.00
OapfSalesAmount: 20.00
OapfBatchTotal: 10.00
OapfCurr: usd
OapfNumTrxns: 2
OapfStoreID: abcd
OapfGWBatchID: 5678

OapfOrderId-0=1111
OapfTrxnType-0=8
OapfStatus-0=0000

OapfOrderId-1=2222
OapfTrxnType-1=5
OapfStatus-1=0000
```

Note: The OapfTrxnType should be set to `Capture (8)` or `Return (5)`.

Close Batch API for Terminal-Based Merchant

For a terminal-based merchant, this operation attempts to close out an open batch and cause funds to change hands. If the batch closes successfully, batch summary as well as transaction details should be returned. If the close batch fails, the merchant or business, optionally, fixes offending transactions in the batch and retries. For payment systems that implement retry logic, use OapfRetry and OapfVpsBatchID for retry. For payment systems that do not include retry logic, this operation attempts to close out the existing open batch again.

Close Batch API for Host-Based Merchant

For a host-based merchant, if you use the auto close option, this operation returns OapfStatus=0000. If you use the manual close option, the payment system sends the total to the processor. The processor checks against its total and closes the batch. If the batch closes successfully, OapfStatus should be set to 0000 and OapfBatchTotal should be returned. If

batch does not close successfully, error messages are returned in OapfStatus and optionally in OapfErrLocation, OapfVendErrCode, and OapfVendErrmsg.

Query Transaction Status API

The merchant or business uses the Query Transaction Status API to query the status of a transaction. Both the iPayment database and the payment system database maintain a record of completed transactions, and these databases may become out of synch due to a communication link breakdown. Similarly, the electronic commerce application database and the iPayment database may become out of synch due to a similar condition. This API returns all existing records for a particular Order ID on a payment system.

Query Transaction Status API Input Name-Value Pairs

To perform this operation, use the name-value pairs listed in this table:

Name	Value
OapfAction	Value = oraqrytxstatus
OapfOrderId	Order ID to query
OapfStoreId	Merchant or business identification. The maximum length is 26 characters.
OapfNlsLang	(Optional) Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Query Transaction Status API Output Name-Value Pairs

Output from the Query Transaction Status API may consist of multiple records for the same Order ID, depending on the transaction type. OapfNumTrxns provides the number of transactions for this Order ID. The output for various transaction types includes the following parameters:

Auth/AuthCapture:

```
OapfOrderId-count=<>
OapfTrxnType-count=<>
OapfStatus-count=<>
OapfPrice-count=<>
OapfCurr-count=<>
OapfAuthcode-count=<>
OapfRefcode-count=<>
OapfAVScode-count=<>
OapfTrxnDate-count=<>
OapfPmtInstrType-count=<>
```

```
OapfErrLocation-count=<>
OapfVendCode-count=<>
OapfVendErrmsg-count=<>
OapfAcquirer-count=<>
OapfAuxMsg-count=<>
```

Capture:

```
OapfOrderId-count=<>
OapfTrxnType-count=<>
OapfStatus-count=<>
OapfPrice-count=<>
OapfPrice-count=<>
OapfCurr-count=<>
OapfTrxnDate-count=<>
OapfRefcode-count=<>
OapfVpsBatchID-count=<>
OapfErrLocation-count=<>
OapfVendCode-count=<>
OapfVendErrmsg-count=<>
```

Credit/Return:

```
OapfOrderId-count=<>
OapfTrxnType-count=<>
OapfStatus-count=<>
OapfPrice-count=<>
OapfCurr-count=<>
OapfTrxnDate-count=<>
OapfPmtInstrType-count=<>
OapfRefcode-count=<>
OapfVpsBatchID-count=<>
OapfErrLocation-count=<>
OapfVendCode-count=<>
OapfVendErrmsg-count=<>
OapfAuxMsg-count=<> (optional)
```

Void:

```
OapfOrderId-count=<>
OapfTrxnType-count=<>
OapfStatus-count=<>
OapfTrxnDate-count=<>
OapfRefcode-count=<>
OapfErrLocation-count=<>
OapfVendCode-count=<>
```

```
OapfVendErrmsg-count=<>
OapfAuxMsg-count=<>
```

Query Batch Status API

The merchant or business uses the Query Batch Status API to query the status of an existing batch. Terminal-based merchants also use the Query Batch Status API to verify the transactions for submission to batch close by iPayment. The merchant or business can use the output from the Query Batch Status API to cross-check the transaction records in the merchant or business database.

Query Batch Status API Input Name-Value Pairs

To perform the Query Batch Status operation, use the name-value pairs listed in this table:

Name	Value
OapfAction	Value = oraqrybatchstatus
OapfVpsBatchID	The payment system batch identification if querying for an existing batch. If a value is not included, the output is pending batch transactions.
OapfStoreId	Merchant or business identification. The maximum length is 26 characters.
OapfNlsLang	(Optional) Language and character-set information for the electronic commerce application. The format is the same as for the Oracle Server NLS_LANG environment variable.

Query Batch Status API Output Name-Value Pairs

Output from the Query Batch Status API is similar to the output of the Close Batch API when you provide the OapfVpsBatchID. When you do not provide the OapfVpsBatchID, the output is all transactions for the terminal-based merchant for a subsequent batch close. OapfNumTrxns provides the number of transactions for the batch. The output for transaction types includes the following parameters:

Capture, Return, Credit:

```
OapfOrderId-count=<>
OapfTrxnType-count=<>
OapfPrice-count=<>
OapfCurr-count=<>
OapfTrxnDate-count=<>
```

Transaction Status and Messages

This section describes the various transaction status codes and error messages returned by iPayment payment system servlet.

Topics include:

- OapfStatus
- OapfErrLocation
- OapfVendErrCode
- OapfVendErrmsg
- OapfBatchState
- OapfOrderId

OapfStatus

Each transaction (including authorize, capture, return, credit, and void) returns the status in the OapfStatus field. A value of 0000 or 0 indicates a successfully completed transaction. A non-zero value indicates that the transaction failed. OapfErrLocation, OapfVendErrCode, and OapfVendErrmsg provide additional error information.

SSL Payment System Servlet

SSL payment systems must return the values listed in this table to iPayment in the OapfStatus parameter:

Value	Definition
0000	Transaction completed successfully
0001	Communications error: the payment system or the processor is out of reach. You should resubmit the request at a later time.
0002	Duplicate Order ID
0003	Duplicate Batch ID
0004	Mandatory fields are required.
0005	Payment system specific error. Refer to OapfVendErrCode and OapfVendErrmsg for more information.
0006	Batch partially succeeded. Some transactions in the batch failed and some processed correctly.
0007	The batch failed. You should correct the problem and resubmit the batch.
0008	Requested action not supported
0017	Card has insufficient funds
0019	Invalid credit card

OapfErrLocation

The OapfErrLocation parameter contains the values listed in this table:

Value	Definition
0	Transaction completed successfully at all levels
1	Transaction failed at the payment system cartridge code
2	Transaction failed at the payment system engine or the payment system server code
3	Transaction failed at the payment system gateway or equivalent to the interface that communicates with the bank
4	Transaction failed at the acquirer bank gateway or equivalent to the bank interface that communicates with the payment system interface
5	Transaction failed at the payment system
6	Transaction failed at iPayment

OapfVendErrCode

OapfVendErrCode contains the payment system's error code. See the documentation that came with the payment system for more information. This parameter is required only if the transaction failed at the payment system.

OapfVendErrmsg

OapfVendErrmsg contains the payment system's message for the error. See the documentation that came with the payment system for more information. This parameter is required only if the transaction failed at the payment system.

OapfBatchState

The OapfBatchState parameter indicates the state of the batch based on the processor. If the state is set to "sent," the merchant needs to query the batch again to find out if the batch is accepted and also to retrieve transaction details. The OapfBatchState parameter contains the values listed in this table:

Value	Definition
0	Batch accepted
1	Batch sent
2	Batch queued
3	Batch rejected.
4	Batch processed.
5	Batch error
6	Batch not found
7	Batch unknown

Note: The close batch operation returns its status in OapfStatus, and has the following possible values: 0000, 0003, 0006, and 0007. See "OapfStatus" for more information.

OapfOrderId

iPayment uses the Order ID to uniquely identify each transaction. In the Core API, if the merchant tries to authorize a previously authorized transaction, the payment system will not accept the authorization. The payment system returns the status "Duplicate Order ID."

How iPayment Uses OapfNlsLang

If the electronic commerce application does not pass the OapfNlsLang parameter, iPayment passes information from the electronic commerce application to the payment service cartridge without performing any conversion of character sets.

If the commerce application does pass a value for OapfNlsLang to iPayment, iPayment tries to convert parameters based on the value of OapfNlsLang before sending those parameters to the payment system cartridge.

To do so, iPayment first checks its database for the list of preferred and optional languages for that payment system. (The information in the database reflects what the iPayment administrator entered using the iPayment Administration user interface.)

Secondly, iPayment does one of the following, depending on what it finds in the database:

- If the database lists a language that matches the value of OapfNlsLang, iPayment keeps the value of OapfNlsLang and passes it to the payment system cartridge.
- If the database does not list a language matching the value of OapfNlsLang, iPayment uses the language specified as the preferred language for that payment system, thus changing the value of OapfNlsLang before sending it to the payment system cartridge.

Finally, iPayment converts the values of other parameters so that they are sent to the payment system cartridge in the language specified by OapfNlsLang.

Notice that this conversion process works in only one direction: from the electronic commerce application to the payment system cartridge. If the payment system sets OapfNlsLang when it sends the data back, iPayment uses that information only to store the value of OapfVendErrmsg in its database. iPayment does not convert data sent from the payment system cartridge back to the electronic commerce application.

Format of the NLS_LANG Parameter

The value of this parameter follows the same format as Oracle Server's NLS_LANG environment variable:

```
language_territory.charset
```

For example, JAPANESE_JAPAN.JA16EUC is a valid value for OapfNlsLang.

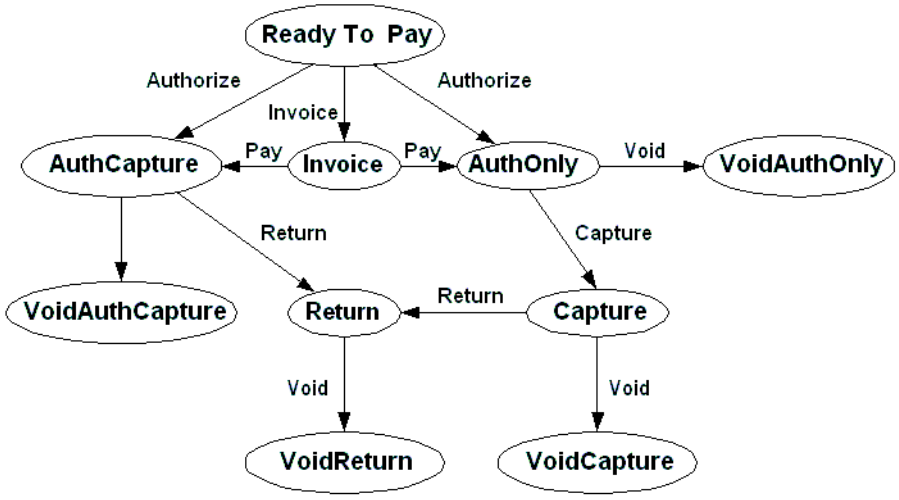
Transaction Types and Transaction States

This section defines the values for OapfTrxnType and includes a discussion of transaction states.

Transaction States

A payment transaction goes through a number of states depending on the operations performed on it. The following illustration depicts the state changes of a transaction in a host based system.

Transaction State Diagram: Host Based System (SSL and SET Systems)

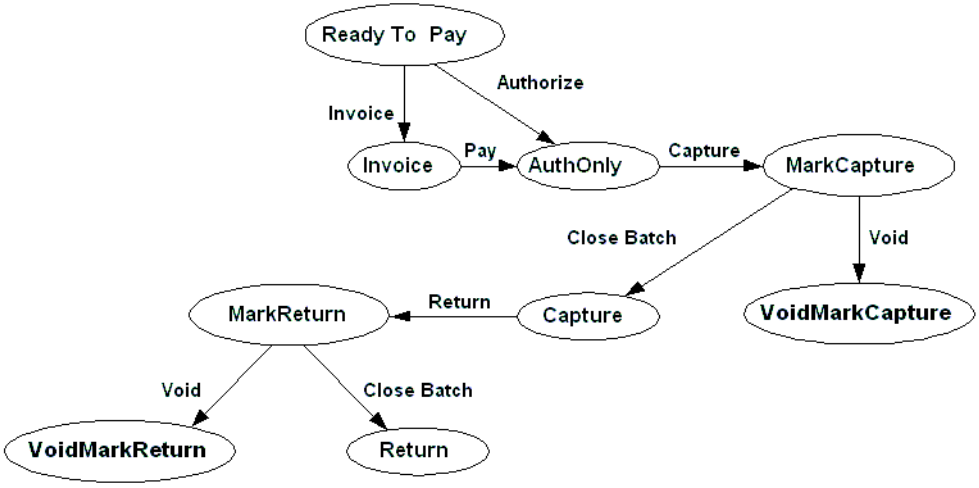


When a customer is ready to pay, the transaction is considered to be in the ready to pay state. If the Authorization API is used, the transaction moves to the authcapture or authonly state after the authorization is complete. If the Invoice and Pay APIs are used, the transaction changes to invoice and moves to authonly or authcapture state. A transaction in the authonly

state needs to be captured for funds to be transferred. All authcapture, capture and return transactions can be voided.

The following graphic illustrates the state changes that a transaction for a terminal based system may undergo. Capture and return operations in terminal based systems only mark the transaction for capture or return in the local batch. After a successful close batch operation the transaction becomes captured or returned.

Transaction State Diagram: Terminal Based System (SSL Systems)



OapfTrxnType: SSL Transactions and Commerce Applications

iPayment returns OapfTrxnType transaction types for the SSL payment system servlet API. This table lists the OapfTrxnType transaction types (SSL).

Value	Type	Definition
2	AuthOnly	An authorization only requested for an order.
3	AuthCapture	An online authorization and capture for an order.
4	VoidAuthOnly	Void of an order that was successfully authorized but not captured. (Electronic Commerce application API only.)
5	Return	Perform a return or credit on an order that was successfully authorized and captured online.
6	ECRefund	Perform a refund on an electronic cash/coin purchase.
7	VoidAuthCapture	Void a previous authorization and capture online.
8	Capture	Capture performed by a host-based or a terminal-based (closed batch) processor system.
9	MarkCapture	Transaction that was marked for capture by a terminal-based processor system.
10	MarkReturn	Transaction that was marked for return by a terminal-based processor system.
13	VoidCapture	Void a transaction captured by a host-based or terminal-based (close batch) processor system.
14	VoidMarkCapture	Void a transaction marked for capture by a terminal-based processor system.
17	VoidReturn	Void a transaction that was returned by a host-based or terminal-based (close batch) processor system.
18	VoidMarkReturn	Void a transaction that was marked for return by a terminal-based system.
101	SplitAuth	A subsequent authorization (Electronic Commerce application API only.)

E

Extensibility

This appendix explains extensibility and how to implement it.

Overview

Extensibility allows interaction between iPayment and a back-end payment system to be customized. Note that extensibility only exists for Gateway-model payment systems. This can be achieved by implementing the following interface:

`ibyextend.TxnCustomizer_<BEP SUFFIX>`

where <BEP SUFFI X> indicates the 3-letter suffix of the back-end payment system.

Custom parameters may be added to those sent by iPayment before the back end payment system servlet is contacted. After the back end payment system servlet responds, the extensibility implementation may take custom parameters that are returned in the response and store them in the database.

Implementation

The Extensibility Interface

To implement extensibility, the Java interface `oracle.apps.iby.extend.TxnCustomizer` must be implemented as class

`ibyextend.TxnCustomizer_<BEP SUFFIX>.`

<**BEP SUFFIX>** is the three letter suffix of the back end payment system.

The `oracle.apps.iby.extend.TxnCustomizer` interface has the following methods:

- **public void preTxn** (String bep, Connection dbconn, AddOnlyHashtable txn_req) throws PSException;
- **public void postTxn** (String bep, Connection dbconn, ReadOnlyHashtable txn_resp) throws PSException;

The parameter `bep` is the three letter suffix, which is specified during registration in the user interface, of the back end payment system that the request goes to, `dbconn` is a connection open to the APPS schema, and `txn_req/txn_resp` are collections of name-value pairs which represent, respectively, the back end payment system request/response.

Note: Both methods can throw a `PSException`. This allows a transaction to be aborted if a critical error, for example, SQLException, occurs in the extensibility implementation class. Releasing the database connection passed to both methods is the responsibility of iPayment and should not be done by the extensibility class.

ReadOnlyHashtable, AddOnlyHashtable Classes

The classes `oracle.apps.iby.util.AddOnlyHashtable` and `oracle.apps.iby.util.ReadOnlyHashtable` are passed as parameters to the `preTxn`, `postTxn` methods respectively. `ReadOnlyHashtable` has the following methods, which are the same in signature and behavior as the corresponding methods of the Java Hashtable class:

`keys, containsKey, isEmpty, size, get`

`AddOnlyHashtable`, which is a subclass of **`ReadOnlyHashtable`**, has the additional method `put`. It differs from the corresponding method in the Java Hashtable class in the way that only keys not already present in the hashtable can be successfully used for

insertions. The **AddOnlyHashtable** version of put returns a boolean value which is true only if the insertion succeeds.

Both types of hashtables are populated with String name-value pairs from one of the back end payment system integration model. In the case of preTxn, these are input name-value pairs. In the case of postTxn, these are output name-value pairs. Below is a piece of sample code illustrating how a value is retrieved:

```
String orderId = (String)txn_resp.get("OapfOrderId");
```

See the Back-End Processing APIs section for a complete listing of all names.

Custom Fields

Custom fields should be prefixed by OapfExtend, which is defined as the constant CUSTOMFIELD_PREFIX in the oracle.apps.iby.extend.TxnCustomizer class. This applies to both fields inserted in the back end payment system request during the call to preTxn, and the custom fields returned by the back end payment system servlet and processed in postTxn. If custom fields do not follow this convention, there is no guarantee that custom fields will be successfully passed through.

Development, Deployment

To develop extensibility classes, include the location of the The Oracle Applications Java class library file containing all of iPayment's classes in the CLASSPATH passed to the compiler.

An extensibility class is deployed by placing it in iPayment's CLASSPATH. Please refer to the local JServ configuration to determine this value.

Note: Since extensibility classes are part of the ibyextend package, the class must be located under a directory called ibyextend.

Exceptions

An exception may be thrown by either the preTxn or postTxn method in the TxnCustomizer class. This exception is the class oracle.apps.iby.exception.PSException

It should be thrown whenever a critical error is encountered in the customizer and the transaction needs to be aborted.

iPayment will take the exception thrown by an extensibility implementation and throw a new PSException based on it with the following error code:

```
IBY_0005
```

The message in the new PSException will have a prefix appended to it, indicating that the error occurred within the extensibility class.

Sample Implementation

```
package ibyextend;

import java.sql.*;
import java.util.Hashtable;
import java.util.Enumeration;

import oracle.apps.iby.extend.TxnCustomizer;
import oracle.apps.iby.util.AddOnlyHashtable;
import oracle.apps.iby.util.ReadOnlyHashtable;
import oracle.apps.iby.exception.PSException;

public class TxnCustomizer_pay implements TxnCustomizer
{

    static final String EXTEND_QUERY="select a, b from
iby.iby_extend_pre where order_id = ?";

    static final String EXTEND_INSERT="insert into iby.iby_extend_post
values (?,?,?)";

   public void preTxn(String bep, Connection dbconn, AddOnlyHashtable
                      inputs) throws PSException
    { String orderId=(String)inputs.get("OapfOrderId");

      try
          { PreparedStatement
                stmnt=dbconn.prepareStatement(EXTEND_TESTQUERY);
           stmnt.setString(1,orderId);
           ResultSet rset=stmnt.executeQuery();

           for (int count=1; rset.next(); count++)
               {
                 String cust1=rset.getString(1),
                     cust2=rset.getString(2);
                inputs.put( TxnCustomizer.CUSTOMFIELD_PREFIX
+
"ReqA-"+count,cust1);
                inputs.put( TxnCustomizer.CUSTOMFIELD_PREFIX
+
"ReqB-"+count,cust2);
               }
           rset.close();
```

```
            stmnt.close();
            // !! do not close the database connection !!
           }
      catch (SQLException sqle)
           { throw new PSException("IBY_0005",sqle.getMessage(),false); }
    }

    public void postTxn(String bep, Connection dbconn,
                        ReadOnlyHashtable outputs) throws PSException
    { String f1=(String)outputs.get("OapfStatus"),

f2=(String)outputs.get(TxnCustomizer.CUSTOMFIELD_PREFIX+"Resp"),
           f3=(String)outputs.get("OapfTrxnDate");
      try
           { PreparedStatement

stmnt=dbconn.prepareStatement(EXTEND_TESTINSERT);
             stmnt.setString(1,f1);
             stmnt.setString(2,f2);
             stmnt.setString(3,f3);
             stmnt.executeUpdate();
             dbconn.commit();
             stmnt.close();
             // !! do not close the database connection !!
           }
      catch (SQLException sqle)
           { throw new PSException("IBY_0005",sqle.getMessage(),false); }
    }
}
```

F

Configuring CyberCash Servlet

This appendix explains how to configure the CyberCash servlet.

Configuring CyberCash Servlet

CyberCash is a Secure Socket Layer (SSL) payment system supporting credit card transactions using Merchant Connection Kit (MCK) and bank account transfers using CyberCash's PayNow services. It supports all Oracle iPayment core operations.

CyberCash Payment System Servlet is only needed if you are planning to process the credit card and Bank Transfer payments through the CyberCash Service. For more information see 'Payment Systems ' in the latest *Oracle iPayment Concepts and Procedures Guide.*

> **Note:** CyberCash is no longer accepting new customers. If you are not an existing CyberCash customer, consider using one of the other out-of-box integrations or contact Verisign, which has written its own iPayment integration servlet

Oracle iPayment integrates with MCK version 3 which connects to CyberCash. Use the parameters in the Oracle iPayment administration user interface while setting up CyberCash as the payment system.

This table lists the parameters for setting up CyberCash as the payment system.

Property	Value
Name	CyberCash
Suffix	cyb (do not use CYB or Cyb)
Base URL	http://<machine_name>.com:<port>/servlet *The machine where CyberCash servlet is to be installed, and any active port, for example:* http://www.merchant.com:9997/servlet
Admin URL	http://amps.CyberCash.com

Installing the CyberCash Servlet

Use the following procedure to configure CyberCash Merchant Connection Kit, also known as MCK to work with Oracle iPayment:

1. Download CyberCash's Merchant Connection Kit (MCK) from http://www.CyberCash.com. Follow CyberCash's instructions to install the MCK.

Note: If your MCK is located inside the firewall and your firewall requires a proxy for outbound communication, then add the following parameters to the MCK merchant_conf file. The merchant_conf file is located in the:

<MCK_HOME>/<merchant-name>/mck-cgi/conf directory:

HTTP_PROXY_HOST=<hostname>

HTTP_PROXY_PORT=<port>

2. Go to the directory where the MCK C libraries are located. The installation directory should be named mck-<version>-<operating system>. For example, if you installed MCK version 3.2.0.6 on Solaris under the /usr/oracle directory, you should do the following:

   % cd /usr/oracle/mck-3.2.0.6-sparc-sun-solaris2.6/c-api/lib

   On Windows NT, the command could be:

   D:\>cd \mck-3.2.0.6-nt\c-api\lib

3. Copy the three MCK libraries mentioned below into the $IBY_TOP/lib (or %IBY_TOP%\lib on Windows NT) directory:

   % cp libCCMck.a $IBY_TOP/lib

   % cp libmckcrypto.a $IBY_TOP/lib

   % cp libmd5hash.a $IBY_TOP/lib

   On Windows NT, the commands will be:

   D:\> copy CCMck.lib %APPL_TOP%\iby\11.5.0\lib

   D:\> copy mckcrypto.lib %APPL_TOP%\iby\11.5.0\lib

   D:\> copy md5hash.lib %APPL_TOP%\iby\11.5.0\lib

   Note: The version number 11.5.0 may differ if you have a different version. Replace 11.5.0 with your specific version number.

4. Go to the $IBY_TOP/admin/driver directory:
   % cd $IBY_TOP/admin/driver

or
cd %APPL_TOP%\iby\11.5.0\admin\driver (Windows NT/2000)

Note: Edit file ibysub01.drv. Make two lines starting with the comment character active by removing the comment character.

5. Go to the $IBY TOP/lib directory:
% cd $IBY_TOP/lib.
or
cd %APPL_TOP%\iby\11.5.0\lib (on Windows NT/2000).

6. Start AD Administration with its command name.
For UNIX users: $ adadmin
For NT users: C:\>adadmin.

After you answer the AD administration questions, the utility takes you to the main menu. Select “Relink Applications programs.”
Log File: the default AD administration log file name is adadmin.log. It is located in $APPL_TOP/admin/<db_name> is the value of your ORACLE_SID or TWO_TASK variable. NT users will find the log file in %APPL_TOP%\admin\<db_name>\log.

7. If JServ is set up for automatic startup, set up the wrapper.env variable in the file jserv.properties as indicated in the following discussion.

.properties file are generally located in the etc directory of your top Jserv engine directory (for example, /d1/testcomn/util/apache/1.3.9/Apache/Jserv/etc).

wrapper.env=LD_LIBRARY_PATH=$IBY_TOP/bin

In Windows NT and Windows 2000, set:
wrapper.env=PATH=%APPL_TOP%\iby\11.5.0\bin

If the file already contains a line for wrapper.env (wrapper.env=LD_LIBRARY_PATH=...), append the location indicated in the preceding instructions as you would append the LD_LIBRARY_PATH environment variable. For example, assume that you have the following line already in the .properties file, line
wrapper.env=LD_LIBRARY_PATH=$ABC/lib

In this case, you should add :$IBY_TOP/bin to the end of the line as shown below:

wrapper.env=LD_LIBRARY_PATH=$ABC/lib:$IBY_TOP/bin

For Windows NT and Windows 2000, wrapper.env should be set as:

wrapper.env=PATH=%ABC%\lib;%APPL_TOP%\iby\11.5.0\bin

If JServ is set up for manual startup, set the appropriate environment variable in your environment shell. This can be done in the jservctl file, or in any other script used to start JServ. The jservctl file is generally located in the bin directory of your top Jserv engine directory (for example, /d1/testcomn/util/apache/1.3.9/Apache/Jserv/bin):

export LD_LIBRARY_PATH=$IBY_TOP/bin

In some shells, you will need to set LD_LIBRARY_PATH as follows:

LD_LIBRARY_PATH=$IBY_TOP/bin

In Windows NT and Windows 2000, set it as follows:

PATH=%APPL_TOP%\iby\11.5.0\bin

If there is already a line setting the LD_LIBRARY_PATH (or PATH in Windows) then append the above location as you would append the LD_LIBRARY_PATH environment variable, using a colon (:) or, in Windows, a semicolon (;).

8. Set up a virtual path mapping for the CyberCash servlet.

Insert the following line in the zone.properties file, in the Servlet Aliases section.

servlet.oramipp_cyb.code=oracle.apps.iby.bep.cybercash.CybServlet.

This allows the servlet to be invoked as: http://<hostname>:<port>/servlet/oramipp_cyb.

9. Set the servlet init parameters. There are several initialization parameters that are recognized by the Oracle iPayment CyberCash Servlet. Set these initialization parameters by inserting the following line in the zone property file <SERVLET_ZONE>.properties file in the Aliased Servlet parameters section.

Note: Replace $MCK_HOME with the absolute path of the MCK installation and replace $IBY_TOP with the absolute path of the Oracle iPayment installation.

```
servlet.oramipp_cyb.initArgs=mckhome=$MCK_HOME,debug=false,logfile=$IBY_
TOP/log/ibycybserv.log
```

In Windows NT, set it to:

```
servlet.oramipp_cyb.initArgs=mckhome=%MCK_HOME%,debug=false,logfile=%APPL_
TOP%\iby\log\ibycybserv.log
```

The following initialization parameters are recognized by the CyberCash Servlet:

- **Mckhome**: This parameter is mandatory. It's the directory path that points to the location where the CyberCash Merchant Connection Kit is installed. For example, if a merchant named, test-mck has been installed in such a way that its associated files can be found under the directory /usr/oracle/mck/test-mck, then mckhome should be set to /usr/oracle/mck. Transaction requests to Oracle iPayment will fail if mckhome is not set correctly.
- **debug**: This parameter is optional. If set to true, then the servlet will print debugging information to the body of its responses in plain text. This information includes the inputs sent to the servlet during the request, and the outputs the servlet sends for its response. If an exception is thrown during the processing of the request, then a stack trace is also printed.
- **logfile:** This parameter is optional. It's a string which specifies the fully qualified path name of the log file location. The input and output values of each transaction are written to this file, and a stack trace if an exception is thrown. If this parameter is not set, logging will be turned off.
- **singlemerch**: This parameter is optional, but may only be set up if the servlet always uses the same CyberCash merchant. The singlemerch parameter helps improve the performance of the CyberCash servlet by eliminating some of the overhead work that is done for multiple merchants. Set up the parameter's value to the CyberCash merchant id. For example, if you are only using the merchant test-mck, use the following initialization argument string:

```
servlet.oramipp_cyb.initArgs=mckhome=$MCK_HOME,debug=false,logfile=$IBY_
TOP/log/ibycybserv.log,singlemerch=test-mck
```

Performance Considerations for Oracle iPayment CyberCash Servlet

The CyberCash servlet makes calls via JNI to CyberCash's C-implemented Merchant Connection Kit (MCK). The MCK is not thread-safe when multiple Cybercash merchants are used. The CyberCash servlet must synchronize access to the MCK, in effect serializing concurrent requests so that each one begins only after a previous one finishes. To improve performance in the case of a single merchant, i.e. when the servlet always uses the same CyberCash merchant, it is recommended that you use the singlemerch parameter. To improve the performance in cases of both the single merchant or multiple merchants, it is necessary to take advantage of a new feature in JServ called load balancing. Load balancing allows requests sent to a single servlet zone to be serviced by multiple JServ instances. Since each JServ instance is a separate process, calls to the MCK occur in distinct memory spaces, allowing multiple concurrent requests to the CyberCash servlet to be successfully processed.

Installing a Load Balanced Servlet Zone

To load balance a servlet zone, make the following changes to your jserv.conf file:

1. For each JServ instance you will reference, include a directive of the form:

   ApJServHost <INSTANCE_NAME> <PROTOCOL>://<HOST>:<PORT>

   For example: ApJServHost PC1 ajpv12://localhost:7777

   Note: Only one protocol is allowed within a zone. You should choose the default one, such as ajpv12.

2. Group JServ instances into sets with the following directive:

   ApJServBalance <SET_NAME> <INSTANCE_NAME>

   For example: ApJServBalance set1 PC1

   ApJServBalance set1 SUN1

3. Define the load-balanced servlet zone with the directive:

   ApJServMount <URL> balance://<SET_NAME >/<SERVLET_ZONE_NAME>

   For example: ApJServMount /cybserv balance://set1/cybserv

   Note: Each JServ instance within the set must have a servlet zone of the given name defined. Using the example above, each JServ instance must have a cybserv zone.

4. Define the shared memory file used by Apache HTTP listeners to keep track of the status of JServ instances use the directive:
   ApJServShmFile <MEM_FILE>

   Note: Note that you may wish to over-write the memory file between Apache restarts to flush old status information.

   After jserv.conf is modified to reflect your installation, restart Apache and make sure each JServ instance within the load balanced zone is running.

   To manually start a JServ instance, do the following steps:

a. Make a copy of your jserv.properties file, assumed to be correctly configured for the CyberCash servlet, for each JServ instance you will run in the new zone.

b. For each properties file, set port to a value correct for that instance.

c. Set your shell environment variables CLASSPATH and LD_LIBRARY_PATH to the values the variables have in your jserv.properties file.

d. From the command line run the command:

java -classpath $CLASSPATH org.apache.jserv.JServ <PROPERTY_FILE> <LOG_FILE> 2>&1

The property file is the jserv.properties file you have configured for that particular instance.

Load Balancing Recommendations

The maximum number of concurrent requests that the CyberCash servlet will be able to process without blocking is equal to the number of JServ instances running in its servlet zone. You should have a number of JServ instances running equal to the average number of concurrent requests, if not slightly more since, under load balancing, JServ instances are randomly chosen, making it possible that two concurrent requests could be sent to a JServ instance when an idle one is already available.

Running multiple JServ instances within a zone will not add significantly to your CPU load versus running a single instance. It will, however, add to your memory load as each instance requires its own JVM. On Solaris, each JVM requires over 6MB of main memory though less than 4MB are actually used since JVMs will share common libraries.

G

Configuring Paymentech

This appendix explains how to configure the Paymentech payment system.

Configuring the Paymentech Servlet

Paymentech is a processor-model payment system which offers online authorization and batch-based settlement support. Oracle iPayment supports the Online Processing Technical Specification, Version 7.2, for online transactions and the 120-Byte Technical Specification, Version 2.1.0, for batch file processing.

Paymentech supports these payment instruments and operations for each payment instrument:

- Credit Cards
  - Online Authorization
  - Batch Authorization
  - Batch Authorization and Deposit
  - Batch Deposit
  - Batch Credit
  - Batch Query
- Purchase Cards
  - Online Authorization
  - Batch Authorization
  - Batch Authorization and Deposit
  - Batch Deposit
  - Batch Credit
  - Batch Query
- PINless Debit Cards
  - Online Authorization
  - Batch Deposit
  - Batch Query
- Bank Receipts
  - Online Verification
  - Batch Validate and Deposit

- Batch Credit
- Batch Query

Prerequisites

Using Paymentech as a payment system has these prerequisites:

- You must have a leased-line connection to Paymentech's payment servers.
- You must have one or more valid Paymentech merchant accounts with support for both IP socket-based online authorization and FTP-based batch-mode settlement.

Please contact Paymentech for help meeting these prerequisites.

The Oracle iPayment Paymentech servlet requires no database connectivity and can be installed on a different application server than iPayment.

To install the Oracle iPayment Paymentech servlet on a different application server:

1. Copy directory $APPL_TOP/java and directory $IBY_TOP/xml to the new machine.
2. Add $APPL_TOP/java to the CLASSPATH of the Jserv instance the servlet will run and set the "xmlbase parameter" to the location of the copied $IBY_TOP/xml. For details on setting the "xmlbase" parameter, see Setting iPayment JVM parameters.
3. Follow the configuration steps.

Servlet Configuration

Follow these mandatory configuration steps regardless of whether iPayment and the Oracle iPayment Paymentech servlet are present in the same machine.

To configure the Oracle iPayment Paymentech servlet:

1. Add this alias statement to the configuration file of the servlet zone that you wish the Paymentech servlet to run in:

   ```
   servlet.oramipp_ptk.code=oracle.apps.iby.bep.proc.paymentech.PTServlet
   ```

2. In the same configuration file, provide these servlet parameters:

Table G–1 Zone-wide servlet parameters for All Processor Servlets

Parameter	Example Value	Description
IBY_XML_BASE	/appl_top/iby/11.5.0/xml	The location of the XML files needed by iPayment's XML framework. This location should point to a directory with the exact same contents as $IBY_TOP/xml.

Table G–1 Zone-wide servlet parameters for All Processor Servlets

Parameter	Example Value	Description
IBY_JAVA_ XML_LOG	/tmp/xml.log	Debug file used to write XML documents in.
ARCHIVE	/var/archive	Directory where iPayment response files are written to. If communication between iPayment and the servlet fails in the middle of a transaction and iPayment retries that transaction at a later date, the archive directory lets the servlet know the original results of the transaction and forward those to iPayment instead of re-attempting the request, which avoids double billing or double authorization.
MAX_ ARCHIVE_AGE	10	Maximum age (in days) that a response file is saved in the archive. The Paymentech servlet will remove all responses in the archive older than this age every time it starts.

Configuring Paymentech in the Oracle iPayment Administrative Interface

Payment System

Paymentech is already seeded in iPayment and you do not need to create a new payment system. Log in to the iPayment administrative user interface as the administrative user to review and modify these parameters:

In this parameter...	Enter this...
Name	Paymentech
Suffix	ptk
Payment System Type	Processor
Base URL	example- http://localhost:8080/servlets
Administration URL	http://www.paymentech.net
Supported Payment Instrument	Purchase Card, PINless Debit Card, Credit Card

Note: Do not change the suffix parameter for seeded payment systems.

Payment System Merchant Identifier

After you have created a payee in the iPayment administrative user interface, you must create a payment system merchant identifier to link the payee's account to the payment system, and let you specify the payee's account and connectivity parameters. For each payee who will use Paymentech enter a recognizable name for the Paymentech payment system identifier. If you upgraded iPayment and already have an existing payment system identifier, you should not change the identifier.

Once you have created a Payment System Merchant Identifier, you must enter these payment system account parameters by clicking on the Enter Parameters icon next to the appropriate payment system identifier.

Table G–2 Paymentech account parameters

Parameter	Description
Merchant Name	Assigned by Paymentech. Company name that appears on the Paymentech account holder's statement.
Division Number	Assigned by Paymentech when a valid merchant account is created. It is also known as the Merchant ID.
Presenter’s ID	Assigned by Paymentech.
PID Password	Assigned by Paymentech.
Submitter’s ID	Assigned by Paymentech.
SID Password	Assigned by Paymentech.

On the same page, enter the connectivity information required to communicate with the payment system servers.

Paymentech uses the same connectivity parameters for all payment instruments supported.

Table G–3 Paymentech online authorization connectivity parameters

Parameter	Example Value	Description
Socket IP Address	192.168.0.1	IP address of the Paymentech host used for online authorizations.
Socket Port Number	8000	Port number to use along with the socket IP address.

Table G–4 Paymentech settlement connectivity parameters

Parameter	Example Value	Description
FTP Server IP Address	192.168.0.1	IP address of the Paymentech host used for batch transactions.
FTP Server Port Number	8000	Port number to use along with the FTP server IP address.
FTP Account Username	Test	FTP username to login to the Paymentech batch transaction server.
FTP Account Password	Test	FTP password to login to the Paymentech batch transaction server.
Local File Directory	/tmp/batch	Directory where batch files to Paymentech are temporarily stored.
Remote File Directory	test/12345	Directory on the Paymentech batch transaction server where batch files should be uploaded to.
Sent File Name		Parameter not used by Paymentech.
Active/Passive Mode	Active	For new connections, Paymentech does not allow FTP connection in the passive mode. The Active/Passive Mode parameter should be set to "Active" for all new merchant connections to Paymentech.

Table G–5 Paymentech online authorization connectivity parameters

Parameter	Example Value	Description
FTP Server IP Address	192.168.0.1	IP address of the Paymentech host used for batch transactions.
FTP Server Port Number	8000	Port number to use along with the FTP server IP address.
FTP Account Name	test	FTP username to login to the Paymentech batch transaction server.
FTP Account Password	test	FTP password to login to the Paymentech batch transaction server.
Local File Directory	/tmp/batch	Directory where batch files to Paymentech are temporarily stored.
Remote File Directory	test/data/12345	Directory where batch files to Paymentech are temporarily stored.
batch Name		Parameter used internally only.

Configuring Paymentech Servlet Load Balancing

If you want to load balance the Oracle iPayment Paymentech servlet, you may want each instance of the servlet to have different values for the connectivity parameters based on your business and technical requirements as well as the payment system's connectivity requirements. You can override the iPayment engine's connectivity parameters by specifying the parameters in this XML file for each instance of the servlet:

<xml_base>/data/TransConfig.xml, where xml_base is a system setup parameter.

```
The structure of the XML file is as follows:
<TransmissionOption>
   <Scheme>PTECH_ONLINE_SOCKET_7_2</Scheme>
   <Parameter>
      <Name>SOCKET_IP</Name>
      <Value>10.140.10.150</Value>
   </Parameter>
   <Parameter>
      <Name>SOCKET_PORT</Name>
      <Value>80</Value>
   </Parameter>
</TransmissionOption>
```

The XML file should have one TransmissionOption element for each transmission protocol that you want to set up.

The tables list the connectivity parameters that you can set at the servlet level.

Table G–6 Paymentech servlet connectivity parameters

Parameter	Example Value	Description
Scheme	PTECH_ONLINE_ SOCKET_7_2	The transmission protocol for the payment instrument. Values for Paymentech are: PTECH_ONLINE_ SOCKET_7_2FTP_PUTPTECH_BATCH_2_1_0_ ACK_GET

Table G–7 Paymentech servlet connectivity parameters - parameters for the PTECH_ ONLINE_SOCKET_7_2 scheme

Parameter	Example Value	Description
SOCKET_IP	192.168.0.1	IP address of the Paymentech host used for online authorizations.
SOCKET_ PORT	8000	Port number to be used along with the socket IP address.

Table G–8 Paymentech servlet connectivity parameters - parameters for the FTP_PUT scheme

Parameter	Example Value	Description
HOST_IP	192.168.0.1	IP address of the Paymentech host used for batch transactions.
HOST_PORT	8000	Port number to use along with the host IP address.
USERNAME	test	FTP username to login to the Paymentech batch transaction server.
PASSWORD	test	FTP password to login to the Paymentech batch transaction server.
LOCAL_DIR	/tmp/batch	Directory where batch files to Paymentech are temporarily stored.
REMOTE_ DIR	test/12345	Directory on the Paymentech batch transaction server where batch files should be uploaded to.

Table G–9 Paymentech servlet connectivity parameters - parameters for the PTECH_ BATCH_3_0_ACK_GET scheme

Parameter	Example Value	Description
HOST_IP	192.168.0.1	IP address of the Paymentech host used for batch transactions.
HOST_PORT	8000	Port number to use along with the host IP address.
USERNAME	test	FTP username to login to the Paymentech batch transaction server.
PASSWORD	test	FTP password to login to the Paymentech batch transaction server.
LOCAL_DIR	/tmp/batch	Directory where batch files to Paymentech are temporarily stored.
REMOTE_ DIR	test/data/12345	Directory on the Paymentech batch transaction server where batch response files may be picked up from.

Enabling the Scheduler

Paymentech is a processor-model payment system. All transactions except authorizations are stored in the iPayment schema and sent to Paymentech only during a batch close operation. Unless you want to manually control the batch close process by implementing calls to the iPayment batch close APIs, the iPayment scheduler program must be enabled with support for these tasks:

- BATCHCLOSE
- BATCHQUERY
- BATCHRETRY

H

Configuring FDC North

This appendix explains how to configure the FDC North payment system.

Configuring the FDC North Servlet

FDC North is a processor-model payment system that offers online authorization and batch-based settlement support. Oracle iPayment supports the ISO 8583 Format Authorization Network Processing Specification for Leased Line Merchants for online transactions and the Magnetic Media and Data Communication Process Specifications Version 2003.1 for batch file processing.

FDC North supports these payment instruments and operations for each payment instrument:

- Credit Card
  - Online Authorization
  - Batch Deposit
  - Batch Credit
  - Batch Query
- Purchase Card
  - Online Authorization
  - Batch Deposit
  - Batch Credit
  - Batch Query

Prerequisites

Using FDC North as a payment system has these prerequisites:

- You must have a leased-line connection to FDC North payment servers.
- You must have one or more valid FDC North merchant accounts with support for both IP socket-based online authorization and FTP-based batch-mode settlement.

The Oracle iPayment FDC North servlet requires no database connectivity and can be installed on a different application server than iPayment.

To install the Oracle iPayment FDC North servlet on a different application server:

1. Copy directory $APPL_TOP/java and directory $IBY_TOP/xml to the new machine.
2. Add $APPL_TOP/java to the CLASSPATH of the Jserv instance the servlet will run and set the "xmlbase parameter" to the location of the copied $IBY_TOP/xml. For details on setting the "xmlbase" parameter, see Setting iPayment JVM parameters.

3. Follow the configuration steps.

Servlet Configuration

Follow these mandatory configuration regardless of whether iPayment and the Oracle iPayment FDC North servlet are present in the same machine.

To configure the Oracle iPayment FDC North servlet:

1. Add this alias statement to the configuration file of the servlet zone that you wish the FDC North servlet to run in:

   ```
   servlet.oramipp_fdn.code=oracle.apps.iby.bep.proc.fdcnorth.FDCNorthServlet
   ```

2. In the same configuration file, provide the servlet parameters.

   For setting the zone-wide parameters, see Table G–1.

Configuring FDC North in the Oracle iPayment Administrative Interface

Payment System

FDC North is already seeded in iPayment and you do not need to create a new payment system. Log in to the iPayment administrative user interface as the administrative user to review and modify these parameters:

In this parameter...	Enter this...
Name	FDCNorth
Suffix	fdn
Payment System Type	Processor
Base URL	example- http://localhost:8080/servlets
Administration URL	http://www.fdms.com
Supported Payment Instrument	Purchase Card, Credit Card

Note: Do not change the suffix parameter for seeded payment systems.

Payment System Merchant Identifier

After you have created a payee in the iPayment administrative user interface, you must create a payment system merchant identifier to link the payee's account to the payment

system, and let you specify the payee's account and connectivity parameters. For each payee that will use FDC North enter a recognizable name for the FDC North payment system identifier. If you upgraded iPayment and already have an existing payment system identifier, you should not change the identifier.

Once you have created a Payment System Merchant Identifier, you must enter these payment system account parameters by clicking on the Enter Parameters icon next to the appropriate payment system identifier.

Table H–1 FDC North account parameters

Parameter	Description
Merchant Type	A code to define whether the merchant or supplier is an independent contractor; has been certified as a small and/or disadvantaged business entity. Refer First Data specifications for the different codes to be set for this field. Required for Level 2 (MasterCard only).
Merchant Account	Twelve-digit account number assigned to the merchant outletby FDC North.
Merchant Postal Code	Five or nine-digit merchant US Zip code OR Canadian Postal code in format ANA_NAN (Example A1B 2C3, with a space in the fourth position).
Merchant ID	Four-digit merchant identification code that is assigned to the merchant by FDC North.
Merchant City	City where the merchant outlet is located.
Merchant Country Code	For US merchants, this parameter must contain the existing two-letter state code with a blank in the third position. For Canadian merchants, this parameter must contain the two-letter province code with an asterisk in the third position. For all other foreign merchants, this parameter must contain a three-letter country code.
Merchant DBA Name	Merchant DBA (Doing Business As) name
Merchant Category Code	Four-digit code that identifies the type of business conducted by the merchant. This parameter, which is found on the Enriched Deposit (E) record, must contain the Merchant Category Code (or SIC Code) identified in the Authorization Request Message.
Terminal ID	Four-character code that identifies a particular terminal at a merchant location. This parameter is found on the Enriched Deposit (E) record.
RPS Info	Requested payment service value for the merchant. EC for merchants using E-Commerce; DM for Direct Marketing.
Security Code	Security code assigned by FDC North.

Table H–1 FDC North account parameters

Parameter	Description
Merchant Customer Service Number	Required for EC transactions. This parameter should contain the customer service telephone number in the format 999-999-9999.
Merchant URL	Merchant URL or e-mail address information for EC transactions. First character cannot be a space. Merchant does not have to include "www".
Merchant Tax ID	Federal Tax ID number or Social Security Number for unincorporated business. Required for Level 2 and MasterCard and preferred for Visa.
Charge Description	The Charge Descriptions that are agreed upon by the client and American Express at the time the Electronic Submission Addendum is completed.

On the same page, enter the connectivity information that is required to communicate with the payment system servers.

FDC North uses the same connectivity parameters for all payment instruments supported.

Table H–2 FDC North online authorization connectivity parameters

Parameter	Example Value	Description
Socket IP Address	192.168.0.1	IP address of the FDC North host used for online authorizations.
Socket Port Number	8000	Port number to use along with the socket IP address.

Table H–3 FDC North settlement connectivity parameters

Parameter	Example Value	Description
FTP Server IP Address	192.168.0.1	IP address of the FDC North host used for batch transactions.
FTP Server Port Number	8000	Port number to use along with the FTP server IP address.
FTP Account Username	test	FTP username to login to the FDC North batch transaction server.
FTP Account Password	test	FTP password to login to the FDC North batch transaction server.

Table H–3 FDC North settlement connectivity parameters

Parameter	Example Value	Description
Local File Directory	/tmp/batch	Directory where batch files to FDC North are temporarily stored.
Remote File Directory	test/12345	Directory on the FDC North batch transaction server where batch files should be uploaded to.
Submission File Generation Data Group	KPTA00Q.DB.KPTD9999.OUTPUT	Generation Data Group used for uploading the Submission file to the Mainframe Server. Provided by FDC North.

Table H–4 FDC North status inquiry connectivity parameters

Parameter	Example Value	Description
FTP Server IP Address	192.168.0.1	IP address of the FDC North host used for batch transactions.
FTP Server Port Number	8000	Port number to use along with the FTP server IP address.
FTP Account Username	test	FTP username to login to the FDC North batch transaction server.
FTP Account Password	test	FTP password to login to the FDC North batch transaction server.
Local File Directory	/tmp/batch	Directory where batch files to FDC North are temporarily stored.
Remote File Directory	test/data/12345	Directory on the FDC North batch transaction server where batch response files may be picked up from.
Acknowledgment Generation Data Group	Acknowledgment Generation Data Group	AcknowledgmentGDG Generation Data Group used for retrieving the Acknowledgment file from the Mainframe Server. Provided by FDC North.

Configuring FDC North Servlet Load Balancing

The Oracle iPayment FDC North servlet does not support load balancing.

Enabling the Scheduler

FDC North is a processor-model payment system. All transactions except authorizations are stored in the iPayment schema and sent to FDC North during a batch close operation. Unless

you want to manually control the batch close process manually by implementing calls to the iPayment batch close APIs, the iPayment scheduler program must be enabled with support for these tasks:

- BATCHCLOSE
- BATCHQUERY
- BATCHRETRY

I

Configuring Concord EFSnet

This appendix explains how to configure the Concord EFSnet payment system.

Implementing Concord EFSnet Servlet

Concord EFS is one of the largest electronic payment service providers in the United States, specializing in credit and debit transaction processing. EFSnet is Concord's Internet payment processing platform. Concord EFSnet is a gateway type payment system, which offers online authorization, settlement, refund and query supports. Oracle iPayment supports the EFSnet Web Services format, Version 2.4.

Oracle iPayment's Concord EFSnet integration supports these payment instruments and the online operations for each payment instrument:

- Credit card
  - Authorization (CreditCardAuthorize)
  - Capture (CreditCardSettle)
  - Auth-capture (CreditCardCharge)
  - Voice authorization (CreditCardVoiceAuthorize)
  - Refund (CreditCardRefund)
  - Credit (CreditCardCredit)
  - Void (VoidTransaction)
  - Query (QueryTransactions)
- PINless debit card
  - Auth-capture (DebitCardChargePINless)

Prerequisites

Using Concord EFSnet as a payment system has these prerequisites:

- You must be able to access Concord EFSnet’s payment servers using HTTP Protocol.
- You must have one or more valid Concord EFSnet merchant accounts with support for HTTP based online authorization and settlement.

The Oracle iPayment Concord EFSnet servlet requires no database connectivity and can be installed on a different application server than iPayment.

To install the Oracle iPayment Concord EFSnet servlet on a different application server:

1. Copy directory $APPL_TOP/java and directory $IBY_TOP/xml to the new machine.

2. Add $APPL_TOP/java to the CLASSPATH of the Jserv instance the servlet will run and set the "xmlbase parameter" to the location of the copied $IBY_TOP/xml. For details on setting the "xmlbase" parameter, see Setting iPayment JVM parameters.
3. Follow the configuration steps.

Servlet Configuration

Follow these mandatory configuration steps regardless of whether iPayment and the Oracle iPayment Concord EFSnet servlet are present in the same machine.

To configure the Oracle iPayment Concord EFSnet servlet:

1. Add this alias statement to the configuration file of the servlet zone that you wish the Oracle iPayment Concord EFSnet servlet to run in:

   ```
   servlet.oramipp_efs.code=oracle.apps.iby.bep.concord.ConcordBEPServlet
   ```

2. In the same configuration file, provide these servlet parameters.

For setting the zone-wide parameters, see Table G–1.

Configuring Concord EFSnet in the Oracle iPayment administrative interface

Payment System

Concord EFSnet is already seeded in iPayment and you do not need to create a new payment system. Log in to the iPayment administrative user interface as the administrative user to review and modify these parameters:

In this parameter...	Enter this...
Name	Concord EFSnet
Suffix	efs
Payment System Type	Gateway
Base URL	example- http://localhost:8080/servlets
Administration URL	http://www.concordefsnet.com
Supported Payment Instrument	Credit Card, PINless debit Card, Purchase Card, Electronic Funds Transfer

Note: Do not change the suffix parameter for seeded payment systems.

Payment System Merchant Identifier

After you have created a payee in the iPayment administrative user interface, you must create a payment system merchant identifier to link the payee's account to the payment system, and let you specify the payee's account and connectivity parameters. For each payee that will use Concord EFSnet enter a recognizable name for the Concord EFSnet payment system identifier. If you upgraded iPayment and already have an existing payment system identifier, you should not change the identifier.

Once you have created a Payment System Merchant Identifier, you must enter these payment system account parameters by clicking on the Enter Parameters icon next to the appropriate payment system identifier.

Table I–1 Concord EFSNet account parameters

Parameter	Description
Store ID	EFSnet store name.
Store key	EFSnet store password.
Application ID	Originating application identifier and version number.

On the same page, enter the connectivity information required to communicate with the payment system servers.

Concord EFSnet uses the same connectivity parameters for all payment instrument types.

Table I–2 Concord EFSNet connectivity parameters

Parameter	Example Value	Description
Destination URL	https://testefsnet.concordebiz.com/efsnet.dll	The URL where the transaction request should be posted.
User proxy	http://www-proxy.us.oracle.com	The proxy used, if any, to connect to the above URL.
Wallet Location	/opt/oracle/wallet	Absolute location of the wallet.
Wallet Location	welcome	Password to open the wallet.

Configuring Concord EFSnet Servlet Load Balancing

If you want to load balance the Oracle iPayment Concord servlet, you may want each instance of the servlet to have different values for the connectivity parameters based on your business and technical requirements as well as the payment system's connectivity requirements. You can override the iPayment engine's connectivity parameters by specifying the parameters in this XML file for each instance of the servlet:

<xml_base>/data/TransConfig.xml, where xml_base is a system setup parameter.

The structure of the XML file is as follows:

```
<TransmissionOption>
   <Scheme>HTTP_POST</Scheme>
   <Parameter>
      <Name>HTTP_URL</Name>
      <Value>10.140.10.150</Value>
   </Parameter>
</TransmissionOption>
```

The XML file should have one TransmissionOption element for each transmission protocol that you want to set up.

These tables list the connectivity parameters that you can set at the servlet level.

Table I–3 Concord EFSnet servlet connectivity parameters

Parameter	Example Value	Description
Scheme	HTTP_POST	The transmission protocol for the payment instrument. Values for Concord EFSnet are:HTTP_POST

Table I–4 Concord EFSnet servlet connectivity parameters - parameters for the HTTP_POST scheme

Parameter	Example Value	Description
HTTP_URL	https://testefsnet.concordebiz.com/efsnet.dll	The URL where the transaction request should be posted.
PROXY	http://www-proxy.us.oracle.com	The proxy used, if any, to connect to the above URL.
WALLET_LOC	/opt/oracle/wallet	Absolute location of the wallet.
WALLET_PASSWD	welcome	Password to open the wallet.

Enabling the Scheduler

Concord EFSnet is a gateway-model payment system. Transactions are submitted to the payment system in real time and you do not need to configure the scheduler.

J

Configuring Citibank

This appendix explains how to configure the Citibank card for transaction processing.

Configuring the Citibank Card Servlet

Citibank is a processor-model payment system which supports two types of transmissions: Online real-time transactions and Batch file processing. Online real-time transactions are used for online real-time authorization request. Batch file processing supports batched credit card authorizations and settlement processing. Oracle iPayment supports the ISO 8583 format for online transactions and the EPF#1 specification for batch file processing.

Oracle iPayment's Citibank integration supports these payment instruments and the operations for each payment instrument:

- Credit Card
  - Online Authorization
  - Online Authorization and Deposit
  - Batch Authorization
  - Batch Authorization and Deposit
  - Batch Deposit
  - Batch Credit
  - Batch Query (does not require communication with Citibank, as Citibank automatically uploads acknowledgment responses for batch files)
- Purchase Card

  The support for purchase card is similar to credit cards, without any level II or III information. Citibank treats purchase card transactions similar to credit card transactions.

Prerequisites

Using Citibank as a payment system has these prerequisites:

- Establish a connection to Citibank payment servers.
- Establish one or more valid Citibank merchant IDs with support for both IP socket-based online authorization and FTP-based batch-mode settlement.
- Configure an FTP server in the machine where you want to set up the Oracle iPayment Citibank servlet. You must communicate the IP address of this FTP server along with the user name and password to Citibank. Citibank will upload the acknowledgment files to the specified directory in this FTP server.

Note: Ensure that you have write permissions on the directory where Citibank uploads the files.

The Oracle iPayment Citibank servlet requires no database connectivity and can be installed on a different application server than iPayment.

To install the Oracle iPayment Citibank servlet on a different application server:

1. Copy directory $APPL_TOP/java and directory $IBY_TOP/xml to the new machine.
2. Add $APPL_TOP/java to the CLASSPATH of the Jserv instance the servlet will run and set the "xmlbase parameter" to the location of the copied $IBY_TOP/xml. For details on setting the "xmlbase" parameter, see Setting iPayment JVM parameters.
3. Follow the configuration steps.

Servlet Configuration

Follow these mandatory configuration steps regardless of whether iPayment and the Oracle iPayment Citibank servlet are present in the same machine.

To configure the Oracle iPayment Citibank servlet:

1. Add this alias statement to the configuration file of the servlet zone that you wish the Citibank servlet to run in:

   ```
   servlet.oramipp_cit.code=oracle.apps.iby.bep.proc.citibank.CitiServlet
   ```

2. In the same configuration file, provide the servlet parameters.

   For setting the zone-wide parameters, see Table G–1.

   This table lists parameters particular to the Citibank servlet (set via a statement of the form servlet.oramipp_cit.initArgs=).

Table J–1 Citibank-specific servlet parameters

Parameter	Example Value	Description
FILELESS_FTP_ENABLED	Y/N	If this parameter is set to Y, the servlet creates a batch file in memory only and uses FTP to send the batch file to the payment system. If this parameter is set to N, the servlet first stores the batch file in local batch directory and then sends the file. We recommend that you set this parameter to Y for enhanced security of your payment information.

Configuring Citibank in the Oracle iPayment Administrative Interface

Payment System

Citibank is seeded in iPayment and you need not create a new payment system. Log in to the iPayment administrative user interface as the administrative user to review and modify these parameters:

In this parameter...	Enter this...
Name	Citibank
Suffix	cit
Payment System Type	Processor
Base URL	example- http://hostname:8080/servlets
Administration URL	http://www.citicorp.com
Supported Payment Instrument	Credit Card

Note: Do not change the suffix parameter for seeded payment systems.

Payment System Merchant Identifier

After you have created a payee in the iPayment administrative user interface, you must create a payment system merchant identifier to link the payee's account to the payment system, and let you specify the payee's account and connectivity parameters. For each payee using Citibank, enter a recognizable name for the Citibank payment system identifier. If you upgraded iPayment and already have an existing payment system identifier, you should not change the identifier.

Once you have created a Payment System Merchant Identifier, you must enter these payment system account parameters by clicking on the Enter Parameters icon next to the appropriate payment system identifier.

Table J–2 Citibank account parameters

Parameter	Description
Merchant ID	Assigned by Citibank Merchant Services (Citi MS) to identify each merchant. This parameter provides the correct merchant information for authorization based upon account type.

Table J–2 Citibank account parameters

Parameter	Description
Acquiring ID	The Acquiring Institution ID Code identifies Citi MS to the Interchange.
Presenter ID	Four-letter ID assigned and provided to merchants by Citi MS to identify each presenter that services are provided to.
Merchant Country Code	Two-letter Merchant Country Code as specified in ISO 3166.
Merchant Postal Code	Postal code of the merchant originating the transaction. This code should be either five or nine digits in length.
Merchant DBA Name	DBA (Doing Business As) Information contains the name of the merchant that defines the point of service in both local and interchange environments.
Merchant DBA City	City where the merchant outlet is located. For an EC transaction, this parameter should contain the customer service telephone number in the format 999-999-9999.
Merchant DBA State	For US merchants, this parameter must contain the existing two-letter state code. A blank must be placed in the third position.
Card Acceptor Terminal ID	The terminal ID at the merchant location.
Terminal Time Offset	Terminal time offset in minutes. The first position must be either '+' or '-'. Example: '+000'
Network Destination	Contains four-letter Citibank Merchant Services network destination for the transaction.
XCF Password	Assigned by Citi MS during Merchant setup.
XCF Request Code	Assigned by Citi MS during Merchant setup.

In the same page, enter the appropriate connectivity information to communicate with the payment system servers.

Citibank uses the same connectivity parameters for all supported payment instrument types.

Table J–3 Citibank online connectivity parameters

Parameter	Example Value	Description
Socket IP Address	150.110.233.112	IP address of the Citibank host used for online transactions.
Socket Port Number	4141	Port number used with the socket IP address.

Table J–4 Citibank batch connectivity parameters

Parameter	Example Value	Description
FTP Server IP Address	163.39.230.33	IP address of the Citibank host that is used for batch transactions.
FTP Server Port Number	21	Port number used with the FTP IP address.
FTP Account Username	Oracle1	FTP username to log into Citibank batch transaction server.
FTP Account Password	welcome	FTP password to log into Citibank batch transaction server.
Local File Directory	/tmp/batch	Directory where batch files are temporarily stored in the user's system.
Data Class Size	SMALL	The size of the file transmitted to Citibank's FTP server (for batch).
Citi Receiving Filename	SIAX00Q.GB.SIAX1011.A01ORCL(+1)	The name of the file transmitted to Citibank's FTP server (for batch).

Table J–5 Citibank status inquiry parameters

Parameter	Example Value	Description
Local File Directory	/tmp/query	Directory in the user's system where acknowledgment files are temporarily stored.
Account Merchant Name	Oracle	Merchant name, assigned by Citibank.

Configuring Citibank Servlet Load Balancing

If you want to load balance the Oracle iPayment Citibank servlet, you may want each instance of the servlet to have different values for the connectivity parameters based on your business and technical requirements as well as the payment system's connectivity requirements. You can override the iPayment engine's connectivity parameters by specifying the parameters in this XML file for each instance of the servlet:

```
<xml_base>/data/TransConfig.xml, where xml_base is a system setup parameter.
The structure of the XML file is as follows:
<TransmissionOption>
   <Scheme>CITI_ONLINE_3_0_SOCKET</Scheme>
   <Parameter>
      <Name>SOCKET_IP</Name>
      <Value>10.140.10.150</Value>
```

```
   </Parameter>
   <Parameter>
      <Name>SOCKET_PORT</Name>
      <Value>80</Value>
   </Parameter>
</TransmissionOption>
```

The XML file should have one TransmissionOption element for each transmission protocol that you want to set up.

These tables list the connectivity parameters that you can set at the servlet level.

Table J–6 Citibank servlet connectivity parameters

Parameter	Example Value	Description
Scheme	CITI_ONLINE_ 3_0_SOCKET	The transmission protocol for the payment instrument. Values for Citibank are:CITI_ ONLINE_3_0_SOCKETCITI_BATCH_3_0_ PUTCITI_BATCH_3_0_ACK_GET

Table J–7 Citibank servlet connectivity parameters - parameters for the CITI_ ONLINE_3_0_SOCKET scheme

Parameter	Example Value	Description
SOCKET_IP	150.110.233.112	IP address of the Citibank host used for online transactions.
SOCKET_PORT	4141	Port number to use along with the socket IP address.

Table J–8 Citibank servlet connectivity parameters - parameters for the CITI_BATCH_ 3_0_PUT scheme

Parameter	Example Value	Description
HOST_IP	163.39.230.33	IP address of Citibank host used for batch transactions.
HOST_PORT	21	Port number to use along with the host IP address.
USERNAME	Oracle1	FTP username to login to Citibank batch transaction server.
PASSWORD	welcome	FTP password to login to Citibank batch transaction server.

Table J–8 Citibank servlet connectivity parameters - parameters for the CITI_BATCH_3_0_PUT scheme

Parameter	Example Value	Description
LOCAL_DIR	/tmp/batch	Directory in a user's system where batch files are temporarily stored.
DATA_CLASS_SIZE	SMALL	The size of the file transmitted to Citibank's FTP server (for batch).
FILE_NAME	SIAX00Q.GB.SIAX1011.A01ORCL(+1)	The name of the file transmitted to Citibank's FTP server (for batch).

Table J–9 Citibank servlet connectivity parameters - parameters for the CITI_BATCH_3_0_ACK_GET scheme

Parameter	Example Value	Description
LOCAL_DIR	/tmp/query	Directory in a user's system where acknowledgment files are temporarily stored.
MERCHANT_NAME	Oracle	Merchant name, assigned by Citibank.

Enabling the Scheduler

Because Citibank is a processor-model payment system, all transactions except authorizations are stored in the iPayment schema and sent to Citibank only during a batch close operation. Unless you want to manually control the batch close process by implementing calls to the iPayment batch close APIs, the iPayment scheduler program must be enabled with support for these tasks:

- BATCHCLOSE
- BATCHQUERY
- BATCHRETRY

K

Profile Options

This appendix lists the profile options that affect the operation of iPayment. This appendix includes a brief description of each profile option that you or your system administrator can set at the site, application, responsibility, or user levels.

Profile Options

During implementation, your system administrator sets a value for each user profile option to specify how Oracle Applications controls access to and processes data.

> **See also:** Overview of Setting User Profiles, *Oracle Applications System Administrator's Guide*

Profile Options Summary

This table indicates whether you can view or update profile options and at which System Administrator levels the profile options can be updated: at the user, responsibility, application, or site levels.

A *Required* profile option requires you to provide a value. An *Optional* profile option already provides a default value which you can change.

The key for this table is:

- **Update** - You can update the profile option
- **View Only** - You can view the profile option but cannot change it
- **No Access** - You cannot view or change the profile option value

Profile Option	Value	Default	User Access	System Admin Access: User	System Admin Access: Responsibility	System Admin Access: Application	System Admin Access: Site
IBY: ECAPP URL	Required	No Default	No Access	No Access	No Access	No Access	Update
IBY: HTTP Proxy	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: No Proxy Domain	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: XML Base	Required	No Default	No Access	No Access	No Access	No Access	Update
IBY: JAVA XML Log File	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: XML Temp Directory	Optional	No Default	No Access	No Access	No Access	No Access	Update

Profile Option	Value	Default	User Access	System Admin Access: User	System Admin Access: Responsibility	System Admin Access: Application	System Admin Access: Site
IBY: Outbound Payment Payer ID	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: Outbound Payment System Suffix	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: Default Payee for BR Remittance	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: UI Visibility Class	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: Wallet Location	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: Wallet Password	Optional	No Default	No Access	No Access	No Access	No Access	Update
IBY: Registered Instrument Encryption	Optional	No	No Access	No Access	No Access	No Access	Update
IBY: Daily Business Close Rporting Currency	Required	USD	Update	Update	Updated	Update	Update

iPayment Profile Options

You can use the System Administrator responsibility to set the iPayment profile options.

IBY: ECAPP URL

This property contains the following URL:

http://machine:port/<jsp>/ecapp?

Replace the machine and port with the names of the actual machine and the actual port where the iPayment ECServlet is installed. Also, make sure that "?" is present at the end of the URL or append "?" at the end.

This information is mandatory if your EC applications use iPayment PL/SQL APIs or if your application is an Oracle 3i client.

IBY: HTTP Proxy

This property specifies the proxy-URL. For example, http://www-proxy.us.oracle.com.

To set up this property with an empty value, insert a string starting with <. For example, <none>.

IBY: No Proxy Domain

This property specifies the domain name for which no proxy is needed. For example, us.oracle.com.

To set up this property with an empty value, insert a string starting with <. For example, <none>.

IBY: XML Base

This property specifies the location of files required by iPayment's XML framework, such as iPayment DTD files. This property should give the location of the $IBY_TOP/xml directory, where $IBY_TOP is expanded to its fully qualified path name. For example, /usr/appl_ top/iby/11.5.0/xml

IBY: JAVA XML Log File

This optional property gives the full-qualified pathname of the debug file where XML messages should be written. This file is similar in purpose to the iPayment debug file, but has been separated from it since XML messages are much larger than single debug statements. If no value is specified for this property, then XML logging is disabled.

IBY: XML Temp Directory

Temporary XML work directory, which must be writable by iPayment's application server. This parameter is optional, but will reduce iPayment's memory usage if provided.

IBY: Outbound Payment Payer ID

Select from the list of values displayed, the payee in iPayment issuing the payment order to the bank. You can set this only at the site level. You need to define this to send transactions from Oracle Payables to iPayment.

IBY: Outbound Payment System Suffix

Enter the three-letter suffix of the payment system that will handle your outbound payment instructions.

IBY: Default Payee for BR Remittance

Select from the list of values displayed, the payee in iPayment remitting the Bills Receivable. You can set this only at the site level. You need to define this to send BR remittance batch from Oracle Receivables to iPayment.

IBY: UI Visibility Class

You can define the visibility class profile option at different levels. This value will determine what data a user can see in the iPayment Operation UI and what mask is applied to the data before displaying it.

IBY: Wallet Location

Location of the Oracle Wallet.

IBY: Wallet Password

Password to open the Oracle Wallet.

IBY: Registered Instrument Encryption

Determines whether registered payment instruments must be stored in encrypted format; if set to 'Yes', the system security key must have been provided to the iPayment engine in order to register/modify payment instruments; use encrypted registered payment instruments as part of a transaction. The default value is ’No’.

Index