Oracle® Retail EFTLink

Security Guide

Release 21.0
F61681-02
August 2022

Oracle Retail EFTLink Security Guide, Release 21.0

F61681-02

Primary Author: Tracy Gunston

Contributors: Matthew Preston, Ian Williams

Contents

Send Us Your Comments

Oracle® Retail EFTLink Security Guide Release 21.0

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note:

Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Help Center (OHC) website (`docs.oracle.com`). It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at `http://www.oracle.com`.

Preface

This guide serves as a best practice guide for ensuring secure operation of Oracle Retail EFTLink. Installation and configuration are covered in more detail in separate guides as listed in the Related Documents section below.

Audience

This document is intended for administrators and engineers who are responsible for secure deployment of EFTLink.

Related Documents

For more information, see the following documents in the Oracle Retail EFTLink Release 21.0 documentation set:

- *Oracle Retail EFTLink Release Notes*
- *Oracle Retail EFTLink Core Configuration Guide*
- *Oracle Retail EFTLink Framework Advanced Features Guide*
- *Oracle Retail EFTLink Framework Installation and Configuration Guide*
- *Oracle Retail EFTLink Xstore Compatibility Guide*
- *Oracle Retail EFTLink Rest API Guide*
- *Oracle Retail EFTLink Validated Partner Cores Guide*
- *Oracle Retail EFTLink Validated OPI Partners Guide*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

`https://support.oracle.com`

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 21.0) or a later patch release (for example, 21.0.1). If you are installing the base release and additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times not be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced at the Oracle Help Center (OHC) website (`docs.oracle.com`), or, in the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

This process will prevent delays in making critical corrections available to customers. For the customer, it means that before you begin installation, you must verify that you have the most recent version of the Oracle Retail documentation set. Oracle Retail documentation is available at the Oracle Help Center at the following URL:

`https://docs.oracle.com/en/industries/retail/index.html`

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number F123456-02 is an updated version of a document with part number F123456-01.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation at the Oracle Help Center

Oracle Retail product documentation is available on the following website:

`https://docs.oracle.com/en/industries/retail/index.html`

(Data Model documents are not available through Oracle Help Center. You can obtain them through My Oracle Support.)

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Security Guidelines

This chapter describes retailer and solution specific responsibilities for ensuring EFTLink is securely implemented and configured.

Oracle Support

It is a best practice to have all Oracle Retail EFTLink support requests submitted through a single point of contact for that customer environment; the client designated administrator is usually designated to perform this role.

The link to use when submitting Service Requests (SR) is:

`https://support.oracle.com`

General Principles

This section describes general principles to be observed.

Securing Sensitive Data

The protection of sensitive data during transit and processing is paramount. Sensitive data includes personally identifiable information such as PAN Numbers and track 2 data. Ensure that if configurable, the EPS/payment terminal is set for PCI compliant masking of card PAN and track 2 data.

Retailer Responsibilities

An instance of EFTLink and any third-party EFT software (dependent on solution) will typically run on the POS hardware and communicate with each other to process EFT transactions when requested by the POS software.

The POS Terminals are in the customer facing areas of the store in proximity to both customers and employees. Physical security of the hardware is the responsibility of the retailer in addition to operational practices like provisioning employees to appropriate application roles and shutting registers down when not in use.

Securing the in-store network is a responsibility of the retailer and is assumed to be compliant with PCI-DSS requirements for topology, wireless access, and wan connections. The connection to the corporate data centers and the external credit authorizers also are assumed to follow PCI-DSS requirements for secured connections.

The PCI-DSS standards are available at:

`https://www.pcisecuritystandards.org/pci_security/`

It is recommended that all machines on the store network be kept up to date with vendor supplied patches, especially security patches. The operating systems on POS Terminals should be locked down by removing or disabling unneeded functionality, such as ensure that the system cannot be used for browsing the internet.

POS Security Considerations

POS security recommendations will vary according to the POS software being used. Please refer to the appropriate POS Security Guide or the POS Implementation Guide for your product.

TLS Encryption

EFTLink V15.0.1 onwards, secures the connection between the POS and the framework using TLS encryption. This is enabled by default.

TLS encryption between the POS and the EFTLink framework can be disabled by setting the Framework configuration file EftlinkConfig.properties `TLSEnabled=false`, but this should only be done after consultation with Oracle, as it reduces the security of the solution, and needs to be disabled at the POS side of the process as well.

Solution Specific Responsibilities

This section gives core specific security guidance.

Adyen

A password for Adyen is required to be encrypted in the `adyen.properties` file.

For the password to be entered, a keystore can be generated via a batch file which will be held in the data directory.

Subsequently the password may also be entered into the system for storage using the batch file, the encrypted password output and this can then be placed into the `adyen.properties` file.

AJB FiPay

AJBFiPay has an `enable.signature.logging` option. Enabling this property (setting to `True`) results in exposing the PII (customer signature). The `enable.signature.logging` option should therefore only be enabled when requested by Oracle Support for debugging purposes and should be turned off **immediately** debugging has been complete.

EFTLink V15.01 includes a modification to allow the core to be used for reading POS handled Cards (Gift Cards, Employee Cards and so on) removing the need for a card swipe attached to the POS itself.

To be able to use this feature, Oracle needs to be consulted, as special configuration needs to be applied to transfer the full card details of the required cards to the POS.

Cayan Core

There are specific security implementation considerations for the Cayan Core. The Cayan Core will be shipped with a public root certificate. When the Cayan Core is initialized a Java Key Store (JKS) is created and secured. If there is no public certificate stored in the JKS then the root certificate file is located and converted to an encrypted certificate file and stored securely in the JKS. The public root certificate file is then deleted from the installation folder.

The certificate is required in order to create the secure socket required for the https session with the authorization host.

Once the encryption key has been created and stored, and the certificate also secured, the install procedure requires the entry of several details such as merchantid to be entered via the POS. As these are entered, they are stored and encrypted as required, and connection to the pin terminal is then possible. Full details of the procedure are available in the installation guide for Cayan.

PayPal

The PayPal implementation requires that a keystore containing a self-signed certificate is generated and stored/trusted on the client. A password is also obfuscated and stored on the client/server.

PointUS

The PointUs device is paired with EFTLink via a registration process, involving a four-digit pin.

No secure connection information is held within EFTLink.

Verifone Ocius Sentinel

Verifone Ocius Sentinel requires a user login ID and PIN to be stored on the POS system.

These are transmitted by EFTLink to the Ocius Sentinel application as part of a login process which is required before Ocius Sentinel can accept EFT requests.

When it is running, the Ocius Sentinel application also has a GUI (GraphicalUserInterface) which can be accessed by an operator from the Windows System Tray. This GUI has a login screen. The login screen accepts the same ID and PIN as stored in the EFTLink core configuration file. Having manually logged into Ocius Sentinel several functions are available to the user, including processing payments and refunds which bypasses the POS software.

In order to prevent unauthorized use of the Ocius Sentinel application the user login ID and PIN should be stored encrypted in the EFTLink core configuration file. An encryption tool is provided to implementers for this purpose and details on its use can be found in the *Oracle Retail EFTLink Core Configuration Guide*. It is recommended that batch encryption of user login ID and PIN data be carried out at a central location and the encrypted data then be distributed to stores as required. Once encryption has taken place the clear text copy of the data can be deleted.

Note:

EFTLink is configured to expect encrypted ID and PIN data by default.

Payment System Comms/Security

The following table shows the comms and security for each payment system:

Table 1-1 Payment System Comms/Security

Payment System	Driver/ Server Application	Comms	Security	Notes
EFTLink Framework	-	Socket XML	TLS	Self-certified certificate generated as part of build process. Certificate stored in a Java Keystore and included in release
Adyen	POS_JNI	Not detailed	None	Uses provided POS_JNI
Cayan	-	WebService SOAP Socket	TLS	Encryption key stored in java keystore and included in release fileset.
Verifone Point US	-	Verifone Point US	Data includes encrypted security field	Encryption key established by initial pairing using RSA exchange. Key stored in a Java Keystore.
AJB FiPay	FiPay	Socket CSV Text	None	
TLG SolveConnect	SolveConnect	Socket XML	None	
Verifone Ocius Sentinel	Ocius Sentinel	Socket XML	None	
Six Payment Services	MPD (OPI mode)	Socket XML	None	
Worldpay (YesPay)	Worldpay	Socket CSV Text	None	

2

Secure Configuration

This chapter describes retailer and solution specific responsibilities for ensuring EFTLink is securely implemented and configured.

EFTLinkConfig.properties

EFTLink installation defaults to secure values, however these may be overridden if desired (for example changes to cryptoagility).

The following is a list of configuration options which are secure by default but may be incorrectly configured to insecure settings.

TLS

Use of TLS over SSL may be specified using the following setting.

```
TLSEnabled = true
```

This specifies that a TLS connection is to be used. If this option is not enabled, then communications between the POS and EFTLink will not be secure and data over this connection can be viewed.

```
Default setting: true
```

Crypto-Agility for Communications

On the connection between POS and EFTLink the protocol and ciphers used to secure the data can be configured.

Protocols

Restricting the protocols to TLS 1.2

ProtocolsWhiteList=SSLv2Hello,TLSv1.2

Currently the supported protocols include SSL and TLS1 / TLS1.1 which are disabled by the above setting.

SSL / TLS1 / TLS1.1 are all considered insecure by today's standards, so only TLS1.2 should be permitted.

SSLV2Hello is retained as this is merely a handshake to determine which protocol to use - in this case only TLSv1.2

Ciphers

Many ciphers are available on the TLS connection, which may be negotiated automatically.

Restrictions on the ciphers used however is applied in the software.

Both a blacklist and whitelist are utilized to enable certain ciphers and disable others.

The cipher must be present on the whitelist and not present on the blacklist in order to be included in the permitted cipher list.

If the blacklist is not specified, only the whitelist will be checked.

For best security, a combination of whitelist and blacklist is configured by default.

If the cipher is a match for both the blacklist and whitelist, the cipher will be excluded.

A full list of ciphers permitted will be logged at the time a connection is attempted.

Recommended Settings (default)

```
#Crypto-Agility - Communications

#Protocols Secure setting

ProtocolsWhiteList=SSLv2Hello,TLSv1.2

#Cipher Secure setting

CipherWhiteList=TLS_DHE_.*_WITH_AES_128_.*,TLS_ECDHE_.*_WITH_AES_128_.*,TLS_ECDH_.*_WITH_AES_128_.*_.*,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA

CipherBlackList=SSL_.*,TLS_EMPTY_.*,.*_SHA,.*_3DES_.*,.*_DES_.*,.*_WITH_NULL_.*,.*_anon_.*_.*,.*EXPORT.*,.*LOW.*,.*MD5.*,.*DES.*,.*RC2.*,.*RC4.*,.*PSK.*
```

Full Crypto-Agility Settings List

```
#Crypto-Agility - Communications

#Protocols Secure setting

ProtocolsWhiteList=SSLv2Hello,TLSv1.2

#Protocols Default

#ProtocolsWhiteList=SSLv2Hello,TLSv1.2

#Protocols Java 1.6 setting for backwards compatibility
```

```
#ProtocolsWhiteList=SSLv2Hello,TLSv1.2,TLSv1,TLSv1.1

#Cipher Secure setting

CipherWhiteList=TLS_DHE_.*_WITH_AES_128_.*,TLS_ECDHE_.*_WITH_AES_128_.*,TLS_ECDH_.*_WITH_AES_128_.*,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA

CipherBlackList=SSL_.*,TLS_EMPTY_.*,.*_SHA,.*_3DES_.*,.*_DES_.*,.*_WITH_NULL_.*,.*_anon_.*_.*,.*EXPORT.*,.*LOW.*,.*MD5.*,.*DES.*,.*RC2.*,.*RC4.*,.*PSK.*

#Cipher Default

#CipherWhiteList=TLS_DHE_.*_WITH_AES_128_.*,TLS_ECDHE_.*_WITH_AES_128_.*,TLS_ECDH_.*_WITH_AES_128_.*_.*,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA

#CipherBlackList=SSL_.*,TLS_EMPTY_.*,.*_SHA,.*_3DES_.*,.*_DES_.*,.*_WITH_NULL_.*,.*_anon_.*_.*,.*EXPORT.*,.*LOW.*,.*MD5.*,.*DES.*,.*RC2.*,.*RC4.*,.*PSK.*

#Cipher Java 1.6 setting for backwards compatibility

#CipherWhiteList=

#CipherBlackList=SSL_.*
```

Crypto-Agility for Data Storage

Several of the cores now support crypto agility.

Each core uses a set of parameters in the [corename].properties file.

```
##############################

# Crypto-agility - keystore #

##############################

#Key Generator (example AES)

crypto.keygenType = AES

#Cipher Type (example AES/CBC/PKCS5Padding)

crypto.cipherType = AES/CBC/PKCS5Padding

#KeySize

crypto.keySize = 128

#No of iterations in keystore

crypto.iterations = 100000
```

This currently applies to Adyen, Cayan and Ocius Sentinel cores.

Currently shipping with the default settings above. The main improvement over existing default settings are to increase the number of iterations of encryption.

A mechanism has been provided to change the algorithm by running a command from the prompt with parameters including keystore location and encryption properties to use. Full details of crypto agility commends for each core is included in the *Oracle Retail EFTLink Core Configuration* guide.

EFTLink Cores

Each individual core may also have specific security requirements. These are detailed here:

Adyen Core

A password for Adyen is entered using the supplied configuration batch file.

The encrypted password is then stored in the adyen.properties file.

AJB FiPay Core

The AJB core may be configured to enable signature logging via the property

```
enable.signature.logging
```

By default, this is set to False. This should not be set to TRUE in a secure environment, as this will expose PII information - the customer signature.

Cayan Core

The cayan communication may be secured by employing an https connection over a secure socket.

This is done through the cayan.properties file:

```
#secure port
```

```
ced.port = 8443
```

Performing all Actions on the Pin Entry device over https is the default secure option:

http.action.add_item = https://cedIp:cedPort/v1/pos?Action=AddItem

http.action.del_item = https://cedIp:cedPort/v1/pos?Action=DeleteItem

http.action.del_items = https://cedIp:cedPort/v1/pos?Action=DeleteAllItems

http.action.start_order = https://cedIp:cedPort/v1/pos?Action=StartOrder

http.action.end_order = https://cedIp:cedPort/v1/pos?Action=EndOrder

http.action.cancel_order= https://cedIp:cedPort/v1/pos?Action=Cancel

http.action.status.check= https://cedIp:cedPort/v2/pos?Action=Status

http.action.get_tk = https://cedIp:cedPort/v2/pos?TransportKey=

http.action.keyed_sale = https://cedIp:cedPort/v1/pos?Action=InitiateKeyedSale

http.action.update_total= https://cedIp:cedPort/v1/pos?Action=UpdateTotal

http.action.discount_item=https://cedIp:cedPort/v1/pos?Action=DiscountItem http.action.update_item =https://cedIp:cedPort/v1/pos?Action=UpdateItem

Ocius Sentinel Core

User loginID and PIN are stored encrypted in the EFTLink core configuration file.

As such the following property is required NOT to be altered:

```
ocius.properties=ocius.keystore
```

This is the default value (secure by default).

OPI Retail Core

Logging of sensitive data defaults to secure settings and sensitive data is masked by default using the following setting:

```
MaskSensitiveDataLoggingEnabled=true
```

XML returned to the OPICore is also validated against a set of XSDs prior to parsing using the following setting:

```
ValidateOPIRetailResponse=true
```

Logging

By default, all logging is secured as each core includes a number of sensitive fields included in communication which will be masked.

All logged communications information is also scanned for numbers which could possibly be full card numbers and are masked by default.

Trace level logging which is not enabled by default should not compromise security of the application.

The default logger should be configured for info only (debug information should not be generated in production environment)

This is configured in the log4j2.xml file:

```
<Loggers>
```

```
<Root level="info">
```

Java

The application should be deployed with Java 1.8. Also, the latest version of the java framework should also be deployed.

There are several issues which require the latest version of the framework to be used in order to ensure a secure deployment:

TLS 1.2 requires the use of Java 1.8

Enhanced Cryptography, ensuring better secured communications.

Use of earlier versions of the java framework may require the configuration of TLS and Crypto agility to be altered which will result in decreased security.

References

The https://confluence.oraclecorp.com/confluence/display/GPS/OSSA%3A+Secure+Configuration+Program%3A+1.0`Oracle Secure Configuration Wiki` page provides information about secure configuration standards for Oracle. There are links to additional sample documentation and topics related to secure configuration.

`Oracle Retail Documentation` will allow navigation to published RGBU product documentation including links to the installation guide.

3

Secure Development

This chapter describes retailer and solution specific responsibilities for ensuring EFTLink is securely implemented and configured.

- Core-Host API
- Development Considerations
- Core How-Tos

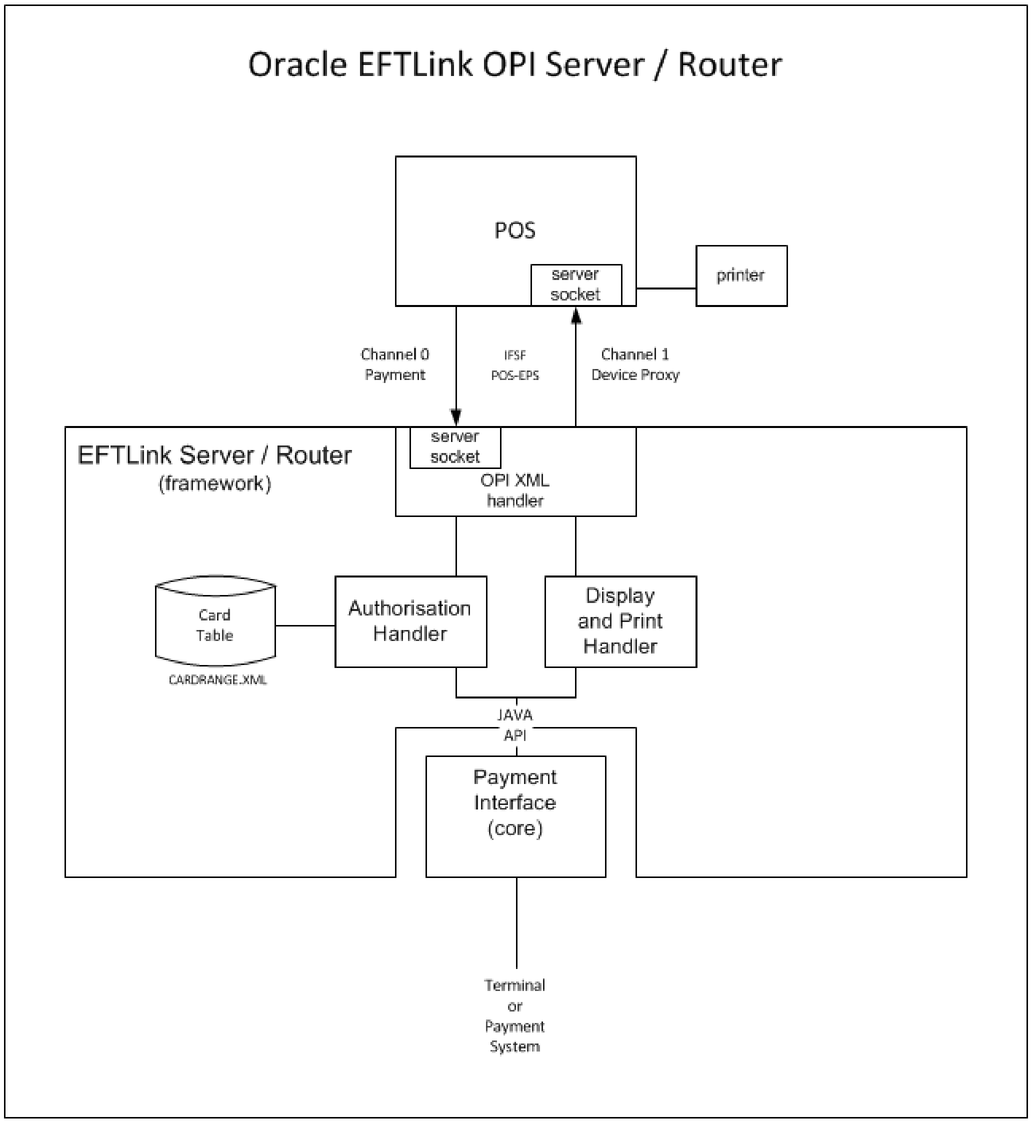
Core-Host API

A standard API is available to payment systems in the EFTLink framework.

Scope

The EFTLink framework presents a standard (Java API) interface for "plugin" payment system implementations, referred to as "cores". It is within the core that all business logic and communications processing that is required by that payment system is implemented. The framework provides an execution environment and access to POS resources such as operator screen/keyboard and printer. It is a best practice to have all Oracle Retail EFTLink support requests submitted through a single point of contact for that customer environment; the client designated administrator is usually designated to perform this role.

Figure 3-1 EFTLink OPI Server/Router

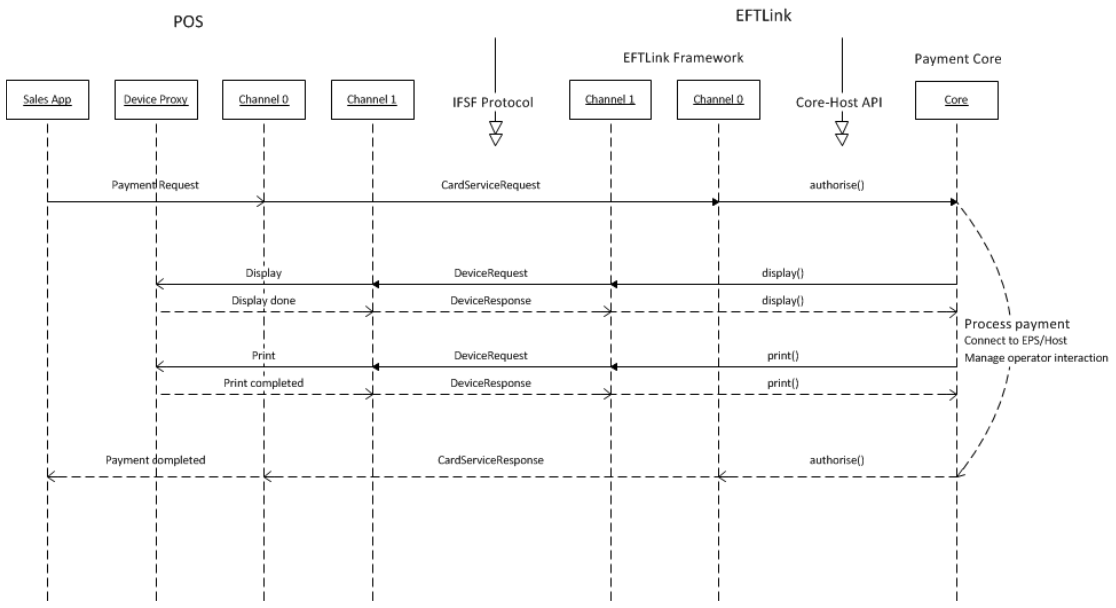


Message Flow

The EFTLink Core API is a set of Java classes, but it is essential when developing a new core to also understand the underlying XML socket protocol that is encapsulated in those classes, to keep in mind how each method call affects the channel/socket connections and to be aware of the generally multi-threaded nature of the architecture.

The diagram below describes the flow of messages from the client application through EFTLink to a payment core.

Figure 3-2 Message Flow



Channel 0

Channel 0 payment/admin request from the POS will be presented to the core as method calls (for example, authorise()). This is a blocking call in that the channel 0 response will not be sent to the POS until that method returns.

Channel 1

Channel 1 device proxy requests are triggered by the core calling a display/print method on the EFTLink framework host. This is a blocking call in that the method will not return until the relevant display/print process has completed on the POS or timed out.

Channel/Socket Contention

There can only be one active request per channel at any one time. Any attempt to use a channel that is already in use is taken as an implicit cancellation of the in-progress session. However, this rule is implemented differently on channel 0 and 1. On channel 0 (POs to EFTLink), the new request (typically a payment request) is rejected as "busy" and the active session is left to run to completion. On channel 1 (EPS to EFTLink), the old session is aborted, and the new request (typically a display request) is processed as the new active session.

Socket/XML Message Abstraction

The Core-Host API abstracts the socket handling and XML message content into a set of java classes and interfaces. Socket/channel activity is passed to the core as an event (for example, the authorise() method call). Message content is abstracted into entity classes. For example, EPSRequest/EPSCard and passed as arguments to those events. All the properties in the entity classes will have a directly related XML element or attribute, and vice versa.

The XML protocol specification should be used as the primary point of reference for how and when a particular property is used.

EPSCore/EPSHost

The key Interfaces that a core has to use are EPSCore and EPSHost. The core implements EPSCore and so receives events (for example authorization requests) from the EFTLink framework. The EFTLink framework implements EPSHost to allow the core to access framework resources and to interact with the POS.

The current version of EPSCore is EPSCore_1_3. Earlier versions are still supported for backward compatibility, but any new core should implement the highest available version to benefit from any recent enhancements.

Javadoc

Javadoc for the Core-Host API is available `here`.

EPSCore Event Mapping

The following table shows the mapping between the XML message and the API method.

Table 3-1 EPSCore Event Mapping

XML Message	API Method
ServiceRequest of type Logon	logon
ServiceRequest of type Logoff	logoff
ServiceRequest of type Reconciliation	reconciliation
ServiceRequest of type Administration	maintenance
CardServiceRequest of any type (such as CardPayment)	authorise
SaleStateNotification of any Type (SaleStart, SaleEnd)	posTransactionInProgress
any message of type AbortRequest	abort

EPSHost Action Mapping

The following table shows the mapping between the API method and the XML message.

Table 3-2 EPSHost Action Mapping

API Method	XML Message
display	DeviceRequest of type Output to CashierDisplay device (simple text display to operator)
print	DeviceRequest of type Output to Printer device (simple print of preformatted text)
deviceAccess	DeviceRequest of any type with extended attributes - timeout and so on.

EPSRequest/EPSResult Classes

The key entity classes that a core must use are EPSRequest and EPSResult. Most EPSCore events carry with them an EPSRequest data object. This holds all the details extracted from the POS XML message. On completing the requested action, the core builds an EPSResult object to return to the host to show the success/fail status of the operation along with all related data (for example payment card details).

DeviceRequest/DeviceResponse Classes

As part of processing the POS request, the core is likely to need to display and/or print messages at the POS. This is done by creating a DeviceRequest object specifying the type of display required and invoking the deviceAccess() method on the host interface. The host will build a DeviceResponse object to return to the core to show the success/fail status of the operation along with all related data (for example keyed data entered by the operator).

Mandatory Content

The core only needs to implement as much of the specified functionality as is required to fulfil the requirements of the payment system. However, where the protocol defines mandatory fields that are not covered by this approach, the core must populate those fields with suitable default values.

Development Considerations

A number of functions must be taken into consideration when developing cores to ensure a base set of functionalities is available.

Logging

EFTLink uses log4j and maintains a daily log file eftlink_mmdd.log. Core implementers are encouraged to integrate their logging into this system. For debug purposes, EFTLink also has a feature where all logging is also echoed to a pop-up dialog box. To support this, individual class logging should be initializsed by calling getLogger() from the EPSLogger class for example.

private Logger log = EPSLogger.getLogger(OPICardServiceSession.class);

Configurable Properties

Configurable properties should be put in a textual property file.

Multi-threading and Synchronisation

The system architecture is inherently multi-threaded. Each new call from the framework will be in a new thread and the communications implementation to the terminal/EPS is likely to introduce further threads. Care must be taken to ensure these threads are synchronized and that data is suitably protected.

Progress Messages

Ideally, the core should send regular progress display messages to the POS to keep the operator informed on the state of the transaction.

Most payment systems report their ongoing status, either by explicit display messages or some form of status broadcast, but where there is no such status update, the core should still endeavour to report progress as far as is possible.

Printer Management

The default way of handling printout is to send it directly to the POS as a device proxy request. However, it is also possible to buffer printout in the core and then include that text in the authorization response, to allow the POS to merge the EFT voucher into the standard POS receipt to save paper. Not all POS systems support this feature, so it should always be made configurable.

It is also possible to tag print lines as being "journal" so that they can be stored rather than printed. Again, not all POS systems support this feature.

Administration Functions

It is quite common for a payment system to have some maintenance/engineer functions that need to be executed when installing the terminal or on a regular basis. Such operations can be put on a maintenance menu, defined as series of device proxy display requests. The POS invokes this menu via an **EFT Maintenance** option that has no business logic associated with it, but just triggers an "Administration" service request and opens channel 1 for device proxy access.

Cancellation

The POS may have a "cancel" button which can be pressed to try to abort a transaction that has been started unintentionally or that is taking too long. This is done by the POS sending an AbortRequest, which in turn is passed to the core as abort(). This should be interpreted as a request not a command - if it is possible to cleanly close the active transaction, then this should be done, but if not, the request ban be ignored.

Translation

Textual display and print data received from the payment system should be shown as is. However ancillary display messages should be shown in the correct language for the region. EFTLink has a "Text" translation class, and a set of LangXX.properties country-specify lookup files.

EFTLink Server Compatibility

When deployed in Server mode, there will be multiple instances of the EFTLink main application class (OPIServer) and multiple instances of the core class, all in memory at the same time. Each instance runs from its own set of configuration files in a sub-folder under the main EFTLink folder.

This imposes rules/restrictions on the core:

- No hard-coded file paths. Always access files relative to INSTANCE_ROOT() or DATA(). Only use PRIMARY_ROOT for files that are intentionally the same for all instances.

  > **Note:**
  >
  > When using framework file access helper classes such as EPSFileProperties, EPSSecureFile and so on, the use of INSTANCE_ROOT() is automatically applied by the framework.

- Do not store instance-specific data in static storage. Static storage will in effect be shared between all instances.
- No passing files (for example signature images) by name. The POS will be on a remote system.

Core How-Tos

Additional information regarding the core:

- MiscellaneousData
- Session Properties
- Secure Data
- Restricted Access
- Key Storage
- Encrypted File Storage
- Security Standards

MiscellaneousData

MiscellaneousData is a free-format text property, available in both the EPSRequest and EPSResult classes, which can be used for any data that is not covered by the standard property options. It can be accessed as a String in the normal way, but there is also an option to reference it as MiscellaneousDataProperties to access it as a collection of key-value pairs but note that the underlying storage is still the same MiscellaneousData String.

Miscellaneous Data Disclaimer

EFTLink along with some selected Cores, has the ability for additional data to be sent and received in a field called `<MiscellaneousData>`.

This can be used by System Implementers (SIs) and Payment Service Providers (PSPs) to pass additional data in the messages between Xstore and the Payment Providers, using custom code.

Typically, this is used to add directives which we can trigger different payment workflows however it can also be used to capture additional payment data for downstream processing for the Retailer's to use for reconciliation or financial purposes.

Under no circumstances should any PCI or potentially sensitive PII data be placed in this field. Oracle will not be responsible for any issues caused by integration changes made by SIs, Retailers and Payment Providers, that enable sensitive data to be added into this field.

Session Properties

If the POS has included data in the XML to EFTLink that for some reason is not mapped into the existing API objects, there is still a means of drilling down into the Session for more detail. The EPSHost Interface includes a getSession() method returning an EPSSession object. This is primarily there to allow access to session variables when processing methods that do not use EPSRequest, such as logon() or maintenance(), but the EPSSession Interface reference can also be cast back to the appropriate underlying OPIXxxSession object to get access to the relevant session properties and even to the XML Document objects.

Secure Data

EFTLink Framework provides three security features to protect data used by the core.

By using these features rather than any bespoke implementation in the core, the security system is made more future-proof because changes (for example encryption enhancements) need only be applied in the framework.

Note:

Sensitive text data should not be stored in Java Strings. Better to use a character array so that the memory can then be explicitly cleared after use.

Restricted Access

The framework provides a DATA() folder with restricted user access rights. Any files created in that folder will only be accessible by the EFTLink host account.

Key Storage

The EPSKeyStoreManager class can be used to store encryption keys (or in fact any text-based data). The data will be encrypted and stored in a password-protected Java Keystore, transparent to the core.

Encrypted File Storage

The EPSSecureFile class can be used to store data in an encrypted file. The read/write interface presented by EPSSecureFile is based on streams, to avoid restricting it to any specific data type. Any data that can be expressed as a stream can be encrypted and serialized in this way. One way of doing this is to define a new serializable entity class to hold all the relevant sensitive data and interface with EPSSecureFile using ObjectInputStream and ObjectOutputStream.

Examples of each of these are included in the EPSSimulator core class.

Further information is available at the following location:

`https://confluence.oraclecorp.com/confluence/display/EF/OPIClient-based+Cores`

Security Standards

All development must be performed in accordance with the Oracle Software Security Assurance Standards.

The Oracle Secure Coding Standards (SCS) are mandatory coding standards for all developers.