Enterprise Integration Platform

Technical Overview
December 2006

PilotFish Technology

Contents

Overview

This document describes the design approach, technical requirements, and installation, configuration, and administration processes for the PilotFish Technology Enterprise Integration Platform (eiPlatform, or EIP).

Chapter 1

Introduction to the eiPlatform

1.1 Purpose

The eiPlatform is an architecture created by PilotFish Technology to facilitate the integration of disparate systems in a logical, standardized way, faster and more efficiently than was previously possible. It is a flexible, extensible solution that can be easily adapted to meet any requirement or fit into a pre-existing architecture. Centered around XML but capable of reading almost any proprietary and/or legacy format, the EIP provides a way to bring legacy systems into the 21st century with a minimum of disruption and effort.

1.2 Design Approach

The eiPlatform handles transformations using a many-to-one-to-many approach. Rather than transform data directly between two different formats, all source data is transformed into an XML standard (such as ACORD XMLifE, ACORD P&C, HR-XML, or a company-specific schema), and from there into the appropriate target format. This approach minimizes the amount of work required to connect multiple systems, and maximizes the potential for extension and reusability.

Chapter 2

Architectural Overview and Transaction Lifecycle

Interfaces on an EIP are deployed by configuring instances of two types of object. A “Route” contains the logic to receive a transaction from an originating (source) system, perform the appropriate transformations to it, and deliver the transaction to the required destination (target) system(s). “Formats”, on the other hand, define the logic required to transform transaction data to and from the XML standard deployed on an EIP.

2.1 Route Objects

A single Route deployed on an EIP represents a single, unidirectional transaction type, moving from an arbitrary number of sources (originating systems) to an arbitrary number of targets (destination systems). The sections below step through the components of a Route, and the lifecycle of a transaction moving through them.

2.1.1 Route Sources

A source in a route is defined by the following items:

Listener Instance

All data entering the EIP does so through Listeners. These are modular components that are capable of receiving data through a specific mechanism and/or protocol. Examples include directory search, FTP, JMS, POP3, web service, and so forth.

Every source in a Route contains a Listener instance, and any configuration information that it requires.

It is important to note that Listeners do not do anything to the data that they receive - their sole responsibility is to relay it to the rest of the EIP. Transformation, validation, and so forth are handled later.

EIP Initial Checkpoint

This is not a configurable component of a Route, but instead a stage that serves as the starting point for all transactions moving through the EIP.

Once the EIP receives data from a Listener, it performs three actions:

Assignment of a Transaction ID Every transaction moving through the EIP is assigned a unique identifier, generated from a database sequence or similar mechanism. This ID is used to identify this transaction in logs and error reports.

Checkpointing of Source Data After assignment of a transaction ID, the transaction's data is cached to disk. This serves two purposes: it ensures that the entire data stream passed from the Listener is readable, and preserves a copy of the transaction for automatic restart (in case of a power failure or system crash), or manual restart (in case of an transient error or misconfiguration).

Recording in Logs After the transaction's data has been cached, a log entry is written to the PF_EIP_TRANSACT table, containing the transaction's ID, the name of the Route it is moving through, the name of the Listener it was received from, the date and time it was received, and other information.

After the last step is complete, the transaction's integrity is protected by the EIP's error recovery facilities. The only time the EIP can lose a transaction's data is during the execution of these three steps.

Processor Instances(s)

After the EIP has received data from a Listener, it may be optionally be passed to any number of Processors. These modules perform low-level operations on the data in the transaction, such as EBCDIC to ASCII conversion, decryption and decompression, XML schema validation, and so forth. They do not perform any operations on individual data elements.

Every source in a Route may contain an arbitary number (including zero) of Processor instances, along with any configuration information that they require.

Format Reference

Every source in a Route also references a Format. Formats are responsible for performing all the data transformation that occurs in the EIP. Once data is received from a source's Listener (or the last Processor, if any have been specified), it is passed to the Format for transformation. In the case of a source, the Format is responsible for transforming the transaction's data from whatever format it was in, to the XML standard deployed on the EIP. Formats also handle the forking and joining of transactions: these are, respectively, the dividing of a single transaction into multiple transactions, and the combination of multiple transactions into a single transaction.

See below for more information on what makes up a Format, and how the EIP's forking and joining systems work.

2.1.2 Routing

Once a transaction has been transformed by the Format associated with the source that it was received from, it is passed to the routing stage. Here, another component calling a Routing Module looks at the transaction (now transformed into standardized XML), and determines which of the target systems it should be sent to.

A Route may only have one Routing Module. If no Routing Module is specified, the transaction will be sent to all targets specified in the route.

Transactional Database

Optionally, once a transaction has been transformed into standardized XML, it can be permanently stored in the transactional database. This is a SQL version of the transaction's XML schema, built by the XCS Database Generator, used for auditing and control purposes. See the XCS Technical Documentation for further details.

2.1.3 Targets

Once a transaction has passed the routing stage, it is passed to the target systems specified by the Routing Module. Targets are essentially mirror images of sources. They reference a Format, which transforms the transaction from standardized XML into the format required by the destination system. Like sources, they may also contain any number of Processors. In this context, they perform the same operations they do for sources, but in reverse. Encryption instead of decryption, compression instead of decompression, and so forth.

In addition to these previously-discussed components, targets also contain a Transport instance.

EIP Final Checkpoint

Similar to the EIP Initial Checkpoint that occurs just after a transaction is received from a Listener, the final checkpoint occurs just before a transaction is sent to a Target.

Here, the data transformed for the target system is cached to disk, and the appropriate log entries are recorded in the PF_EIP_TRANSPORTLOG table. Once this is complete, the original cache of data from the Listener is deleted.

This functionality allows the Transports in the Route to operate independently, while still preserving data integrity: a failure in one Transport (usually due to transient network or connectivity issues) does not automatically result in the failure of all others in the Route, nor does it result in the loss of the data that was going to be sent: a system administrator can manually retry the Transport, using the data from the on-disk cache, at a later date.

Transport Instance

Transports in targets are the opposite of Listeners in sources. Instead of reading data into the EIP, they are responsible for sending data out. Nearly every Listener supplied with the EIP has a corresponding Transport, capable of sending data via the same mechanism that the Listener receives it.

Once a Transport has successfully executed a transaction, the on-disk cache created during the final checkpoint is deleted.

2.1.4 Transaction Monitors

In addition to the modules already described, a Route may also have any number of Transaction Monitors associated with it. Transaction Monitors are notified of errors that occur during transaction processing, and may take action such as sending an email, raising an SNMP trap, communicating with an online pager service, or activating a separate error-processing route. These routes may perform more advanced processing, such as sending an error notification back to the originating source system as a fixed-width file, placing an alert record in a database, and so forth.

2.2 Format Objects

Formats have already been mentioned in the context of sources and targets within Routes. Each Format object corresponds to a specific data format and/or transaction type, such as a MIB-COMM Search Request. They are responsible for translating data to and/or from whatever format they support, and the XML standard deployed on the EIP. When referenced by a source in a Route, a Format must transform into the XML standard, and when referenced by a target, out of it.

Formats are not required to support transformation in both directions. A Format that is referenced only by sources does not need to be capable of transforming data from standardized XML, and the same holds true in reverse for Formats referenced only by targets.

To illustrate the components that make up a Format, we will look at the processing of a transaction when it is received from a source. Formats use a two-stage transformation: first, a Transformation Module converts the transaction data from the format it was in to some kind of XML, and then an XSLT stylesheet is applied to convert that "generic" XML into standards-compliant XML. This process operates in reverse for data that is about to be sent to a target: the XSLT stylesheet is applied first, and then the Transformation Module is invoked. In between these two steps, a Forking or Joining module can intercede in order to manipulate the transaction as well.

2.2.1 Transformation Module

Once any Processors specified in a source have finished with a transaction, it is passed to the Transformation Module specified in the referenced format. These modules are responsible for XML-izing and de-XML-izing data: for instance, wrapping fields in fixed-width and delimited files with XML tags, or creating serialized Java objects based on an XML file. In cases where the file format a Format supports is already some dialect of XML, a Transformation Module is not required.

Note that Transformation Modules do not deal directly with standards-compliant XML. The XML data they produce and consume is merely an intermediate format, that is transformed to or from standards-compliant XML using XSLT.

2.2.2 Forking and Joining Modules

Once the Transformation Module (for sources) or XSLT transformer (for targets) has finished with a transaction, it can be passed to a Forking or Joining module. Forking refers to the dividing of data from a single transaction into multiple new transactions, for reasons of resource efficiency, routing flexibility, and so forth. Joining refers to the combination of data from multiple transactions into a single new transaction, and it is used to implement caching and buffering, reconstitution of batch transactions that were previously split for transformation, and other functionality.

For sources, a Forking module can be invoked after the Transformation Module but before the XSLT transform, and for targets, a Joining Module can be invoked after the XSLT transform but before the Transformation Module. For reasons that will become clear, it is not possible to join transactions moving through a source, or fork transactions moving through a target.

Lifecycle of Forked and Joined Transactions

Transactions that are manipulated by Forking and Joining modules follow a slightly different lifecycle than those that simply move straight through undisturbed.

When a transaction is forked, it is immediately halted. In its place, new "child" transactions are created, with new transaction IDs, each containing a subset of the original transaction's data. These

child transactions continue to execute, at the point in the Route immediately following the Forking Module.

The behavior of joined transactions is similar, but not identical. In a group of transactions being joined together, all but one of them ceases to execute. The data from all halted transactions is combined with the data in the single remaining transaction, referred to as the "master", which then continues to execute at the point in the Route immediately following the Joining Module.

The use of Forking and Joining modules also presents special problems for logging and error handling. Please read the"Monitoring and Error Recovery" section below for more information on this before using a Forking or Joining module in your interfaces.

2.2.3 XSLT Stylesheets

Formats use XSLT stylesheets to handle the transformation between the "generic" XML produced by Transformation Modules, and the XML standard deployed on the EIP. Separate stylesheets are defined for source and target transformations. If a stylesheet is not defined for a source or target, no transformation is performed.

2.3 Extensibility

The EIP is designed to allow for maximum extensibility. The API's for all the modules described above (Listeners, Transports, Processors, Transformation Modules, Forking Modules, Joining Modules, Routing Modules, and Transaction Monitors) are public, documented, and easy to implement using Java. If the modules supplied with the EIP do not provide the functionality required for a specific application, writing one that does is a simple exercise.

The section "Developing Custom Modules" below has more information on this topic.

Chapter 3

Installation

3.1 System and Deployment Requirements

The EIP has few dependencies. It requires only an installation of Java and a SQL-compliant relational database. It will run adequately on low-end systems under light load, but CPU, memory and disk I/O requirements increase linearly with transaction volume.

3.1.1 Standards-Level Requirements

- J2SE 1.3 or greater
- J2EE 1.3 or greater (only for deployments requiring J2EE features beyond servlets
- Java Servlet container supporting JSS 2.3 or greater
- SQL-compliant database with an available JDBC driver.

Below are the major versions of software that PilotFish has verified compatibility against. This is not an exhaustive list, and the fact that some product (or product release) is absent from this list doesn't indicate that it can't be supported. Adding support for a software product that meets the standards-level requirements above is included in the cost of an installation and/or support agreement.

3.1.2 Java Virtual Machine

- Sun JVM 1.3 and greater
- Apple JVM 1.4 and greater
- IBM JVM 1.4 and greater

3.1.3 Operating Systems

- Windows 2000
- Windows XP
- Windows Server 2003

- Linux (RedHat, Gentoo, Debian, SuSE)
- Mac OS X 10.3 and 10.4 (Client and Server)

3.1.4 Servlet Containers

- Oracle 9i and greater
- JBoss 3.0 and greater
- Apache Tomcat 4.1 and greater
- Jetty 5.0 and greater
- IBM Websphere 5 and greater
- BEA Weblogic 7 and greater

3.1.5 Databases

- Oracle 8i and greater
- Microsoft SQL Server 2000 and greater
- Microsoft Access 2000 and greater
- IBM DB2 v7 and greater
- MySQL 3.23 and greater
- PostgreSQL 7.3 and greater

3.2 Installation Methods

An EIP instance can be deployed on a server in several ways: it can be installed as a standalone server process, deployed as a servlet in a Java servlet container or application server, or integrated into a custom application using the EIP API. Each method has it's own advantages and disadvantages.

3.2.1 Servlet Deployment

One way of installing the EIP is as a servlet in an existing servlet container or application server. This allows it to co-exist with other applications and/or security mechanisms, and is usually the best solution for an organization that has experience with deploying and maintaining Java web applications.

The EIP is provided as a WAR file. To install the EIP as a servlet, simply deploy the WAR file as your application server requires, and follow the instructions in the "Database Initialization" section below.

3.2.2 Standalone Deployment

Alternatively, the EIP can be installed as a standalone application. This avoids the need to configure it as a web application in an existing servlet container. The disadvantage of this deployment mechanism is that it gives you less control over logging, security, and integration with existing software.

Nevertheless, this is the best option for organizations looking for a low-maintenance EIP installation that requires little configuration. The EIP is provided as an installer for the appropriate platform: simply run the installer and follow the instructions.

3.2.3 API Integration

It is also possible to create an EIP instance from within Java code. This allows maximum flexibility, but the EIP Administration Servlet is not currently supported through this deployment method.

In addition, the public Java interfaces to the functionality the administration servlet provides have not yet been released. Therefore, EIP instances created from within non-PilotFish code do not currently support administrative functionality. PilotFish support can provide early access to these interfaces if this poses a problem for a specific implementation.

See the `com.pilotfish.eip.server.EIPServer` class Javadoc for the details of how to manually create an EIP instance.

Chapter 4

Runtime Configuration

4.1 Configuration Parameter Reference

The EIP can accept runtime configuration parameters from any of three places: a file named eipServer.conf located in its working directory, the `PF_EIP_CONFIG` table in the database it connects to, or a Java Properties object passed directly to an EIPServer instance. Database connection parameters must be supplied either through the eipServer.conf file or Properties object, but all others can be specified in any location. Parameters specified in the database override those in a Properties object or configuration file.

The table below describes the configuration parameters that the EIP accepts. All parameters beyond the database connection settings and route configuration directory are optional.

com.pilotfish.eip.database.plugin Specifies the database plugin to connect with. To use one of the built-in plugins, set this to `pilotfish:` followed by `Access`, `DB2`, `Oracle`, `sqlServer`, or `MySQL`.

To use a custom class implementing `XCSDatabasePlugin`, set this to "custom:" followed by the fully-qualified name of the class to use. For instance, `custom:com.company.eip.CustomDatabasePlugin`.

com.pilotfish.eip.database.url Specifies the JDBC URL to use to connect to the database. For instance, `jdbc:mysql://localhost/eip`.

com.pilotfish.eip.database.username Specifies the username to connect to the database with.

com.pilotfish.eip.database.password Specifies the password to connect to the database with.

com.pilotfish.eip.database.minConnections Specifies the minimum number of database connections to keep alive. On high-volume EIP instances, increasing this parameter will decrease response times at the expense of resource efficiency.

com.pilotfish.eip.database.maxConnections Specifies the maximum number of database connections allowed. Any EIP components that request a database connection once this limit has been reached will wait for one to become available. Set this parameter low enough to avoid bogging down the database, but high enough to avoid unnecessary bottlenecks.

com.pilotfish.eip.configDirectory Specifies the directory to read route/format definitions from. See "Developing Routes and Formats" below.

com.pilotfish.eip.modelReferenceLocation Specifies the XCS model reference file the EIP should use when parsing data into the transactional database. This file is generated during EIP installation by the XCS Database Generator. This parameter is not required if no Routes are configured to store transactions in the transactional database.

com.pilotfish.eip.logConfigFile Specifies the Apache Log4J configuration file to initialize the logging subsystem with. A default file is provided with the EIP distribution that should work well in most situations. It is automatically selected if no value for this parameter is supplied.

See the "Server Logs" section below, as well as the Apache Log4J documentation (`http://logging.apache.org/log4j/`), for further information.

com.pilotfish.eip.transact.baseThreadCount Specifies the number of concurrent transactions to allow through the EIP under normal conditions. If a transaction is received when `baseThreadCount` transactions are already being processed, it is held temporarily in the incoming transaction queue until a thread is available to process it.

com.pilotfish.eip.transact.queueSize Specifies the size of the queue that stores received transactions when `baseThreadCount` has been reached. The purpose of the incoming transaction queue is to act as a buffer, allowing the server to gracefully handle bursts of heavy volume in which transactions arrive faster than they can be processed efficiently. If a transaction is received while the queue is full, then additional threads are created up to the maximum specified by `maxThreadCount`.

com.pilotfish.eip.transact.maxThreadCount This specifies the absolute maximum number of concurrent transactions to allow through the EIP. In situations involving extraordinarily high transaction volume, when `baseThreadCount` transactions are already active and the incoming transaction queue is full, the server will create additional threads up to the number specified by this parameter in order to supplement those that were created normally.

com.pilotfish.eip.transact.idleTimeout Specifies how long to keep idle extra server threads (threads beyond those created to satisfy `baseThreadCount`) alive until they are killed.

Chapter 5

Developing Routes and Formats

Routes and Formats are defined through XML files in the EIP configuration directory. Specified through a configuration parameter, this may exist anywhere on the EIP server where the EIP process has read and write access. Inside the configuration directory are two subdirectories: `/routes` and `/formats`. These contain one directory for every Route and Format (respectively) deployed on the EIP.

5.1 Configuration Directory Layout

The Route and Format configuration directories share a similar layout. Both contain the object's definition file (named `route.xml` or `format.xml`), as well as any additional files that it requires. These may be placed anywhere in the directory, with the exception of Java classes. They must be placed in `/lib` under the Route or Format's configuration directory, where the EIP will automatically look for and load them as required.

Note that it is also possible to use custom Java modules in Routes and Formats by including them in the JVM classpath. However, placing the classes under the individual Route or Format's directory is the recommended approach. It allows for finer-grained version control, allowing you to make a change to code deployed with one Route without worrying about retesting the changes with the others. In addition, using this approach guarantees compatibility with internal PilotFish diagnostic and remote administration tools.

5.2 Developing Custom Modules

As mentioned previously, it is possible to develop custom modules for use in Routes and Formats, beyond those provided with the EIP distribution. EIP Modules are defined as Java interfaces. The relevant `.class` files are available in the `eip-api.jar` file bundled with the EIP distribution. Javadoc is provided as well.

This section is intended to provide a high-level overview of what is involved in writing custom Listeners, Transports, Processors, Tranformation Modules, Routing Modules, and Transaction Monitors. For specific implementation details, consult the Javadoc on the specific module interface you are implementing.

5.2.1 Abstract Classes

Abstract implementations for all EIP Module types may be found in `com.pilotfish.eip.extend` package. These provide implementations of all methods that should not vary in functionality between modules. To ensure compatibility with future EIP releases, in which module features and functionality may change, PilotFish recommends that users developing custom modules extend these classes rather than implement the module interfaces directly.

5.2.2 Module Naming

All modules in the EIP (with the exception of Transformation and Routing modules) are named for the purposes of logging and error recovery. The name is specified in the module's XML definition element, and passed to the module by the EIP infrastructure via `setName()`. The module is responsible for remembering it, and returning it from `getName()`. The various abstract module implementations handle this automatically.

5.2.3 Description Methods

For display purposes in graphical configuration and administration tools, all modules should provide a description of the functionality they provide. `getType()` should return a one- or two-word String (such as "SMTP Transport"), and `getDescription()` a slightly longer version ("Sends data to the specified SMTP server / email address").

5.2.4 Graphical Configuration

Because the Route and Format definition file formats allow modules to use any configuration format they want, every module should provide its own Swing-based graphical configuration component, through the `getConfigurationComponent()` method. This is not necessary if you do not expect to configure your EIP installation using graphical configuration tools. If this is the case, or if the module you are writing does not require any configuration, return `false` from `supportsGraphicalConfiguration()`.

5.2.5 Configuration Readiness

When a module is initialized, it is first given its name via the `setName()` method. Then, the module is passed a DOM `Element` corresponding to the `ModuleConfig` element under its definition, through the `readConfiguration()` method. From this, the module can read its configuration information.

`readConfiguration()` implementations may not throw checked exceptions. If an error is found in the module's configuration, it should be logged if necessary (see the "Logging" section below) and the module should return `false` from `isConfigurationComplete()`.

5.2.6 Locating Resources

Modules may rely on external resources located in their Route or Format's definition directory. These may be accessed through the `ResourceLoader` class, which provides methods for accessing files and instantiating Java classes. An instance of this class is provided to the module as a parameter to the `readConfiguration()` method.

5.2.7 Logging

The EIP uses the Apache Log4J library (`http://logging.apache.org/log4j/`) for logging. Modules should use only the Logger named `com.pilotfish.eip.extend`, or any of its children. The Logger may be obtained by importing the `org.apache.log4j.Logger` class and calling `Logger.getLogger("com.pilotfish.eip.extend")`.

Developers are encouraged to include as many logging statements in their modules as they feel necessary. Recording of debugging and trace statements may be easily disabled by editing the server's Log4J configuration. See the Server Logs section below for details.

5.2.8 Error Handling and EIPException

All EIP module methods that may throw exceptions throw `EIPExceptions`. If an error needs to be reported in a method that does not, it should be logged. To aid in debugging, error messages in thrown `EIPExceptions` should be as complete and descriptive as possible. If an `EIPException` is being thrown as a result of another exception being caught, the original exception should be passed to the `EIPException` constructor.

Chapter 6

Monitoring and Error Recovery

Once installed and configured properly, the EIP is a low-maintenance piece of software. The few administrative tasks that it requires are described in this section, along with how to access the server's logs and transaction records.

6.1 Server Logs

The EIP's server log file records transaction debugging information as well as statistics, warnings and error messages. It should be reviewed periodically for any unexpected behavior.

The file is located by default at the path `logs/eip.log`, which is specified relative to the EIP's working directory at launch time. The format of the file is as follows:

```
Date/Time Message-Priority [Message-Category]: Message
```

6.1.1 Message Categories

The `Message-Category` field indicates which EIP component issued the logging message. The basic categories are described below:

com.pilotfish.eip.server Used for general server messages: startup, shutdown, and any functionality controlled by runtime parameters.

com.pilotfish.eip.config Used by the EIP's route/format configuration management mechanism.

com.pilotfish.eip.transact Used by the EIP's transaction handling system. Contains trace, debugging, and status messages, along with debugging context information for any errors that occur.

com.pilotfish.eip.admin Used by the EIP's administration system. Contains messages indicating manual restart of failed transactions, manual activation/deactivation of Listeners and Transports, and other operations provided by the EIP Administration Servlet.

com.pilotfish.eip.extend Used by EIP modules to record any information that they deem pertinent. Usually status and debugging messages.

6.1.2 Message Priorities

The EIP uses several priority levels to identify the severity of any given log message. The logging subsystem can be configured to ignore all messages below a certain priority.

By default, all messages with a priority of INFO or higher are logged.

FATAL Used to indicate severe errors that will result in execution halting: for instance, a failure to connect to the EIP database.

ERROR Used to indicate errors that result in failure to perform an operation (such as a transaction), that do not affect other areas of the EIP.

WARN Used to warn of abnormal operational conditions that did not result in an operational failure, such as the inability to delete a transaction cache file. While they generally do not indicate a serious problem, these messages should not be ignored.

INFO Used to provide high-level status information. For instance, startup and and shutdown messages

DEBUG Used to provide detailed debugging trace information.

6.1.3 Log4J Configuration

The EIP's logging subsystem uses Apache Log4J (`http://logging.apache.org/log4j`). While the default logging configuration should be sufficient for most applications, it can be completely customized by supplying a different Log4J configuration file with the `com.pilotfish.eip.logConfigFile` runtime parameter.

Your PilotFish support representative will be happy to assist you with whatever customization you require. That said, documentation on the Log4J configuration file format is available at the URL listed above.

6.2 Transaction Logs

In addition to the server log file, the EIP also provides more structured transaction records in three database tables, described below. The table column types are described as the classes they correspond to in the `java.sql` package, as the actual data type names vary between database vendors.

6.2.1 PF_EIP_TRANSACTIONLOG

This table contains one row for every transaction that enters the EIP. Its fields are:

transactionID A String field containing the transaction's unique ID.

receivedTime A Timestamp field containing the time that the EIP was passed the transaction after the Listener received it.

listenerName A String field containing the name of the Listener that received the transaction.

routeName A String field containing the name of the Route that the receiving Listener was part of.

transformCompleted A Boolean field indicating whether the transaction was successfully transformed into all the required target formats. See the Error Recovery section below for more details. If this field is `false`, the specific error that was encountered may be found in the `PF_EIP_ERRORLOG` table.

cancelled A Boolean field indicated whether an administrator marked the transaction as cancelled, following an unrecoverable error. This field is provided solely for reporting purposes.

forked A Boolean field indicating whether this transaction stopped execution because it was forked into multiple subtransactions.

forkParentID A String field containing the ID of the transaction this transaction was forked from, if it is a forked subtransaction, or nothing if it is not.

joinedWithID A String field containing the ID of the transaction this transaction was joined with, or nothing if it was not joined.

Note the subtle difference between `forked` and `forkedParentID` - the former is used by transactions that produced subtransactions, while the latter is used by the subtransactions themselves.

6.2.2 PF_EIP_TRANSPORTLOG

This table contains at least one, and possibly more rows for each transaction that passes through the EIP. Each row corresponds to a single Transport in the transaction's route, and thus a single target system. Its fields are:

transactionID A String field containing the unique ID of the transaction this target/Transport was part of.

transportName A String field containing the name of the Transport.

sentTime A Timestamp field indicating the time the Transport executed, if no errors were encountered.

completed A Boolean field indicating whether the Transport executed successfully. See the Error Recovery section below for more details. If this field is `false`, the specific error that was encountered may be found in the `PF_EIP_ERRORLOG` table.

cancelled A Boolean field indicated whether an administrator marked the transaction as cancelled, following an unrecoverable error. This field is provided solely for reporting purposes.

6.2.3 PF_EIP_ERRORLOG

This table contains records of any errors that occur during the processing of transactions in the EIP. Each row in the table corresponds to a single error, and may be linked to the `PF_EIP_TRANSACTIONLOG` and `PF_EIP_TRANSPORTLOG` tables with the `transactionID` field. Its fields are:

transactionID A String field containing the unique ID of the transaction in which the error ocurred.

errorTime A Timestamp field containing the time the error occurred.

errorStage A String field containing a description of the stage the transaction was in when the error occurred. Allowable values are: `Listener`, `Transport`, `XSLT`, `Transformation`, `Processor`, `Routing`, `Infrastructure`.

errorComponentName A String field containing the name of the module that the error occurred in. For example, if the error occurred in a Processor, this will be the value of `Processor.getName()`.

errorMessage A String field containing a description of the error that occurred. More detailed debugging context information may be found in the server's log file.

6.3 Transactional Database

In addition to the transaction log tables outlined above, the EIP is capable of logging the contents of transactions in its transactional database. This database is built from the XML schema files defining the XML standard deployed on the EIP, and the data is logged when it is in that standardized XML format.

This behavior may be enabled or disabled through the `logTransactionContent` flag on the `Route` element in a Route definition.

The structure of this database is described fully in the XCS Technical Overview, which is available directly from PilotFish or as part of the XCS Toolset public download.

6.4 Route Readiness and Activation

When the EIP server first starts, it queries all deployed Routes and Formats to determine whether they are ready to process transactions. If any of them are not, they are disabled, and the remaining Routes and Formats are initialized normally.

Any Routes and Formats disabled for this reason are displayed in the EIP Administration Servlet, as well as in the server's startup log. The object's name is recorded along with whatever configuration problem it reported. If a Route or Format does not seem to be loading properly, check in either location before trying anything else.

Once the problem that disabled the Route or Format has been fixed, it will be activated the next time the server starts.

6.5 Error Recovery

The EIP includes robust error recovery facilities. Transactions moving through the EIP pass through two "checkpoints". The first occurs when they first enter the EIP - the transaction data is cached to a folder on the hard disk, and the receipt of the transaction is recorded in the `PF_EIP_TRANSACTIONLOG` table. The second checkpoint occurs once the transaction data has been transformed into all the required target formats: the data is cached to disk in separate files (one for every target system/Transport), and the first cache is erased.

If an error occurs before the second checkpoint, the first cache is not erased, and the error is recorded in the `PF_EIP_ERRORLOG` table and flagged in the EIP Administration Servlet. Once whatever condition caused the error has been identified and dealt with, the transaction can be retried from the cached data. It may also be cancelled, which is merely an administrative action indicating that the error was unrecoverable (for instance, because the source data was corrupted or in an incorrect format).

After the second checkpoint, the Route's Transports are invoked with the second set of cached data. Once a Transport succeeds, its cache is deleted. Just as with the transformation stages of the transaction, if a Transport fails, it can be restarted from the cached data at a later date.

Note that all transformations in a transaction must complete successfully before any Transports are invoked. However, Transports operate independently - an error in one Transport does not affect the others.

6.5.1 Error Recovery of Forked and Joined Transactions

Forked and Joined transactions are handled slightly differently when an error occurs.

If a forked subtransaction encounters an error, every other subtransaction that was forked from the same original transaction is halted with an error as well. Similarly, if a transaction that has had other transactions joined to it encounters an error, every transaction that made up the joined transaction is halted with an error.

This mechanism prevents the loss of data from undetected configuration mistakes or errors in the forking and joining modules. The halted transactions can later be restarted manually by an administrator, once the cause of the problem has been isolated and fixed.

6.5.2 Auto-Recovery of Incomplete Transactions

If the EIP is shut down irregularly in the middle of processing a transaction, it will automatically resume the transaction when it restarts. This facility ensures that uncontrollable events such as hardware errors or power failures will not result in data loss.

Appendix A

Supplied Module Reference

This section contains descriptions of the functionality and configuration requirements of all the modules supplied with the EIP distribution.

A.1 Configuration Format

All stock EIP modules, with the exception of the XPath-based Routing Module, use a property-value configuration mechanism. These properties should be specified as XML tags under the module's `<ModuleConfig>` element. For instance, to set the `PollingDirectory` property to `c:\Incoming`, use the following block of XML.

```
<ModuleConfig>
    <PollingDirectory>c:\Incoming</PollingDirectory>
</ModuleConfig>
```

A.2 Listeners

A.2.1 Directory Listener

`com.pilotfish.eip.modules.DirectoryListener`

A Directory Listener scans a single directory for files. Each file found is considered a single transaction. The polling interval, and action to take with the files are both configurable.

PollingInterval Specifies the number of seconds to wait between polling attempts.

PollingDirectory Specifies the absolute path to the directory to poll for files.

FileNameRestriction Specifies a pattern to match file names against. Only files that match will be processed.

FileExtensionRestriction Specifies a space-separated list of extensions to match. Only files with extensions in the list will be processed.

PostProcessOperation Specifies the action to take after a file has been processed. May be either `Delete` or `Move`.

TargetDirectory If `PostProcessOperation` is set to `Move`, specifies the absolute path of the directory to move files to.

A.2.2 FTP Listener

`com.pilotfish.eip.modules.FTPListener`

An FTP Listener acts identically to a directory listener, except that it polls a directory on a remote FTP server instead of the local filesystem.

Host Specifies the hostname or IP address of the FTP server to connect to.

Port Specifies the FTP server port to connect to. This is usually `21`, but this parameter must always be supplied.

UserName Specifies the username to connect to the FTP server with.

Password Specifies the plaintext password to connect to the FTP server with.

KeepConnection Either `True` or `False`: specifies whether to maintain the FTP connection between polling attempts, or re-establish it each time.

PollingInterval Specifies the number of seconds to wait between polling attempts.

PollingDirectory Specifies the absolute path to the directory on the remote system to poll for files.

FileNameRestriction Specifies a pattern to match file names against. Only files that match that match will be processed.

FileExtensionRestriction Specifies a space-separated list of extensions to match. Only files with extensions in the list will be processed.

PostProcessOperation Specifies action to take after a file has been processed. May be either `Delete` or `Move`.

TargetDirectory If `PostProcessOperation` is set to `Move`, specifies the absolute path of the directory on the remote system to move files to.

A.2.3 SFTP Listener

`com.pilotfish.eip.modules.SFTPListener`

An SFTP Listener acts identically to a directory listener, except that it polls a directory on a remote SFTP server instead of the local filesystem.

Host Specifies the hostname or IP address of the SFTP server to connect to.

Port Specifies the SFTP server port to connect to. This is usually `22`, but this parameter must always be supplied.

KeyAuthentication Either `True` or `False`: specifies whether to use key- or password-based authentication. If `True`, the key must be present in the Route's definition directory under the filename `/sftp_key.dat`.

UserName Specifies the username to connect to the SFTP server with.

Password If `KeyAuthentication` is `False`, specifies the plaintext password to connect to the SFTP server with.

KeepConnection Either `True` or `False`: specifies whether to maintain the SFTP connection between polling attempts, or re-establish it each time.

PollingInterval Specifies the number of seconds to wait between polling attempts.

PollingDirectory Specifies the absolute path to the directory on the remote system to poll for files.

FileNameRestriction Specifies a pattern to match file names against. Only files that match will be processed.

FileExtensionRestriction Specifies a space-separated list of extensions to match. Only files with extensions in the list will be processed.

PostProcessOperation Specifies action to take after a file has been processed. May be either `Delete` or `Move`.

TargetDirectory If `PostProcessOperation` is set to `Move`, specifies the absolute path of the directory on the remote system to move files to.

A.2.4 POP3 Listener

`com.pilotfish.eip.modules.EmailListener`

A POP3 / E-Mail Listener polls a POP3 mailbox on a remote server for messages. The content of the message or the content of any attachments is used as transaction data. Processed messages are deleted.

PollingInterval Specifies the number of seconds to wait between polling attempts.

Host Specifies the hostname or IP address of the POP3 server to connect to.

Port Specifies the POP3 server port to connect to. This is usually `110`, but this parameter must always be supplied.

UserName Specifies the POP3 username to connect with.

Password Specifies the POP3 password to connect with.

SubjectRestrictions Specifies a pattern to filter email subjects by. Only emails with subjects that match the pattern will be processed.

SenderRestrictions Specifies a pattern to filter email senders by. Only emails with senders that match the pattern will be processed.

DataInAttachment Either `True` or `False`: specifies whether to pull transaction data from the body of the email, or its attachments.

A.2.5 Database Table Listener

`com.pilotfish.eip.modules.DatabaseTableListener`

Database Table Listeners periodically poll database tables for transaction data. Every row returned is treated as a separate data unit, although results can optionally be lumped together into a single transaction. The name of the XML document's root element is `DataStream`. Under `DataStream` are elements with names corresponding to the table being polled. Under those are elements corresponding to the data in each row, with tag names generated from column names.

Note that the Database Table Listener is not capable of retrieving data from more than one table.

To take an example, let's assume there is a table named `Names`, with two fields, `First` and `Last`. There are two rows in the table, for John Smithson and Smith Johnson. The Database Table Listener would produce the following output:

```
<DataStream>
    <Names>
        <First>John</First>
        <Last>Smithson</Last>
    </Names>
    <Names>
        <First>Smith</First>
        <Last>Johnson</Last>
    </Names>
</DataStream>
```

Null fields are represented with empty tags. Most SQL datatypes are translated seamlessly to their XML equivalents.

PollingInterval Specifies the number of seconds to wait between polling attempts.

JdbcDriver Specifies the fully-qualified class name of the JDBC driver to use when connecting to the database.

JdbcUrl Specifies the JDBC URL to use when connecting to the database.

UserName Specifies the username to use when connecting to the database.

Password Specifies the password to use when connecting to the database.

TableName Specifies the database table to poll.

UseKeyField Either `True` or `False`: specifies the name of a field in the table that is guaranteed to be unique. Specifying a key field makes the process of deleting rows more efficient.

KeyField If `UseKeyField` is `True`, specifies the name of the unique field.

KeepConnection Specifies whether to keep the database connection alive between polling attempts, or close and re-establish it on each attempt.

SeparateTransaction Either `True` or `False`: specifies whether to create a separate transaction for every row found in the database, or combine all results into a single transaction.

A.2.6 JMS Listener

`com.pilotfish.eip.modules.JMSListener`

JMS Listeners are capable of receiving data through the Java Messasing Service. They support both the `point-to-point` and `publish-subscribe` messaging domains.

Note that a JMS Listener, in most cases, must be running in a JMS-enabled environment in order to be useful.

NamingContextFactoryClass Specifies the fully-qualified class name of the initial context factory to use.

ProviderURL Specifies the URL of the lookup service to use.

UserName Specifies the username to use when performing context lookups.

Password Specifies the password to use when performing context lookups.

PointToPointMode Either `True` or `False`: specifies whether to operate in point-to-point (queue-based) or publish-subscribe (topic-based) mode.

QueueName If `PointToPointMode` is set to `True`, specifies the name of the JMS queue to attach to.

TopicName If `PointToPointMode` is set to `False`, specifies the name of the JMS topic to attach to.

QueueConnectionFactoryName If `PointToPointMode` is set to `True`, specifies the name of the QueueConnectionFactory to lookup and use.

TopicConnectionFactoryName If `PointToPointMode` is set to `False`, specifies the name of the TopicConnectionFactory to lookup and use.

A.2.7 HTTP POST Listener

`com.pilotfish.eip.modules.HttpPostListener`

HTTP POST Listeners receive transaction data from the body of an HTTP POST request.

RequestPath Specifies the path to listen for responses at. The complete URL of this listener will be the URL of the EIP installation itself, plus `/receipt/`, plus this value. So, if `RequestPath` is set to `requestTest`, and the EIP is installed at `http://eip.company.com/eip/`, the full path to this `HttpPostListener` would be `http://eip.company.com/eip/receipt/requestTest`.

A.2.8 SOAP Web Service Listener

`com.pilotfish.eip.modules.SoapHttpPostListener`

SOAP Listeners receive transaction data from the body of a SOAP-compliant HTTP POST request.

RequestPath Specifies the path to listen for responses at. The complete URL of this servlet will be the URL of the EIP installation itself, plus `/receipt/`, plus this value. So, if `RequestPath` is set to `soapTest`, and the EIP is installed at `http://eip.company.com/eip/`, the full path to this `SoapHttpPostListener` would be `http://eip.company.com/eip/receipt/soapTest`

A.2.9 Programmatically-Invokable Listener

`com.pilotfish.eip.modules.TriggerableListener`

This Listener type may be invoked from within a Java class, by calling the static method `TriggerableListene` `trigger(StringlistenerName, InputStreamdata)`. It requires no configuration. The `listenerName` parameter is the name of the Listener, from its definition element.

A.3 Transports

A.3.1 Directory Transport

`com.pilotfish.eip.modules.DirectoryTransport`

Directory Transports deliver transaction data by creating files in a directory on the local system.

TargetDirectory Specifies the absolute path of the directory to create files in. This directory must already exist.

FileName Specifies the name to create files with. A unique identifier will be appended to the file to prevent duplication.

FileExtension Specifies the extension to create files with.

A.3.2 FTP Transport

`com.pilotfish.eip.modules.FTPTransport`

An FTP Transport behaves identically to a Directory Transport, except that it delivers files to a directory on a remote FTP server.

Host Specifies the hostname or IP address of the FTP server to connect to.

Port Specifies the FTP server port to connect to. This is usually `21`, but this parameter must always be supplied.

UserName Specifies the username to connect to the FTP server with.

Password Specifies the plaintext password to connect to the FTP server with.

KeepConnection Either `True` or `False`: specifies whether to maintain the FTP connection between delivery attempts, or re-establish it each time.

TargetDirectory Specifies the absolute path of the directory on the remote system to create files in. This directory must already exist.

FileName Specifies the name to create files with. A unique identifier will be appended to the file to prevent duplication.

FileExtension Specifies the extension to create files with.

A.3.3 SFTP Transport

`com.pilotfish.eip.modules.SFTPTransport`

An SFTP Transport behaves identically to a Directory Transport, except that it delivers files to a directory on a remote SFTP (file transfer over SSH [secure shell]) server.

Host Specifies the hostname or IP address of the SFTP server to connect to.

Port Specifies the SFTP server port to connect to. This is usually `22`, but this parameter must always be supplied.

KeyAuthentication Either `True` or `False`: specifies whether to use key- or password-based authentication. If `True`, the key must be present in the Route's definition directory under the filename `/sftp_key.dat`.

UserName Specifies the username to connect to the SFTP server with.

Password If `KeyAuthentication` is `False`, specifies the plaintext password to connect to the SFTP server with.

KeepConnection Either `True` or `False`: specifies whether to maintain the SFTP connection between delivery attempts, or re-establish it each time.

FileName Specifies the name to create files with. A unique identifier will be appended to the file to prevent duplication.

FileExtension Specifies the extension to create files with.

A.3.4 SMTP Transport

`com.pilotfish.eip.modules.EmailTransport`

SMTP Transports deliver transactions via email.

Host The hostname or IP address of the SMTP server to connect to.

Port The port on the SMTP server to connect to. The standard is 25.

SendFrom The text to place in the email's sender field.

ReplyTo The text to place in the email's reply-to address.

SendTo The email address(es) to send to. Separate addresses with spaces.

Subject The text to place in the email's subject field.

DataInAttachment Either `True` or `False`: specifies whether to include the transaction data in the message body, or as an attachment.

A.3.5 Database Table Transport

`com.pilotfish.eip.modules.DatabaseTableTransport`

Database Table Transports deliver data in XML format to a database table. In all respects, they operate as the reverse of Database Table Listeners. The DTT's input format is the same as the DTL's output format.

By default, Database Table Transports create a single new row in the database for every transaction unit. However, it is also possible to configure them to update existing rows by specifying a "key field". If a key field is specified, the DTT will attempt to perform an update on any rows matching that key. If no rows match, a new row will be isnerted.

JdbcDriver Specifies the fully-qualified class name of the JDBC driver to use when connecting to the database.

JdbcUrl Specifies the JDBC URL to use when connecting to the database.

UserName Specifies the username to use when connecting to the database.

Password Specifies the password to use when connecting to the database.

TableName Specifies the database table to poll.

UseKeyField Either `True` or `False`: specifies whether to attempt to update existing data in the database using a key field, or always insert new rows.

KeyField If `UseKeyField` is `True`, specifies the field to use for key matching.

A.3.6 JMS Transport

`com.pilotfish.eip.modules.JMSTransport`

JMS Tranports are capable of sending data via the Java Messaging Service, and by extension all technologies that it integrates with. Like the JMS Listener, it supports both the `point-to-point` and `publish-subscribe` messaging domains.

NamingContextFactoryClass Specifies the fully-qualified class name of the initial context factory to use.

ProviderURL Specifies the URL of the lookup service to use.

UserName Specifies the username to use when performing context lookups.

Password Specifies the password to use when performing context lookups.

PointToPointMode Either `True` or `False`: specifies whether to operate in point-to-point (queue-based) or publish-subscribe (topic-based) mode.

QueueName If `PointToPointMode` is set to `True`, specifies the name of the JMS queue to send data to .

TopicName If `PointToPointMode` is set to `False`, specifies the name of the JMS topic to send data to.

QueueConnectionFactoryName If `PointToPointMode` is set to `True`, specifies the name of the QueueConnectionFactory to lookup and use.

TopicConnectionFactoryName If `PointToPointMode` is set to `False`, specifies the name of the TopicConnectionFactory to lookup and us

A.3.7 HTTP POST Transport

`com.pilotfish.eip.modules.HttpPostTransport`

HTTP POST Transports send data via an HTTP POST request to a remote URL.

TargetURL The URL to send to.

UserName The username to connect with, if the server requires authentication.

Password The password to connect with, if the server requires authentication.

A.3.8 SOAP Web Service Transport

`com.pilotfish.eip.modules.SoapHttpPostTransport`

SOAP Web Service Transports send data via a SOAP-wrapped HTTP POST request to a remote URL.

TargetURL The URL to send to.

UserName The username to connect with, if the server requires authentication.

Password The password to connect with, if the server requires authentication.

A.4 Processors

A.4.1 ZIP/GZIP/BZIP2 Compression Processors

```
com.pilotfish.eip.modules.ZipProcessor
com.pilotfish.eip.modules.UnzipProcessor

com.pilotfish.eip.modules.GZIPCompressionProcessor
com.pilotfish.eip.modules.GZIPDecompressionProcessor
```

```
com.pilotfish.eip.modules.BZIP2CompressionProcessor
com.pilotfish.eip.modules.BZIP2DecompressionProcessor
```

These processors are capable of reading and writing compressed data, using the ZIP, GZIP, and BZIP2 algorithm. ZIP is the fastest but least effective, while BZIP2 is the slowest but most effective. GZIP is in the middle.

None of these require any configuration.

A.4.2 Asymmetric Encryption Processor

```
com.pilotfish.eip.modules.AsymmetricEncryptionProcessor
```

Asymmetric Encryption Processors (ASP's) perform asymmetric encryption, encrypting data with the recipient's public key so that it can only be decrypted by their private key. ASP's are also capable of signing data with the sender's private key.

EncryptionAlgorithm The name of the encryption algorithm to use. Defaults to CAST5. The allowable names vary depending on how the Java Cryptography Extension is installed in a particular JVM.

PublicKeyFile The file containing the public key to encrypt with.

Signature Either `True` or `False`, indicating whether to sign the encrypted data with a private key.

PrivateKeyFile If `Signature` is `True`, the private key to sign with.

PrivateKeyPassword The private key's password, if `Signature` is `True` and the private key specified requires one.

A.4.3 Asymmetric Decryption Processor

```
com.pilotfish.eip.modules.AsymmetricDecryptionProcessor
```

Performs asymmetric decryption, using a private key. Capable of verifying signatures on incoming data using the sender's public key.

PrivateKeyFile The private key file to decrypt with.

PrivateKeyPassword The private key's password, if it requires one.

VerifySignature Either `True` or `False`: indicates whether to verify the signatures on decrypted data using the sender's public key.

PublicKeyFile If `VerifySignature` is `True`, the public key to verify signatures with.

A.4.4 Symmetric Encryption Processor

Symmetric Encryption Processors perform symmetric (shared-key) encryption, where the encryption and decryption keys are the same.

EncryptionAlgorithm The name of the encryption algorithm to use. Defaults to CAST5. The allowable names vary depending on how the Java Cryptography Extension is installed in a particular JVM.

Password The password (encryption key) to encrypt with.

A.4.5 Symmetric Decryption Processor

Symmetric Decryption Processors perform symmetric (shared-key) decryption.

Password The password (encryption key) to decrypt with.

A.4.6 EBCDIC I/O Processors

`com.pilotfish.eip.modules.EbcdicInputProcessor`
`com.pilotfish.eip.modules.EbcdicOutputProcessor`

These processors convert data from EBCDIC to ASCII. No configuration is required.

A.5 Transformation Modules

A.5.1 Java Objects

`com.pilotfish.eip.modules.JavaObjectTransformationModule`

This Transformation Module converts data between serialized JavaBeans (Java objects with accessor methods following a certain naming convention) and XML.

The XML format used is similar to that of the Database Table Listener and Transport. The root element is `DataStream`, and below that are elements corresponding to the class names of the objects found in the stream. Below each of those are elements corresponding to the JavaBeans' properties.

These Transformation Modules require no configuration.

A.5.2 Relay

`com.pilotfish.eip.modules.RelayTransformationModule`

This Transformation Module does nothing. It should be used when the source or target format is already some form of XML, and no conversion is required.

This module is used by default if no Transformation Module is specified in a Format's configuration.

This module requires no configuration.

A.5.3 Fixed-Width and Delimited File

`com.pilotfish.eip.modules.FlatFileTransformationModule`

The Flat File Transformation Module is capable of transforming fixed-width and delimited files into XML, and vice-versa. It can support files of arbitary complexity, as well as perform data and format conversions at the same time as the transformation.

It is configured through the use of a File Format Specification file, the layout of which is described in an appendix to this document.

FileSpec The File Format Specification file to use.

A.6 Forking and Joining Modules

A.6.1 Numeric Forking Module

`com.pilotfish.eip.modules.NumericForkModule`

The Numeric Forking Module splits transactions at the level of elements immediately below the XML root element. It takes a single parameter, indicating how many elements each forked subtransaction should include.

Each forked subtransaction uses the same XML root element as the original transaction.

ForkCount How many elements each forked subtransaction should include below the XML root element.

A.6.2 Numeric Joining Module

`com.pilotfish.eip.modules.NumericJoinModule`

The Numeric Joining Module combines transactions at the level of elements immediately below the XML root element. For it to work properly, all transactions it processes must use the same XML root element.

MaximumCombinedTransactions The maximum number of transactions to join together.

MaximumWaitSeconds The number of seconds to wait for MaximumCombinedTransactions to arrive, before simply sending whatever has accumulated.

A.6.3 Null Joining Module

`com.pilotfish.eip.modules.NullJoinModule`

This Joining Module does nothing. It is used by default if no Joining Module is specified in a Format's configuration.

This module requires no configuration.

A.6.4 Null Forking Module

`com.pilotfish.eip.modules.NullForkingModule`

This Forking Module does nothing. It is used by default if no Forking Module is specified in a Format's configuration.

This module requires no configuration.

A.7 Routing Modules

A.7.1 Relay Routing Module

`com.pilotfish.eip.modules.NullRoutingModule`

This Routing Module does nothing. It passes all transactions to all the target systems in Routes it is used in.

This module is used by default if no Routing Module is specified in a Format's configuration.

This module requires no configuration.

A.7.2 XPath Routing Module

`com.pilotfish.eip.modules.routing.XPathRoutingModule`

The XPath routing module evaluates a series of arbitarily complex XPath expressions against the transaction data once it is in standardized XML, in order to determine which target systems to send the transaction data to.

It is the only stock EIP module that does not use a simple property-value format for its `<ModuleConfig>` element. The format that is does use is described in an appendix to this document.

A.8 Transaction Monitors

A.8.1 Email Alert Transaction Monitor

`com.pilotfish.eip.modules.EmailAlertTransactionMonitor`

This Transaction Monitor sends an email alert to the specified address whenever a transaction error occurs.

Host The hostname or IP address of the SMTP server to connect to.

Port The port on the SMTP server to connect to. The standard is 25.

SendFrom The text to place in the email's sender field.

ReplyTo The text to place in the email's reply-to address.

SendTo The email address(es) to send to. Separate addresses with spaces.

Subject The text to place in the email's subject field.

A.8.2 SNMP Trap Transaction Monitor

`com.pilotfish.eip.modules.SNMPTransactionMonitor`

This Transaction Monitor sends an SNMP trap to the specified destination whenever a transaction error occurs.

DestinationIP The IP address of the trap destination.

DestinationPort The UDP port of the trap destination (usually 162).

CommunityName The trap SNMP community string.

A.8.3 Route Trigger Transaction Monitor

`com.pilotfish.eip.modules.RouteTriggerMonitor`

The Route Trigger Transaction Monitor initializes a new transaction by invoking the specified Triggerable Listener whenever an error occurs on the route it is associated with. The new transaction's data is an error report that takes the following format:

```
<errorReport xmlns="...[1]">
    <transactionID>...</transactionID>
    <errorStage>...</transactionID>
    <errorComponent>...</errorComponent>
    <exceptionClass>...</exceptionClass>
    <exceptionMessage>...</exceptionMessage>
    <exceptionTrace>...</exceptionTrace>
</errorReport>
```

`transactionID` is the ID of the transaction that failed with an error. `errorStage` is the stage that the transaction failed at: `listener`, `xslt`, `routing`, etc. `errorComponent` is present when a named component (like a Listener or Transport) generates an error, and it contains that component's name. `exceptionClass`, `exceptionMessage`, and `exceptionTrace` are only present when the error is the result of an unexpected exception being thrown, and contain the exception's class, error message, and stack trace, respectively.

ErrorListenerName The name of the Triggerable Listener to invoke when transaction errors occur.

[1] http://www.pilotfishtechnology.com/eip/RouteErrorReport

Appendix B

XPath Routing Module Configuration

B.1 Overview

The XPath Routing Module supplied with the EIP uses a unique configuration format. It consists of a `RuleSet` element, followed by a series of `Rule` elements. Each rule contains an expression to match against, and a list of the target systems that the transaction should be sent to if the match succeeds.

Expressions within rules can be chained using the `Or` and `And` elements, which represent boolean operations.

B.2 Element Reference

B.2.1 RuleSet Element

The `RuleSet` element is the root of the XPath Routing Module's configuration. It may contain any number of child `Rule` elements.

Name	`<RuleSet>`
Required	Yes
Maximum Occurrences	One
Children	Rule

Attributes

accumulateTargets Specified how rule matching should work. If `False`, processing will stop at the first rule matched, and the transaction will be sent only to the targets that rule specifies. If `True`, every rule will be processed, and the transaction will be sent to all targets specified by the rules that matched.

B.2.2 Rule Element

The Rule element defines a single routing rule, containing a condition and a list of targets. If the condition matches the transaction data, it will be sent to the targets listed.

Name	<Rule>
Required	Yes
Maximum Occurrences	Unlimited
Children	Condition, Targets

B.2.3 Condition Element

The Condition element is a wrapper around the condition expression(s) within a rule.

Name	<Condition>
Required	Yes
Maximum Occurrences	One
Children	Expression, Or, And

B.2.4 Expression Element

The Expression element defines an XPath expression to match against. The XPath expression will be evaluated as a boolean value.

Complex rules can be expressed either within a single expression, or by combining expressions using the `Or` and `And` elements.

The XPath expression should be the value of this element.

Name	<Expression>
Required	No
Maximum Occurrences	Unlimited

B.2.5 Or Element

The Or element combines any expression elements under it using the boolean OR operation.

Name	<Or>
Required	No
Maximum Occurrences	Unlimited
Children	Expression, Or, And

B.2.6 And Element

The And element combines any expression elements under it using the boolean AND operation.

Name	<And>
Required	No
Maximum Occurrences	Unlimited
Children	Expression, Or, And

B.2.7 Targets Element

The Targets element wraps the list of targets to send data to if the rule it is under matches a transaction.

Name	<Targets>
Required	Yes
Maximum Occurrences	One
Children	TransportTarget, ErrorTarget

B.2.8 TransportTarget Element

The TransportTarget element specifies that transaction data should be sent to the target in a route that contains a Transport with the specified name.

Name	<TransportTarget>
Required	No
Maximum Occurrences	Unlimited

Attributes

name The name of the Transport to send data to.

B.2.9 ErrorTarget Element

The ErrorTarget element specifies that an error should be raised if the rule it is in matches. The value of the element should be the error message to use.

Name	<ErrorTarget>
Required	No
Maximum Occurrences	Unlimited

Appendix C

Writing File Format Specifications

C.1 Overview

The EIP's Flat File Transformation Module relies on File Format Specification (FFS) definitions for configuration. Nearly every fixed-width or delimited file format can be read by creating an FFS for it. These documents are XML, and as such can be created and edited using any common text editor.

This section describes the general operational approach that the EIP's Flat File Transformation Module uses, as well as the format of a valid FFS definition file.

C.1.1 General Description

The core FFS format is quite simple. Besides some required initialization information, the majority of the file consists of a list of the fields that make up the format which needs to be read/written.

For delimited files, each field element specifies only a field name. This is used to identify the field's data when the Flat File Transformation Module produces an XML document. Fields in fixed-width files are named as well, but they also specify the starting and ending positions of the field within the record.

Modules called Converters and DataSources may also be associated with fields. Converters perform field-level data conversion, and DataSources are capable of retrieving data from outside the document being transformed (for instance, querying a database for information that the document lacks).

C.1.2 Module Definition vs. Usage

In FFS files, modules must be defined before they are used. Module definitions are placed under the `<ModuleRegistry>` tag at the top of the document. Defining a module creates an instance of it in memory, which is then shared by all the fields that access it. This means that modules performing common tasks (such as date/time transformation) can be defined and configured once, then reused throughout the file without having to repeat their configuration information.

Modules accept configuration parameters when they are defined as well as each time they are used.

C.1.3 Record Controls

One problem often encountered when dealing with flat files is determining what type of record is currently being read. The FFS handles this using a third type of module, referred to as a Record

Control.

Every record type in a FFS definition may have a Record Control associated with it. When the Flat File Transformation Module finishes processing a record, it passes the remaining bytes of data in the file to the associated Record Control, which is responsible for determining which type of record appears next.

Two Record Controls are supplied with the EIP distribution. `com.pilotfish.eip.transform.ConstantRecordControl` always returns the same record type, and `com.pilotfish.eip.transform.FieldRecordControl` determines the next record type using a conditional decision tree based on field values.

C.2 Element Reference

C.2.1 TransRoot Element

The TransRoot element is the document element in a FFS definition. It has no attributes.

Name	`<TransRoot>`
Required	Yes
Maximum Occurrences	One
Children	HandlerConfig, ModuleRegistry, Initialization, RecordControl, RecordSegment

C.2.2 HandlerConfig Element

The HandlerConfig element is a required element that does not affect any functionality. It needs to be included for compatibility reasons.

It has two required child elements, `<Builder>` and `<Serializer>`. These must be set to `com.pilotfish.eip.transform.flatfile.RecordDeserializer` and `com.pilotfish.eip.transform.flatfile.RecordSerializer`, respectively.

C.2.3 ModuleRegistry Element

The ModuleRegistry element is a wrapper containing module (Converter, DataSource, or Record Control) instance definitions.

The ModuleRegistry element must be present, even if no modules need to be defined.

Name	`<ModuleRegistry>`
Required	Yes
Maximum Occurrences	One
Children	RecordControl, Converter, DataSource

C.2.4 ModuleRegistry.RecordControl Element

The RecordControl element defines a named RecordControl instance that can be referenced from elsewhere in the file.

The children of the RecordControl element are passed to the new instance as configuration.

Name	`<RecordControl>`
Required	No
Maximum Occurrences	Unlimited
Children	Any

Attributes

name The name to assign the new RecordControl instance. This is how it will be referenced later in the file.

class The fully-qualified class name of the RecordControl class to instantiate.

C.2.5 ModuleRegistry.Converter Element

The Converter element defines a named Converter instance that can be referenced from elsewhere in the file.

The children of the Converter element are passed to the new instance as configuration.

Name	`<Converter>`
Required	No
Maximum Occurrences	Unlimited
Children	Any

Attributes

name The name to assign the new Converter instance. This is how it will be referenced later in the file.

class The fully-qualified class name of the Converter class to instantiate.

C.2.6 ModuleRegistry.DataSource Element

The DataSource element defines a named DataSource instance that can be referenced from elsewhere in the file.

The children of the DataSource element are passed to the new instance as configuration.

Name	`<DataSource>`
Required	No
Maximum Occurrences	Unlimited
Children	Any

Attributes

name The name to assign the new DataSource instance. This is how it will be referenced later in the file.

class The fully-qualified class name of the DataSource class to instantiate.

C.2.7 Initialization Element

The Initialization element is a wrapper containing global configuration information for the FFS definition.

Name	`<Initialization>`
Required	Yes
Maximum Occurrences	One
Children	Record, Field

C.2.8 Initialization.Record Element

The Record element defines the type of record in use in the file format that needs to be read. It can specify either fixed-width or delimited records, as well as a delimiter in the latter case.

Name	`<Record>`
Required	Yes
Maximum Occurrences	One

Attributes

type Either `fixed-width` or `delimited`.

delimiter Specifies the record delimiter in the case of a delimited file. Should be specified in the form of Java escape characters, i.e. `\n` for a newline.

C.2.9 Initialization.Field Element

The Field element defines the type of field in use in the file format that needs to be read. It can specify either fixed-width or delimited fields, as well as a delimiter in the latter case.

Note that this is a different element than that allowed under the `RecordSegment` element.

Name	`<Field>`
Required	Yes
Maximum Occurrences	One

Attributes

type Either `fixed-width` or `delimited`.

delimiter Specifies the field delimiter in the case of a delimited file. Should be specified in the form of Java escape characters, i.e. `\n` for a newline.

C.2.10 RecordSegment Element

The RecordSegment element defines a set of fields that make up a logical component of a data file. Simple data files will only have one RecordSegment, complex ones may have several.

At least one RecordSegment element is required in all FFS definitions.

Name	`<RecordSegment>`
Required	Yes
Maximum Occurrences	Unlimited
Children	Field, RecordControl

Attributes

name The name of the RecordSegment, and thus the name of the XML element that should represent it in the transformed document.

C.2.11 Field Element

The Field element represents a single field within a record segment. It may have a Converter or DataSource module attached to it.

Field elements in FFS definitions decsribing delimited files must contain two child elements, named `Start` and `End`. These contain the starting and ending positions of the field *relative to the beginning of their record segment*. For example:

```
<Field name="FirstName">
    <Start>1</Start>
    <End>20</End>
</Field>
```

Name	`<Field>`
Required	Yes
Maximum Occurrences	Unlimited
Children	Start, End, CustomData, Format

C.2.12 CustomData Element

CustomData elements are wrappers around DataSource references. DataSources are not referenced by fields directly for reasons of forwards-compatibility.

Name	`<CustomData>`
Required	No
Maximum Occurrences	One
Children	DataSource

C.2.13 Format Element

Format elements are wrappers around Converter references. Converters are not referenced by fields directly for reasons of forwards-compatibility.

Name	<Format>
Required	No
Maximum Occurrences	One
Children	Converter

C.2.14 Converter Element

<Converter> elements reference an instance of a Converter, previously defined under the ModuleRegistry element. This means that the specified Converter will be used to supply the data for the field this element is included under. Any child elements of the <Converter> element are passed to the Converter instance as invocation parameters.

Name	<Converter>
Required	Yes
Maximum Occurrences	One
Children	Any

Attributes

name The name of the Converter instance to reference.

C.2.15 DataSource Element

<DataSource> elements reference an instance of a DataSource, previously defined under the ModuleRegistry element. This means that the specified DataSource will be used to supply the data for the field this element is included under. Any child elements of the <DataSource> element are passed to the DataSource instance as invocation parameters.

Name	<DataSource>
Required	Yes
Maximum Occurrences	One
Children	Any

Attributes

name The name of the DataSource instance to reference.

C.2.16 RecordControl Element

<RecordControl> elements reference an instance of a RecordControl, previously defined under the ModuleRegistry element. This means that the specified RecordControl will be used to determine the type of the next record in the file, when the record they are associated with has been processed. Any child elements of the <RecordControl> element are passed to the RecordControl instance as invocation parameters.

Name	`<RecordControl>`
Required	Yes
Maximum Occurrences	One
Children	Any

Attributes

name The name of the RecordControl instance to reference.

Appendix D

Object Definition File Formats

D.1 Route Definition File Format

Route definition files are XML. This section is intended as a reference describing all the XML tags and attributes that they may contain. Example Route definitions are provided with the EIP distribution to illustrate how these all fit together.

D.1.1 Route Element

The Route element is the root element in a Route definition.

Name	`<Route>`
Required	Yes
Maximum Occurrences	One
Children	RoutingMonitor, TransactionMonitors, Source, Target

Attributes

name Required string attribute defining the name of the Route. This must match the name of the directory the definition file is in.

logTransactionContent Optional boolean attribute specifying whether transactions passing through this Route should be logged in the transactional database.

D.1.2 RoutingModule Element

The RoutingModule element is optional - it defines the Routing Module that the Route uses to determine which targets a transaction should be passed to.

If this element is not present, an instance of `NullRoutingModule` is used.

Name	`<RoutingModule>`
Required	No
Maximum Occurrences	One
Children	ModuleConfig

Attributes

class Specifies the name of the module's Java class. This class must implement the com.pilotfish.eip.RoutingModu interface, and exist in the EIP distribution or the Route's /lib directory.

D.1.3 TransactionMonitors Element

The TransactionMonitors element wraps a list of TransactionMonitor elements. The order in which the Transaction Monitors are defined is the order in which they will be notified of events, with the first monitor listed being notified first.

Name	`<TransactionMonitors>`
Required	No
Maximum Occurrences	One
Children	TransactionMonitors

D.1.4 TransactionMonitor Element

TransactionMonitor elements define Transaction Monitors to be attached to the Route, and notified of transaction events as they occur.

Name	`<TransactionMonitor>`
Required	Yes
Maximum Occurrences	Unlimited
Children	ModuleConfig

Attributes

name Required string attribute specifying the name of the module. This must be unique within all other names in the Route.

class Required string attribute specifying the name of the module's class.

D.1.5 Source Element

A Source element defines a source in the Route. Within every source is a Listener and Format reference, as well as an optional list of Processors.

Name	`<Source>`
Required	Yes
Maximum Occurrences	Unlimited
Children	Listener, Processors, FormatProfile

D.1.6 Listener Element

The Listener element defines a Listener instance within a source.

Name	<Listener>
Required	Yes
Maximum Occurrences	One
Children	ModuleConfig

Attributes

name Required string attribute specifying the name of the module. This must be unique within all other names in the Route.

class Required string attribute specifying the name of the module's class.

D.1.7 Target Element

A Target element defines a target in the Route. Within every target is a Format and Transport reference, as well as an optional list of Processors.

Name	<Target>
Required	Yes
Maximum Occurrences	Unlimited
Children	Transport, Processors, FormatProfile

D.1.8 Transport Element

The Transport element defines a Transport instance within a source.

Name	<Transport>
Required	Yes
Maximum Occurrences	One
Children	ModuleConfig

Attributes

name Required string attribute specifying the name of the module. This must be unique within all other names in the Route.

class Required string attribute specifying the name of the module's class.

D.1.9 FormatProfile Element

The FormatProfile element specifies the Format that a source or target should use to perform transformation.

Name	<FormatProfile>
Required	Yes
Maximum Occurrences	One
Children	None

Attributes

name Required string attribute specifying the name of the Format to use.

D.1.10 Processors Element

The Processors element is wrapper for a list of Processor elements. The order in which Processor elements are listed specifies the order in which they will be invoked, with the first Processor listed being invoked first.

Name	`<Processors>`
Required	No
Maximum Occurrences	One
Children	Processor

D.1.11 Processor Element

Processor elements define Processor instances attached to a source or target in a Route.

Name	`<Processor>`
Required	Yes
Maximum Occurrences	Unlimited
Children	ModuleConfig

Attributes

name Required string attribute specifying the name of the module. This must be unique within all other names in the Route.

class Required string attribute specifying the name of the module's class.

D.1.12 ModuleConfig Element

ModuleConfig elements are allowed under all Listener, Transport, Processor, Transaction Monitor, and Routing Module definitions. They contain the module's configuration information, and their format is defined by the specific module being defined. Some modules do not require any configuration at all, in which case a ModuleConfig element does not need to be supplied with their definition.

Name	`<ModuleConfig>`
Required	No
Maximum Occurrences	One
Children	Any

D.2 Format Definition File Format

Format definition files are XML. This section is intended as a reference describing all the XML tags and attributes that they may contain. Example Format definitions are provided with the EIP distribution to illustrate how these all fit together.

D.2.1 Format Element

The Format element is the root element in a Format definition.

Name	`<Format>`
Required	Yes
Maximum Occurrences	One
Children	TransformationModule, XSLT

Attributes

name Required string attribute defining the name of the Format. This must match the name of the directory the definition file is in.

D.2.2 TransformationModule Element

The TransformationModule element defines an instance of a Transformation Module within a Format.

If this element is not present, an instance of `RelayTransformationModule` is used.

Name	`<TransformationModule>`
Required	No
Maximum Occurrences	One
Children	ModuleConfig

Attributes

class Required string attribute specifying the name of the module's class.

D.2.3 XSLT Element

The XSLT element contains two sub-elements specifying the XSLT stylesheets that the Format should use for transformation.

Name	`<XSLT>`
Required	Yes
Maximum Occurrences	One
Children	ToXML, FromXML

D.2.4 ToXML Element

The value of the ToXML element specifies the XSLT stylesheet to use for source transformations: i.e. transforming the output of the specified Transformation Module into the standardized XML format deployed on the EIP.

The stylesheet needs to be located in the Format's definition directory, and the path should be specified relative to the directory's root. For instance, `/mibRequestTransform.xml`.

This tag is required. To indicate that no transformation should be performed, include it as an empty tag (`<ToXML/>`).

Name	<ToXML>
Required	Yes
Maximum Occurrences	One
Children	Value: name/path of the XSLT stylesheet to use.

D.2.5 FromXML Element

The value of the FromXML element specifies the XSLT stylesheet to use for target transformations: i.e. transforming standards-compliant XML into the format expected by the specified Transformation Module.

The stylesheet needs to be located in the Format's definition directory, and the path should be specified relative to the directory's root. For instance, `/mibRequestTransform.xml`.

This tag is required. To indicate that no transformation should be performed, include it as an empty tag (`<FromXML/>`).

Name	<FromXML>
Required	Yes
Maximum Occurrences	One
Children	Value: name/path of the XSLT stylesheet to use.

D.2.6 SplitControl Element

The SplitControl element contains two sub-elements specifying the forking and joining modules that this Format uses.

Name	<SplitControl>
Required	No
Maximum Occurrences	One
Children	ForkModule, JoinModule

D.2.7 ForkModule Element

The ForkModule element defines an instance of a Forking Module within a Format. The Forking Module will only be used in source transformations.

If this element is not present, an instance of `NullForkModule` is used.

Name	<ForkModule>
Required	No
Maximum Occurrences	One
Children	ModuleConfig

D.2.8 JoinModule Element

The JoinModule element defines an instance of a Joining Module within a Format. The Joining Module will only be used in target transformations.

If this element is not present, an instance of `NullJoinModule` is used.

Name	`<JoinModule>`
Required	No
Maximum Occurrences	One
Children	ModuleConfig