FMS

Flame Message Server

System Manual

FMS Flame Message Server System Manual

Copyright © 2006 - 2024 Flame Computing Enterprises cc All Rights Reserved

$RCSfile: FMS-Manual.sgml,v $ $Revision: 1.101.2.7.2.2.4.176 $ $Date: 2024/06/08 10:03:07 $

Table of Contents

Introduction

The Flame Message Solutions (collectively known as FMS) provides a multi-protocol secure business message server and utilities providing the middleware for communicating business messages over industry standard protocols conforming to the Electronic Business using eXtensible Markup Language (ebXML) Messaging Services Version 3.0, AS4 profile of ebMS 3.0 Version 1.0, the RosettaNet® Implementation Framework (RNIF) Version 2.00.01, and the Extensible Provisioning Protocol (EPP). Also included are the FMS AS4 light client, the server light client and the FMS management console.

FMS provides the critical middleware in the implementation of Service Oriented Architectures (SOA) between heterogeneous business processes.

This manual is specifically targeted at system administrators and integrators and provides the guidelines for installing, configuring and maintaining FMS products.

Detailed guidelines on integrating FMS with business application processes are also provided.

Chapter 1. Architecture

System Architecture

The typical B2B System Architecture is as follows:

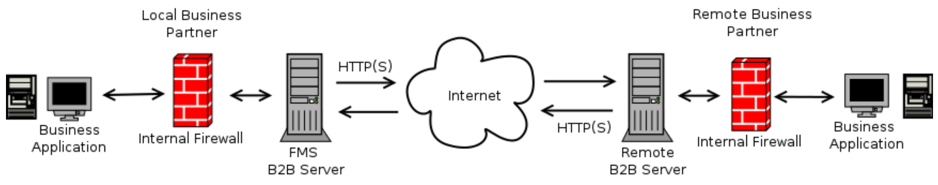


Figure 1-1. Generic Architecture

The various components in the above diagram include the following

- Local Business Application - produces and consumes B2B messages. These messages are either sent or received from a compliant B2B message handler.

  Local FMS Message Handler - provides the messaging service for communicating B2B messages over a secure connection to a remote message handler. It is the responsibility of the business application to generate XML business messages with optional associated attachments complying to relevant business agreements and the responsibility of the Message Handler to wrap these messages in an envelope conforming to the B2B protocol. The Message Handler also secures these messages using any required authentication tokens, signatures and encryption services.
- Local and Remote Firewalls - Optional firewalls which must be configured to allow the local message handler to communicate with a remote messaging handler over the required TCP/IP ports.
- Remote Message Handler - Receives and sends B2B messages. Received messages are verified according to the necessary security requirements and then handed to the remote business application. The Remote Message Handler may optionally respond back to the local Message Handler with a receipt or error signal indicating successful or unsuccessful processing of a message.

Product Architecture

FMS AS4 Light Client

The FMS AS4 Light Client supports push and pull message exchange patterns for secure B2B messaging with a remote partner. It is typically used in low volume B2B applications where a permanent connection to the Internet may not be available. The FMS AS4 Light Client provides a simple command line interface for ease of interfacing to any existing business application. It conforms fully to the OASIS Open Group AS4 Profile of ebMS V3 and may be used for secure signed and encrypted pushing (sending) and pulling (receiving) of business messages in a non-realtime environment.

The FMS Light Client push and pull scenarios are illustrated in the following diagrams.

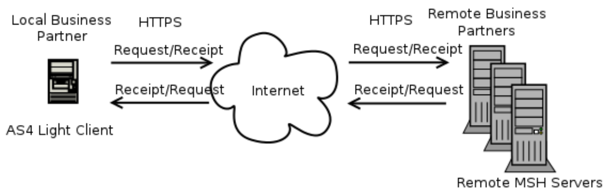


Figure 1-2. AS4 Push Scenario

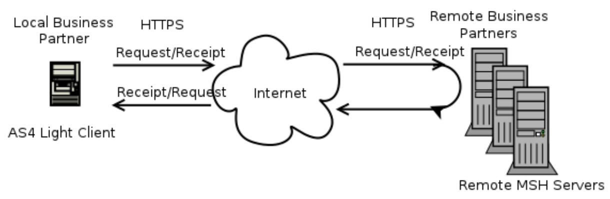


Figure 1-3. AS4 Pull Scenario

FMS Starter Edition

The FMS Starter Server supports a single local partner to single remote partner B2B middleware server application. It is typically used in low to medium volume real-time B2B messaging gateway applications where a permanent connection to the Internet is required. The FMS Starter Server provides a powerful trigger and file based interface making it easy to interface to any existing business application. It conforms fully to the OASIS Open Group AS4 Profile of ebMS V3 and may be used for secure signed and encrypted pushing (sending) and pulling (receiving) of business messages in a realtime B2B environment.

The starter solution licence may be extended to multiple trading partners.

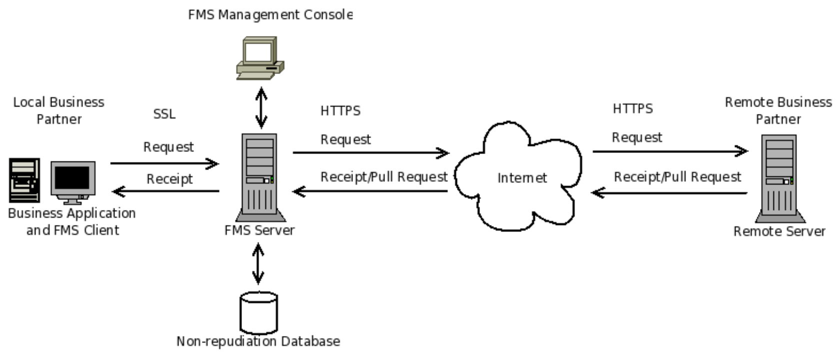


Figure 1-4. FMS Starter Edition Architecture

FMS Professional Edition

The FMS Professional Server supports a high availability single local partner to multiple remote partners B2B middleware server application. It is typically used in medium volume real-time B2B applications where a permanent connection to the Internet is required. The FMS Professional Server provides a powerful trigger and file-based interface making it easy to interface to any existing business application. It conforms fully to the OASIS Open Group AS4 Profile of ebMS V3 and may be used for secure signed and encrypted pushing (sending) and pulling (receiving) of business messages in a realtime B2B environment.

The professional solution provides the ability to define multiple business partner connections with different business process documents such as invoices and statements thereby enabling use in different business disciplines from a local business partner to multiple remote business partners.

The professional solution comes with an optional developers application programmers interface (API) for the client interface. This option provides the ability to integrate business applications for pushing and pulling messages to a remote MSH.

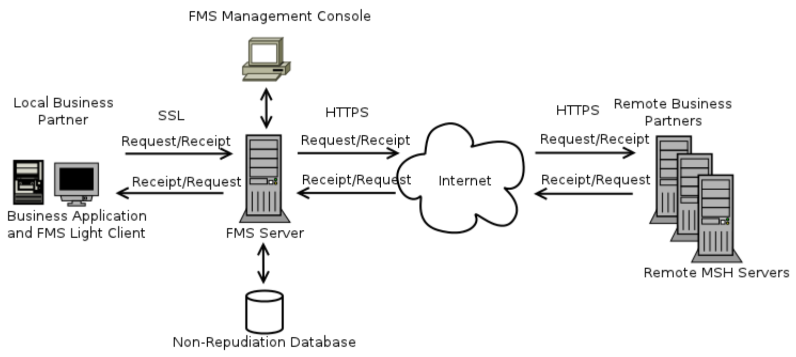


Figure 1-5. FMS Professional Edition Architecture

FMS Enterprise Edition

The FMS Enterprises Server supports a hub based multiple partners to multiple partners B2B middleware high availability server application. It is typically used in medium to high volume B2B applications where a permanent connection to the Internet is required. The FMS Enterprises Server provides a powerful trigger and file-based interface making it easy to interface to any existing business application. It conforms fully to the OASIS Open Group AS4 Profile of ebMS V3 and may be used for secure signed and encrypted pushing (sending) and pulling (receiving) of business messages in a realtime B2B environment.

The enterprise solution includes the tools to interactively monitor business transactions between multiple business partners on multiple virtual hosts, interactively managing messaging server resources such as connections, and allowing for store-and-forward functionality.

The enterprise solution comes with an optional developers application programmers interface (API) allowing customisation of the messaging server connections. This provides a flexible solution for extending the messaging protocols to incorporate alternate protocols.

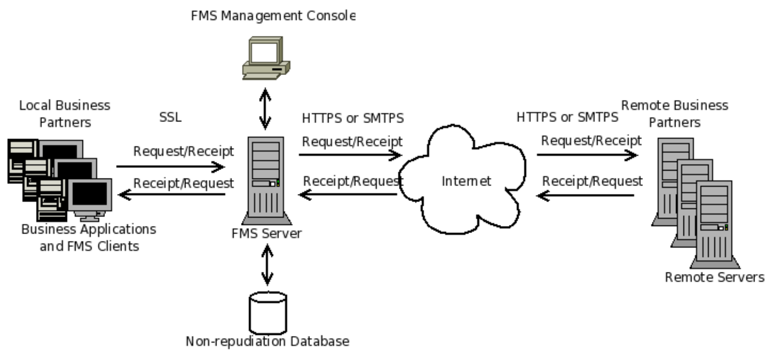


Figure 1-6. FMS Enterprise Edition Architecture

FMS AS4 Specifications

- *Drummond* Certified - 2013, 2014.
- *e-SENS* CEF Certified including SML/SMP Lookup Compliant - 2016.
- *ENTSOG* European Gas Industry - CEF Certified 2018.
- *SuperStream* Certified - 2012, 2014.
- *One Way Push*.
- *Two Way Asynchronous Push* Server only.
- *One Way Pull*.

- *Secure Transport* including SSL, SSLv3, TLS v.1, TLS v1.1, TLS v1.2 and TLS v1.3.
- *WSS v1.1 X509* signature support including support for SHA 256 algorithms.
- *WSS v1.1 X509* encryption support including support for the following AES GCM key and data encryption algorithms
  - `http://www.w3.org/2009/xmlenc11#rsa-oaep`
  - `http://www.w3.org/2009/xmlenc11#aes128-gcm`
  - `http://www.w3.org/2009/xmlenc11#aes256-gcm`

  The above algorithms are only supported with java 1.8 and later.
- *Non Repudiation Receipts*.
- *Synchronous Error Signals*.
- *SWA v1.1* Attachments (compressed, signed and encrypted attachments).
- *Ping* test service - server only.

Chapter 2. Installation

Overview

The following sections discuss the requirements for a succesful installation of FMS. Further in depth detail on the requirements are discussed in the following chapters.

System Requirements

The FMS Server requires setup by a suitably qualified system administrator who has a good understanding of system security and networking. The server requires access to various system resources including the following

- Access to TCP/IP ports 80, 443, 5432, 29360, 29450. Port 29360 is used for listening for management console connections typically coming from behind the company firewall. Port 80 and 443 access is required for receiving remote HTTP and HTTPS incoming connections. Port 5432 is typically used for connecting to the PostgreSQL database server if required.

  Any of the TCP/IP ports required by FMS may be configured to different values in the server interface configuration.
- The FMS server must be installed on a machine appropriately configured for secure remote access with visibility from both the internal network and outside network for the responses. This may require configuration on the firewall.
- FMS may be vulnerable to common TCP denial-of-service attacks including TCP SYN flooding. Server installations should include steps to minimize the impact of a denial-of-service attack using combinations of easily implemented solutions, such as deployment of firewall technology and border router filters to restrict inbound server access to known, trusted clients.

Supported Operating Systems

FMS runs on various UNIX, Linux and Windows operating systems but can also be installed on any operating system that supports the Java run time environment (JRE 1.8), including (but not limited to) the following:

- CentOS® 6, 7 and above
- Mac OS X® 10.8 and upwards
- Microsoft Windows® Server 2003 and above
- Microsoft Windows® 7 and above
- RedHat® Enterprise Linux 5, 6 and above
- Sun Solaris®
- Ubuntu® 14.04 and above

Java Runtime Environment (JRE) Requirements

FMS requires JRE version 1.8 or later. This is included in the Windows distribution by default and is also available at https://java.com/en/download/.

Before proceeding verify that Java is in the executable path as follows

```
java -version
```

Should the version of Java not be at 1.8 or greater then either contact the system administrator to load the correct version or ensure that the `PATH` environment variable is updated to point to the correct version.

Certain modern encryption algorithms are only supported by Java 1.8 and later. These include the following

- `http://www.w3.org/2009/xmlenc11#rsa-oaep`

- `http://www.w3.org/2009/xmlenc11#aes128-gcm`
- `http://www.w3.org/2009/xmlenc11#aes256-gcm`

Licence Requirements

The FMS licence file (`fms.lcn`) must be obtained from the provider and placed in the FMS installation directory prior to operation as per the Section called *Licence Configuration* in the post installation section.

Certificate Requirements

Secure Certificates may either be self generated or obtained from a commercial certificate authority (CA) such as Verisign or Thawte. See the Section called *Creating a Keystore using the Key Generator* in the Windows installation section and the Section called *Keystore Configuration* in the post installation section for further details.

Database Server

FMS uses a JDBC compliant database server for, amongst others, non-repudiation storage, message reporting, monitoring and logging, and for caching Collaboration Protocol Agreement (CPA) details.

Use of the database is recommended for all professional and enterprise installations.

Should FMS be configured with no database then AS4 pull requests are not supported. Message reporting and realtime display of message statistics will also not be available and the CPA details will only be cached in the running instance and will be lost on server re-start.

The database may be either installed on the same machine or on a dedicated server on the local area network. Version 12 of PostgreSQL or later is recommended. and may be obtained either as a standard install package with Linux operating systems or from http://www.postgresql.org.

Other JDBC compliant databases are currently not supported. Please contact the suppliers should this be a requirement.

FMS Installation

The FMS distributions are available in various package formats for different operating systems as per the following

Linux Distributions

The Linux distributions consist of separate RPM or debian packages for the server, management console, client and documentation.

RPM and Debian Package Distribution Server Installation

The following steps need to be taken to install the FMS server:

1. Ensure that PostgreSQL version 12 or later, and JRE version 1.8 or later are installed
2. As the `root` user execute the `rpm` install command as follows for RPM based package systems such as Redhat, Fedora or CentOS systems

   ```
   [root@localhost ~]# rpm -ivh fms-<VERSION>.noarch.rpm
   ```

   For Debian based package systems such as Ubuntu execute the install command as follows

   ```
   [root@localhost ~]# dpkg --install fms-<VERSION>.deb
   ```

   If upgrading RPM systems execute the rpm upgrade command as follows

   ```
   [root@localhost ~]# rpm -Fvh fms-<VERSION>.noarch.rpm
   ```

   For Debian systems execute the upgrade (same as install) command as follows

```
[root@localhost ~]# dpkg --install fms-<VERSION>.deb
```

3. Initialise the server as the root user by executing the following command:

   ```
   [root@localhost ~]# service fms init
   ```

   An initialisation may also be necessary after an upgrade, however, previous initialised configuration files will be extended with any new facilities. The server will terminate after initialisation.

4. Start the FMS server as the root user by executing the following command:

   ```
   [root@localhost ~]# service fms start
   ```

   Default configuration files will automatically be created should these not exist the first time the server is started.

5. To stop the FMS server as the root user execute the following command:

   ```
   [root@localhost ~]# service fms stop
   ```

6. To restart the server as the root user execute the following command:

   ```
   [root@localhost ~]# service fms restart
   ```

7. To check the status of the server as the root user execute the following command:

   ```
   [root@localhost ~]# service fms status
   ```

   If the server is not running the following message will be displayed:

   ```
   Flame Message Server :  [ DOWN ]
   ```

   If the server is running the following message will be displayed:

   ```
   Flame Message Server :  [ RUNNING ]
   ```

8. The FMS files are typically installed to the `/home/fms`, `/etc/fms` and `/usr/share/java/fms` directories.

   An `fms` user is created, if it does not exist already, with home directory set to `/home/fms`.

   The server is installed as a system service and will automatically be invoked on system restart. It is important to take note of the following essential points before starting the system.

   - FMS server output is logged to `/var/log/fms/fms.out` and errors are logged to `/var/log/fms/error.log` by default.

     FMS webservices security related detail is logged to `/var/log/fms/fms-webservices.log`.

     Additional FMS related logging detail may be logged to separate log file in the `/var/log/fms` directory.

     Logging output is controlled using the `/etc/fms/log4j.properties.as4` and `/etc/fms/logging.properties` configuration files. Instructions on how to configure FMS logging output using the system logger are detailed in the Section called *Syslog Configuration* in Chapter 3. Configuration changes will only take effect after restarting the FMS server.

   - The product MUST be licenced prior to first invocation.

     The licence MUST be obtained from the supplier and installed into the FMS home directory. Refer to the Section called *Licence Configuration* for further details.

   - Next the necessary server private and public keys MUST be obtained, or created if self signed, and installed in the FMS server keystore.

     The keystore MUST be created to store the private and public keys (certificates) used for secure communication between FMS and remote servers.

     Creating a keystore is not necessary when performing an upgrade as these may already exist from the previous installation.

     Refer to the Section called *Certificate Configuration* for instructions on how to create the keystore.

   - The various configuration files are located in the `/etc/fms` directory. The configuration files are automatically created during installation.

     Configuration setup details are documented in the Section called *Server Administration* in Chapter 3.

   - Digital Signature and Encryption setup must to be configured as per the Section called *Messaging Security* in Chapter 3.

- If the ebXML protocol is used then either a processing mode (P-Mode) or Collaboration Protocol Agreement (CPA) must be created The CPA settings will override any corresponding P-Mode settings.

  The FMS client must specify the location of the CPA should CPA support be required. Refer to Appendix E for the necessary details.
- Server certificates MUST be exported and imported into the client keystore before the client process can be used to send messages to the server. Refer to the Section called *Exporting a Public Key (Certificate)* for further information.

Linux FMC Installation

The following steps need to be taken to install the FMC:

1. Ensure that JRE version 1.8 or later is installed
2. As the `root` user execute the `rpm` install command as follows for RPM based package systems such as Redhat, Fedora or CentOS systems

   ```
   [root@localhost ~]# rpm -ivh fms-mc-<VERSION>.noarch.rpm
   ```

   For Debian based package systems such as Ubuntu execute the install command as follows

   ```
   [root@localhost ~]# dpkg --install fms-management-console-<VERSION>.deb
   ```

   If upgrading RPM systems execute the rpm upgrade command as follows

   ```
   [root@localhost ~]# rpm -Fvh fms-mc-<VERSION>.noarch.rpm
   ```

   For Debian systems execute the upgrade (same as install) command as follows

   ```
   [root@localhost ~]# dpkg --install fms-management-console-<VERSION>.deb
   ```

Windows Distribution

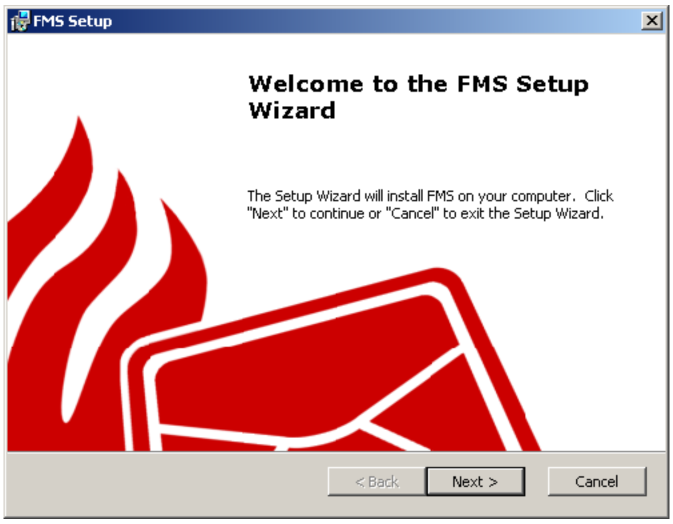
The Windows installation distribution supports the following Microsoft Windows® versions:

- Windows 7 to 10
- Windows Server 2008 to 2016

Windows Installation

The FMS distribution contains both the server, AS4 light client and management console. To install from the FMS distribution proceed with the following steps:

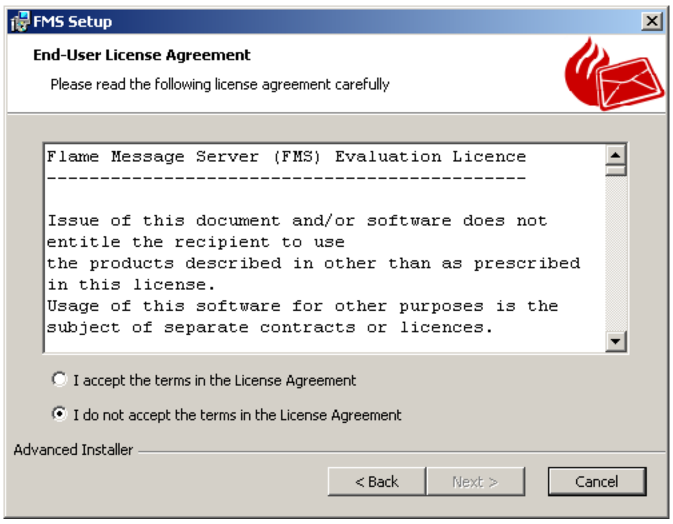
1. Locate or download the FMS Windows Installer distribution, typically named `fms-setup-version.exe`.
2. Double click the file `fms-setup-version.exe` where version depicts that actual release version number.



The Setup Wizard will install FMS on your computer. Click "Next" to continue or "Cancel" to exit the Setup Wizard.

Figure 2-1. Windows Installer Front Page

3. Read the license and click "I accept" if appropriate.


FMS Setup
End-User License Agreement
Please read the following license agreement carefully
Advanced Installer
< Back
Next >
Cancel

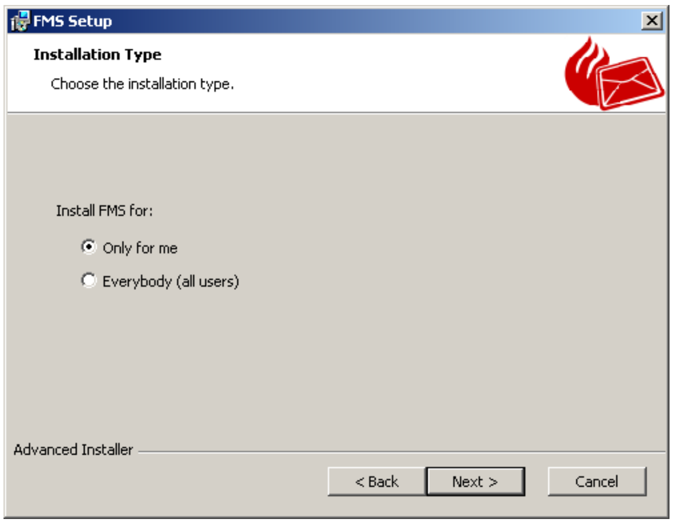
```
Flame Message Server (FMS) Evaluation Licence
---------------------------------------------

Issue of this document and/or software does not
entitle the recipient to use
the products described in other than as prescribed
in this license.
Usage of this software for other purposes is the
subject of separate contracts or licences.
```

I accept the terms in the License Agreement [✓ I do not accept the terms in the License Agreement]

Figure 2-2. Windows Installer License

4. Select the user access and permissions for the installation.


FMS Setup

Installation Type

Choose the installation type.

Install FMS for:

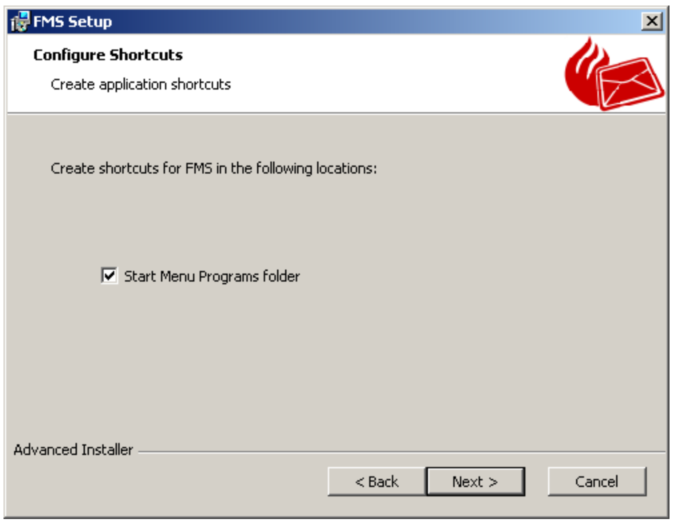
[✓ Only for me]

Everybody (all users)

Advanced Installer

Figure 2-3. Windows Installer Installation Type

5. Configure Shortcuts.


FMS Setup

Configure Shortcuts

Create application shortcuts

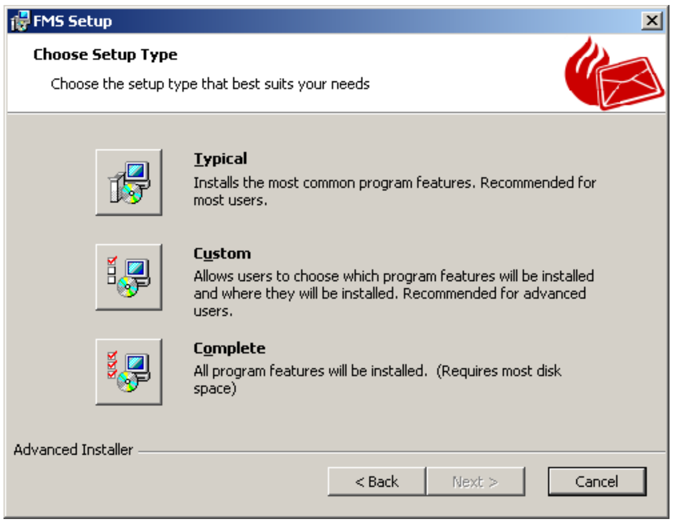
Create shortcuts for FMS in the following locations:

☑ Start Menu Programs folder

Advanced Installer

Figure 2-4. Windows Installer Shortcuts

6. The features, inclusion of the Java JRE, and installation directory (typically `\Program Files\FMS\`) may be configured by selecting the `Custom` setup type.



Choose Setup Type

Choose the setup type that best suits your needs

Typical

Installs the most common program features. Recommended for most users.

Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

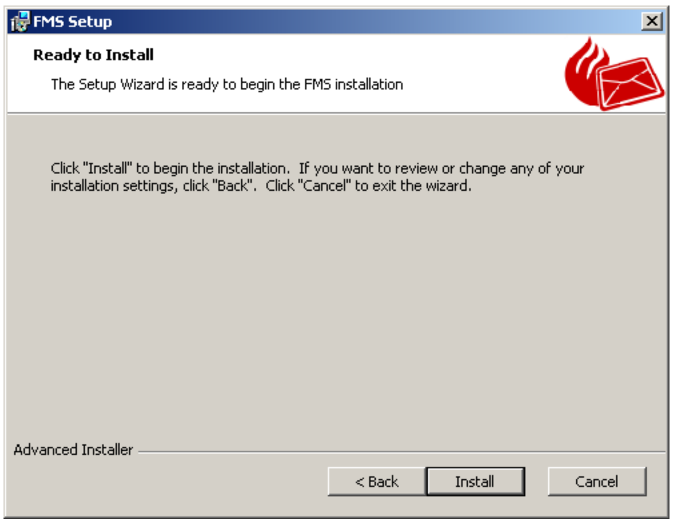
Complete

All program features will be installed. (Requires most disk space)

Advanced Installer

Figure 2-5. Windows Installer Setup Type

7. Windows Installer is ready to install the program files and service.



Ready to Install

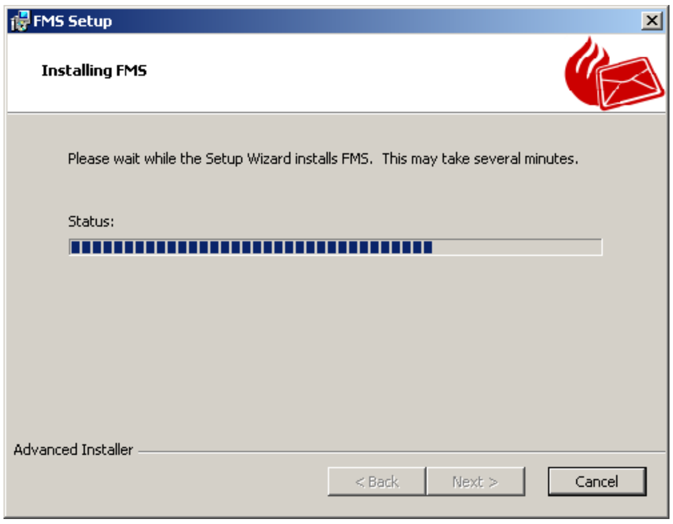
The Setup Wizard is ready to begin the FMS installation

Click "Install" to begin the installation. If you want to review or change any of your installation settings, click "Back". Click "Cancel" to exit the wizard.

Advanced Installer

Figure 2-6. Windows Installer Setup Type

8.

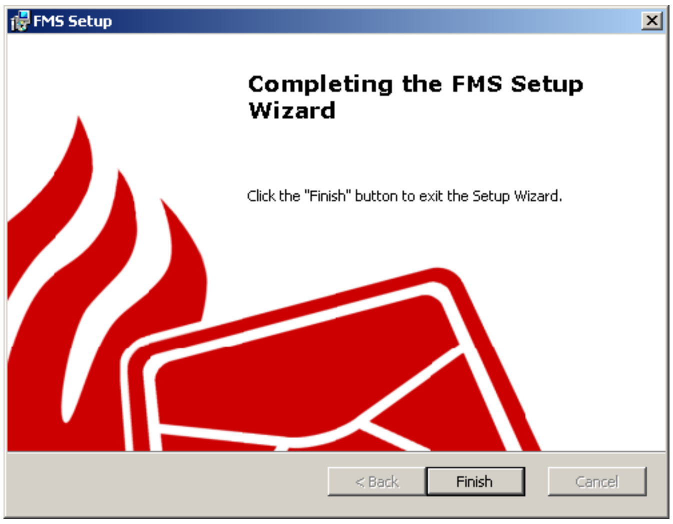


Please wait while the Setup Wizard installs FMS. This may take several minutes.

Status:

Figure 2-7. Windows Installer Installing

9. Click "Finish" to exit.



Completing the FMS Setup Wizard

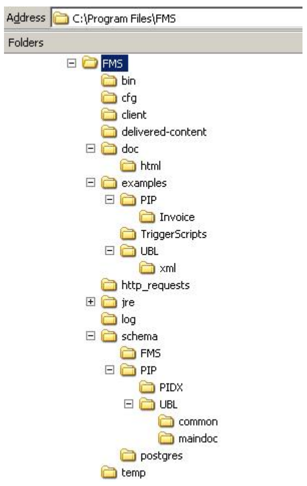
Click the "Finish" button to exit the Setup Wizard.

Figure 2-8. Windows Installer Finished

The majority of the configuration files are created by the server after the first invocation except for the License file '`fms.lcn`' as discussed in the Section called *Licence Configuration*, and the certificate keystore '`certs`' as discussed in the Section called *Creating a Keystore using the Key Generator*. Both the license and certificate keystore must be installed in the installation directory which defaults to `\Progam Files\FMS` for successful operation.

Windows Directory Structure

The default directory structure for Windows is as per the following diagram

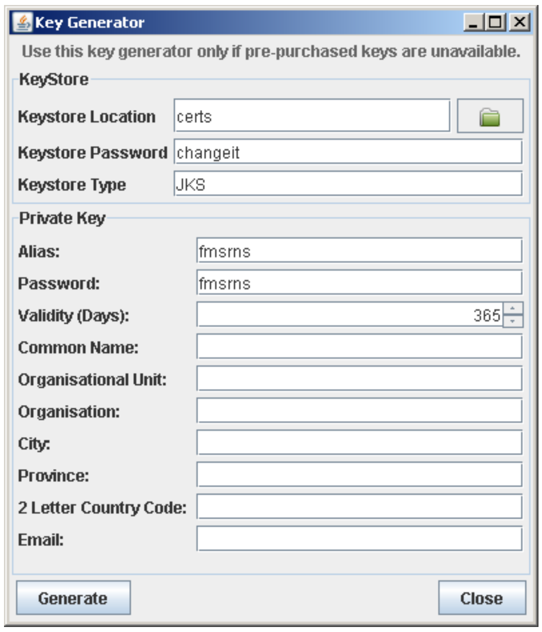


```
Address  C:\Program Files\FMS
Folders
FMS
  bin
  cfg
  client
  delivered-content
  doc
    html
  examples
    PIP
      Invoice
    TriggerScripts
    UBL
      xml
  http_requests
  jre
  log
  schema
    FMS
    PIP
      PIDX
      UBL
        common
        maindoc
    postgres
  temp
```

Figure 2-9. Windows Installation Directory Structure

Creating a Keystore using the Key Generator

When the FMS Installer has exited, select the `Key Generator` shortcut in the start menu to generate a default keystore containing private and public keys. Alternately refer to the Section called *Certificate Configuration* for details on manual key generation.



Use this key generator only if pre-purchased keys are unavailable.

KeyStore

Keystore Location	certs
Keystore Password	changeit
Keystore Type	JKS

Private Key

Alias:	fmsrns
Password:	fmsrns
Validity (Days):	365
Common Name:	
Organisational Unit:	
Organisation:	
City:	
Province:	
2 Letter Country Code:	
Email:	

Figure 2-10. GUI Based Key Generator

The Key Generator is a simple access first start application used to generate keystores for containing private and public keys. The generated keys are in RSA 1024bit format.

The default Keystore Name, Type, Alias and Password should be used for intial configuration of FMS. The default settings can subsequently be changed using the management console.

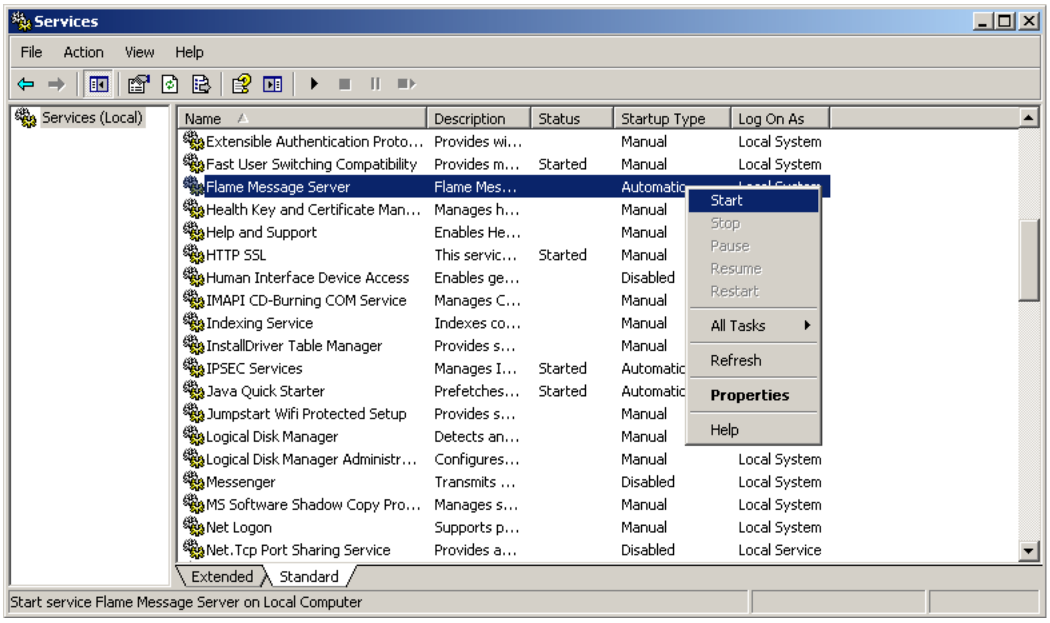
The following information needs to be provided:

- *Common Name* - The hostname that this private key and certificate will be used for.
- *Organisational Unit* - The department within the organisation using this key.
- *Organisation* - The organisation using this key.
- *City* - This city that the organisation is located in.
- *Province/State* - The Province/State where the city is located.
-

  2 Letter Country Code - The 2 letter ISO 3166-1 country code, eg: NL, UK, US, ZA.
- *Email* - The email address of the person responsible for the certificate.

Starting the FMS Service

The FMS Microsoft Windows® Installer configures a new service named Flame Message Server which uses the `FMSService.exe` executable. Start FMS by opening the Services console and selecting and starting the Flame Message Server as per the following diagram



Name	Description	Status	Startup Type	Log On As
Extensible Authentication Proto...	Provides wi...		Manual	Local System
Fast User Switching Compatibility	Provides m...	Started	Manual	Local System
Flame Message Server	Flame Mes...		Automatic	
Health Key and Certificate Man...	Manages h...		Manual	
Help and Support	Enables He...		Manual	
HTTP SSL	This servic...	Started	Manual	
Human Interface Device Access	Enables ge...		Disabled	
IMAPI CD-Burning COM Service	Manages C...		Manual	
Indexing Service	Indexes co...		Manual	
InstallDriver Table Manager	Provides s...		Manual	
IPSEC Services	Manages I...	Started	Automatic	
Java Quick Starter	Prefetches...	Started	Automatic	
Jumpstart Wifi Protected Setup	Provides s...		Manual	
Logical Disk Manager	Detects an...		Manual	
Logical Disk Manager Administr...	Configures...		Manual	Local System
Messenger	Transmits ...		Disabled	Local System
MS Software Shadow Copy Pro...	Manages s...		Manual	Local System
Net Logon	Supports p...		Manual	Local System
Net.Tcp Port Sharing Service	Provides a...		Disabled	Local Service

Start service Flame Message Server on Local Computer

Figure 2-11. Windows Service Console

Should the server not start up confirm that both the license and keystore have been installed correctly. Also check the logs which will by default be written into the `log` directory under the FMS installation directory.

Starting another FMS

It is also possible to run the server in a console by invoking the standalone `fms.exe` executable. This may be useful in a test environment where two FMS server need to be active on the same machine.

Advanced users may also use the Windows service utilities to create a second FMS service.

Either way ensure that the FMS installation directory is replicated to another directory for the second server and that a second database or separate schema is used for the second server. Ensure that the FMS administration port and messaging incoming interface ports are set to different values for the respective servers in the `cfg\ConnectionConfiguration.xml` file and that the `bin\fms.ini` has the path values adjusted for the new directory. Also note that default log files will appear in the log directory as per the configuration.

Post Installation Setup

FMS has certain post installation requirements after product installation to ensure smooth operation. These are described in this section.

Licence Configuration

A licence (`fms.lcn`) MUST be obtained from the provider and placed in the FMS installation directory.

The licence is only valid for a specific amount of time, typically one year. Once it expires the server will halt operation and not restart, a new licence must be obtained to allow for normal operation.

The licence detail including the licence holder and the licence expiry date is written to the system logger. Details on where the server outputs logging detail is in the Section called *Log4j Configuration* in Chapter 3.

Contact <`fms@flame.business`> for more information on obtaining a licence.

Keystore Configuration

Certificate Configuration

Certificates come in two forms, one being a certificate as obtained from a commercial certificate authority (CA) such as Verisign or Thawte, and the other being a self generated certificate.

Commercial certificates are signed by a CA and are generally accepted by browsers without being prompted to accept the certificate.

Both the server (producer) and the client (consumer) require a keystore where the server keystore will contain the server private key and the client Public Key (certificate) as generated on the client machine. Conversely the client Keystore contains the Client Private Key and the Server Public Key (certificate). These keys are used to enable secure message communication between the client and the server as well as between server and server. The keys are also used to encrypt and digitally sign the content of any envelope communicated between a local and remote server.

Obtaining a CA Signed Certificate

Purchase a certificate from a recognised Certificate Authority and proceed with the Section called *Importing Public Keys (Certificates) into the Server Keystore* for installing it.

Generating a Private and Public key (Certificate)

A private and public key can be generated using readily available tools such as the Java `keytool` utility with the generated keys stored in a keystore. Separate keystores must be created for the server and client. Different aliases must also be used for each key stored in the keystore.

See the Section called *Keystore Setup and Examples* for generating the keystore containing the private and public keys.

The keystore is secured with a configurable password. By default this password is `changeit`. Any configuration that uses a keystore to access certificates MUST have the correct password configured. Entering a different password when generating a key will require re-configuration of the connection interfaces as per the Section called *FMS Messaging Configuration* in Chapter 3.

In addition to the keystore password, an additional password for the actual private key (-keypass) is required which will be prompted for or can be set using the `-keypass` argument to keytool. The default password is `fmsrns`. Changing this requires re-configuration of the interface configuration.

Note that PKCS12 keystores permit only a single private key/certificate combination.

It is strongly advised that the default passwords be changed from the above. The affected configuration file is called `ConnectionConfiguration.xml` typically located in directory `/etc/fms/` for the Linux distributions and in the FMS installation directory, typically `Program Files\fms`, for the Windows distributions.

Exporting a Public Key (Certificate)

The local server public key must be exported for importing into the remote keystore. The remote public key must be exported for importing into the local keystore.

See the Section called *Keystore Setup and Examples* for exporting the public keys.

Importing Public Keys (Certificates) into the Client Keystore

Obtain the local server public key as per the Section called *Exporting a Public Key (Certificate)*.

See the Section called *Keystore Setup and Examples* for importing the public keys into the client keystore.

Importing Public Keys (Certificates) into the Server Keystore

Obtain the client public key and remote server public key as per the Section called *Exporting a Public Key (Certificate)*.

See the Section called *Keystore Setup and Examples* for importing the public keys into the server keystore.

Keystore Setup and Examples

The following Linux examples provide the necessary steps for setting up the keystores containing the private and public keys on both the local (producer) and remote (consumer) hosts. Prior to running these examples ensure that the `PATH` environment is set to include the directory where the `keytool` executables reside. This is typically the same directory where the Java executable resides.

Note that, in this case, the keystore acts as the truststore for remote host certificates.

```
# Generate the server keystore containing the server private and
# public keys on the local server (producer).
# Set storetype to PKCS12 for PKCS keystores
cd ~fms
keytool -genkeypair -keyalg RSA -validity 365 -keystore certs \
  -storepass changeit -keypass fmsrns -alias fmsrns -keysize 2048 -storetype JKS

# Optionally generate the server keystore containing the server private and
# public keys on the remote server (consumer).
cd ~fms
keytool -genkeypair -keyalg RSA -validity 365 -keystore certs \
  -storepass changeit -keypass fmsrns -alias fmsrns -keysize 2048 -storetype JKS

# Export the public key from the local server (producer) keystore.
keytool -export -file producer.public.cer -keystore certs \
  -storepass changeit -alias fmsrns -storetype JKS

# Export the public key from the remote server (consumer) keystore.
# Alternately obtain it from the system administrator on the remote server
keytool -export -file consumer.public.cer -keystore certs \
  -storepass changeit -alias fmsrns -storetype JKS

# Import the remote server public key (consumer) into the local server keystore (producer)
# after having obtained it from the remote server
cd ~fms
keytool -import -file consumer.public.cer -keystore certs \
  -storepass changeit -alias consumer_fmsrns -storetype JKS

# Import the local server public key (producer) into the remote server keystore (consumer)
# after having provided it to the remote server
cd ~fms
keytool -import -file producer.public.cer -keystore certs \
  -storepass changeit -alias producer_fmsrns -storetype JKS
```

Database Installation and Configuration

Installing and configuring the database and creating the FMS database schema requires the necessary database administration permissions and skills. Also see the PostgreSQL documentation and in particular the Server Configuration and Client Authentication chapters.

UNIX or Linux PostgreSQL Installation

Ensure the PostgreSQL version 12 or later database server is installed. To verify the version query the software installation database for the PostgreSQL packages.

The following example illustrates querying an RPM based PostgreSQL database distribution.

```
[root@localhost ~]# rpm -qa postgres*
```

The following example illustrates querying a debian based PostgreSQL database as used in Ubuntu.

```
[root@localhost ~]# dpkg -l postgres*
```

Once PostgreSQL is installed, configure the database to allow TCP/IP connections by changing the `listen_addresses` entry in the `postgresql.conf` file as per the following example

```
listen_addresses = 'localhost,192.168.0.1'
```

where address `192.168.0.1` is the TCP/IP address of the computer connecting to the database.

Ensure that the entries for local connections are set to trust in `pg_hba.conf` else PostgreSQL will prompt for a password when connecting to database `fms` as user `fms` should the user `fms` not also exist as a system user.

The entry in `pg_hba.conf` should take the following form and restart PostgreSQL for unconditional user access without authentication

host sameuser all 127.0.0.1/32 trust

and as follows if password authentication across all users from any host is required

host all all 0.0.0.0/0 md5

The above configuration change requires restarting PostgreSQL.

Refer to the PostgreSQL documentation for the further configuration details.

Once the PostgreSQL database server is configured and started, the database user which FMS uses to connect to the database must be created as follows

First change to the default PostgreSQL user as follow

```
[root@localhost ~]# su - postgres
```

Create the FMS database user by executing the following command as user postgres. The password for the postgres user is not set by default. If set obtain the password from the system administrator. .

```
[postgres@localhost ~]# createuser -d -S -R -D fms
```

The above command creates a PostgreSQL user called FMS with no superuser privileges and requiring no password.

Create the database as follows.

```
[fms@localhost ~]$ createdb --owner fms fms
```

Execute the provided `databaseCreation.sql` script located in the `fms/schema/postgresql/` directory to create the required database tables as follows

```
[fms@localhost ~]$ cd /home/fms/schema/postgres/
[fms@localhost ~]$ psql --username fms --file databaseCreation.sql fms
```

Alternatively the `fms.sql` script may be used. This script is a wrapper that creates the user and database and then drops the FMS tables should these exist by including `dropTables.sql`. It then creates the tables in optional schemas by including `databaseCreation.sql`.

Feel free to to make any adjustments to the SQL code in the scripts including creating and using separate schemas as discussed in the Section called *Advanced PostgreSQL Database Installation and Configuration*.

Windows PostgreSQL Database Installation and Configuration

PostgreSQL natively supports the Windows operating systems. The following steps illustrate the required steps for installing and configuring the PostgreSQL database server as well as setting up the table schema as used by FMS.

1. Install the Windows PostgreSQL version 12 or later distribution available for free download at http://www.postgresql.org.
2. Create an `fms` user and database using the `pgAdmin` utility. Ensure that the `fms` database is owned by the `fms` user.
3. Create the `fms` data table schema as follows
    - Select the `fms` database, select *OK* if the Database Encoding window pops up, then open the SQL dialog by selecting the *'Execute arbitrary SQL queries'* button.
    -

      In the SQL dialog click on the open button and select the `fms.sql` file in the the '`\Program Files\FMS\schema\postgres`' directory.

      The `fms.sql` script is a wrapper that first drops the `fms` tables, should these exist, by including `dropTables.sql`, and then creates the tables by including `databaseCreation.sql`. Feel free to to make any adjustments to the SQL code in the script including creating and using separate schemas as discussed in the Section called *Advanced PostgreSQL Database Installation and Configuration*.

      Optionally select the `databaseCreation.sql` if creating the tables for the first time in the default schema.
    - Select the *Execute Query* button to execute the table creation script. Once completed a summary will be displayed below. If successful then close the SQL dialog.

      Modify the owner of the schema in which the `fms` tables are created to `fms` by right clicking on the schema and selecting properties. Then, if necessary, change the owner by selecting `fms` in the drop down list.

Advanced PostgreSQL Database Installation and Configuration

Advanced users may wish to install the PostgreSQL on a separate machine from the FMS machine and may also wish to create separate database schemas for use with FMS. To do so is feasible but requires the necessary knowledge of PostgreSQL.

Remote PostgreSQL Database Installation

The database may be installed on a separate machine from the FMS machine. The only criteria is that the database machine is reachable by the FMS machine over a fast network connection and that the database on the remote machine is configured to accept connections from the machine hosting FMS. This is achieved by specifying the database host in the FMS database configuration .

Database Schemas

A database may optionally have multiple schemas created where a single database may be used for several trading partners but with a separate schema for each partner. Separate schemas may also be created on the same database for each interface. It is also possible to create a set of schemas for each company on a single database in the case of the enterprise version. Separate schemas may also be created for the `production` and `test` systems.

Details on creating database schemas are outside the scope of this manual but are well described in the PostgreSQL documentation.

FMS associates databases at the `Interface` level where an optional schema may be specified. If a schema is specified then it is essential to create the data table configuration for that schema. This may be done by simply editing the `fms.sql` database creation script and removing the comments from the lines prior to including the the table drop (`dropTables.sql`) and creation (`databaseCreation.sql`) scripts.

Chapter 3. Configuring FMS

Introduction

Configuring FMS requires knowledge of various disciplines, including networks, security protocols, keystores and certificates, business agreements, business processes and messaging protocols and services.

The FMC utility is the preferred way of configuring FMS. and may be invoked by clicking on the FMC icon in Microsoft Windows® or by invoking `fms-config` in other operating systems. Direct editing of the various FMS configuration files is permissable for advanced users who have an understanding of XML file structure.

In addition to the FMS configuration, the FMC provides FMS system management interfaces for the following

- Keystore creation.
- Security certificate management.
- Administration connection to multiple FMS servers.
- Log monitoring across multiple FMS servers.
- FMS server control including reloading of connections and configurations.
- Reporting of connection specific realtime statistics and message status across multiple FMS servers.

Initialisation Overview

During the first invocation FMS will initialise default configuration files and directories and populate these with default settings and a single administration connection interface listener.

FMS uses an initial server settings file and a server messaging configuration file, as follows

1. The FMS server settings file `main.conf`, automatically created and populated with initial server settings as detailed in the Section called *Server Settings*.
2. The default configuration file, `ConnectionConfiguration.xml`, automatically created and populated with default messaging settings.

   This file defines the configurations required for communicating messages between partners and contains configuration settings for the following

   Partner Identifiers

   : This section is used for storing specific partner values such as the partner identity and type, identifier type, keystore alias and password, the partner endpoint URL, and a comma delimited set of content codes used for authorisation purposes.

   Processing Modes (P-Mode)

   : This section contains the processing mode (P-Mode) configurations for message communication between an initiating partner and a responding partner, including the events or legs with associated triggers for a message conversation.

   Interfaces

   : This section contains the port settings and protocol definitions for the supported interfaces including the ebXML, RosettaNet, Reliable Messaging and administration connections. It also contains the database configuration for message logging per interface and associated triggers.

   KeyStores

   : This section contains references to filesystem-based KeyStores. KeyStores contain the private and public keys (Certificates) used when signing and/or encrypting outbound messages and verifying and decrypting inbound messages.

Services

This section contains a list of Services, with each service associated with a business process.

A service contains a subset of Dictionary action names relevant to a required business process which may be referenced from a Processing Mode Business Information Service and set of associated Actions.

Dictionaries

This section contains a mapping between the payload schema names (Actions) and a Dictionary identifier to be referenced by Service entries. A Dictionary would typically consist of a set of XML Schemas corresponding to a specification such as UBL (Universal Business Language), PIDX (Petroleum Industry Data eXchange), or EPP (Extensible Provisioning Protocol).

Packagers

This section contains a mapping between a text name and a Java class path denoting the location of a package manager to load, as well as relevant packager protocol properties used for various stages of messaging including security and messaging protocol validation processes.

The enterprise version of FMS is required to create new protocol packages.

System Configuration

After installation the system needs to be configured to allow successful message communication between business partners. This section discusses the FMC and the various configuration settings in detail.

Prior to invoking the management console to configure the system ensure that the server keystore containing private and public keys are created as described in the Section called *Generating a Private and Public key (Certificate)* in Chapter 2, and that the FMS server is running with the administration interface listening for connections (default).

Admin User Creation

The FMC connects to the server over a secure authenticated connection. The default user is `admin`. The password must be created in file `user.cfg` on the machine on which the server is installed prior to connecting to the FMS server. This is done as follows

Windows

1. Open a new command console in windows and navigate to the FMS configuration directory.
   ```
   cd "\Program Files"\FMS\cfg\
   ```
2. Invoke the `adduser.exe` utility with the username as the argument as follows
   ```
   ..\bin\adduser admin
   ```
3. Enter the user password, and again to confirm.

Repeat the above steps to reset a password.

UNIX and Linux

1. Open a console and navigate to the FMS configuration directory.
   ```
   cd /home/fms/
   ```
2. Locate and invoke the jar file `fms_connections.jar` with `adduser` and the username as the argument in the configuration directory.
   ```
   java -jar /path/to/fms_connections.jar adduser admin
   ```
3. Enter the user password, and again to confirm.

Repeat the above steps to reset a password.

Admin User Deletion

Delete a user by using a text editor to remove the line containing the user name in the `user.cfg` file.

FMC

FMC is the primary administration interface for administering the FMS server. Invoke FMC as follows

Windows

Select `FMC` from the Windows command menu and log in with the user and password as set using the `adduser.exe` utility.

UNIX and Linux

Invoke FMC from the command line as follows

```
[user@localhost fms]$  java -jar fms_mc.jar &
```

Server Administration

The FMS server configuration is stored in an XML file typically called `ConnectionConfiguration.xml`. The configuration can be accessed either via connecting the FMS Management Console (FMC) to an FMS server or by directly editing the FMS configuration file using a standard text editor and the fmsconf utility as described in section the Section called *fmsconf Server Configuration Utility* in Chapter 6.

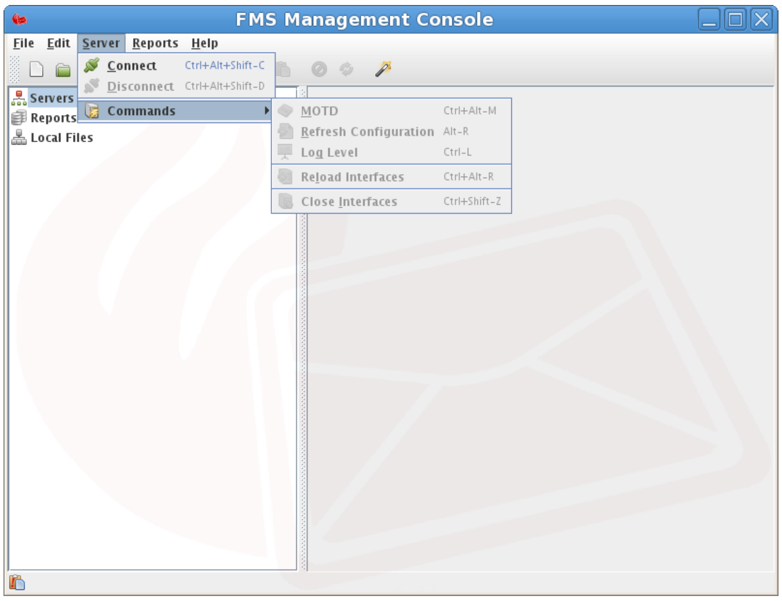

FMS Management Console
File Edit Server Reports Help
Connect Ctrl+Alt+Shift-C
Disconnect Ctrl+Alt+Shift-D
Commands
Servers
Reports
Local Files
MOTD Ctrl+Alt-M
Refresh Configuration Alt-R
Log Level Ctrl-L
Reload Interfaces Ctrl+Alt-R
Close Interfaces Ctrl+Shift-Z

Figure 3-1. Screen shot of the FMC before opening an administration connection to the server

Command Descriptions:

- MOTD

  View the Message Of The Day as set in the administration interface.

- *Refresh Configuration*

  Reload the Configuration file but do not reload the live connections, useful when adjusting Partner Identifiers, Processing Modes/Events and other business orientated configurations.

- *Log Level*

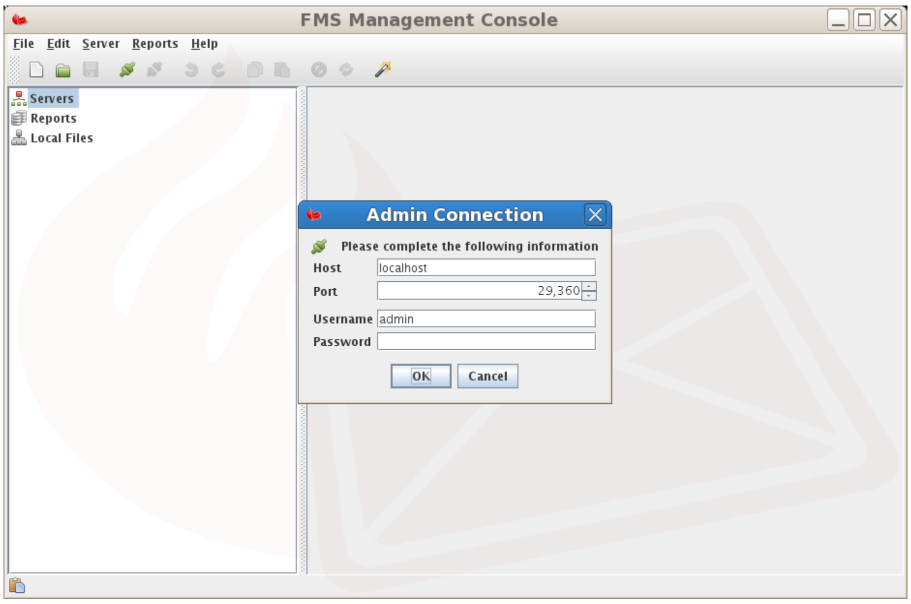
  Adjust the current server logging level. The previous log level will be indicated by the selected button in the list of log levels.

- *Reload Interfaces*

  Close all the current connections and reload the Interface Configuration, useful when updating port numbers, hostnames, or security requirements, refer to the Section called *FMS Messaging Configuration* for more information.

- *Close Interfaces*

  Close all the current interfaces. Warning: All the interfaces will be shut down, including the administration interface (excluding current sessions). A server restart will be required to re-establish the interfaces should the management console disconnect while the interfaces are closed.



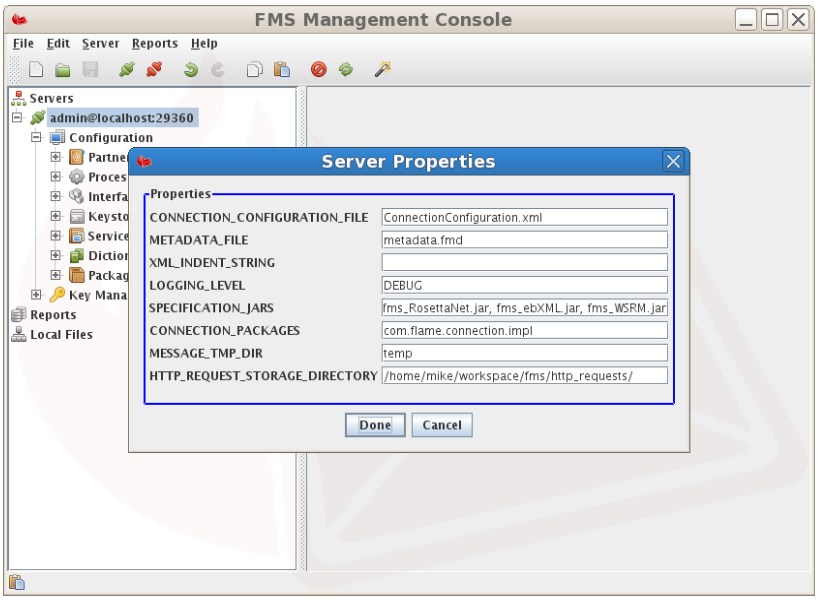
Host: localhost
Port: 29,360
Username: admin
Password:

Figure 3-2. Screen shot of the FMC prior to making a connection

Server Settings

The server initial settings, typically stored in `main.conf`, can be viewed by right clicking on the `Server` connection node and selecting `Properties`.

The detail of the main configuration is as follows



CONNECTION_CONFIGURATION_FILE	ConnectionConfiguration.xml
METADATA_FILE	metadata.fmd
XML_INDENT_STRING	
LOGGING_LEVEL	DEBUG
SPECIFICATION_JARS	fms_RosettaNet.jar, fms_ebXML.jar, fms_WSRM.jar
CONNECTION_PACKAGES	com.flame.connection.impl
MESSAGE_TMP_DIR	temp
HTTP_REQUEST_STORAGE_DIRECTORY	/home/mike/workspace/fms/http_requests/

Figure 3-3. Default Configuration Example

CONNECTION_CONFIGURATION_FILE

The name of the default Configuration File. This file is stored in the FMS startup directory and contains configuration settings including partner, processing mode, interface definitions, business services and protocol packagers as required for a functioning FMS.

HTTP_REQUEST_STORAGE_DIRECTORY

Incoming requests are stored in this directory in the event of the Non-Repudiation database being unavailable.

LOGGING_LEVEL

The starting Log Level for FMS with default set at INFO. This can be set to any one of OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, ALL.

SPECIFICATION_JARS

A comma ',' delimited list of Java archives (Jars) containing specification schema (xsd), document type definition (dtd) and java class files to be referenced for entity resolution when validating/parsing XML incoming and outgoing message payloads.

The default setting is `fms_RosettaNet.jar`, `fms_ebXML.jar`, `fms_WSRM.jar`.

This setting is relevant in the enterprise edition when extending the available interfaces.

XML_INDENT_STRING

XML Indent String (Spaces or \t for Tab). Defaults to two (2) spaces.

MESSAGE_TMP_DIR

A temporary directory, automatically created, relative to the installation directory used for intermediate message payload and attachment files. The default name is `temp`.

METADATA_FILE

The default FMS metadata file name (`metadata.fmd`) as created in the `delivered-content/initiatingParty/respondingParty/message-id` directory where `message-id` is a unique identifier for each received message ensuring that each received message has it's own associated metadata file. This XML file contains information related to delivered message payload and attachments and has the following format

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fmd:FMSMetadataDocument xmlns:fmd="http://fms.flame.business/FMS/schema/FMSMetadataDocument">
  <fmd:Legal>
```

```
    <fmd:Copyright>(c) Flame Computing Enterprises cc. All rights reserved</fmd:Copyright>
  </fmd:Legal>
  <fmd:To>ToPartyID</fmd:To>
  <fmd:From>FromPartyID</fmd:From>
  <fmd:ConversationID>123</fmd:ConversationID>
  <fmd:Service>A06</fmd:Service>
  <fmd:Action>http://docs.oasis-open.org/ebxml-msg/as4/200902/action</fmd:Action>
  <fmd:MessageID>739cdd60-5d45-4132-a43e-48226a4ba383@domain.com</fmd:MessageID>
  <fmd:ProcessingMode>AS4-ENTSOG-PROD</fmd:ProcessingMode>
  <fmd:Event>ReceiveMessage</fmd:Event>
  <fmd:ToRole>ZSH</fmd:ToRole>
  <fmd:FromRole>ZSO</fmd:FromRole>
  <fmd:AgreementRef>ThisAgreement</fmd:AgreementRef>
  <fmd:Timestamp>2019-05-09T15:57:46Z</fmd:Timestamp>
  <fmd:MessagePayloads>
    <fmd:Payload>
      <fmd:MimeContentID>739cdd60-5d45-4132-a43e-48226a4ba383@domain.com</fmd:MimeContentID>
      <fmd:MimeContentType>application/xml</fmd:MimeContentType>
      <fmd:Location
       >/home/fms/delivered-content/FromPartyID/ToPartyID/739cdd60-5d45-4132-a43e-48226a4ba383@domain.com/20190509175746
      </fmd:Location>
    </fmd:Payload>
  </fmd:MessagePayloads>
</fmd:FMSMetadataDocument>
```

FMS Messaging Configuration

After a Business to Business (B2B) Messaging Agreement has been proposed the messaging configuration settings in accordance with the messaging protocol and partner agreements must be defined in order to receive and send B2B Messages.

Messaging configuration details may be configured using the FMC or directly using a text editor in the server `ConnectionConfiguration.xml` file.

The Interface Configuration node of the FMC enables a fast and efficient way of configuring FMS interfaces. This configuration can also be performed manually by editing the file `ConnectionConfiguration.xml`, note that this is only recommended for advanced administrators.

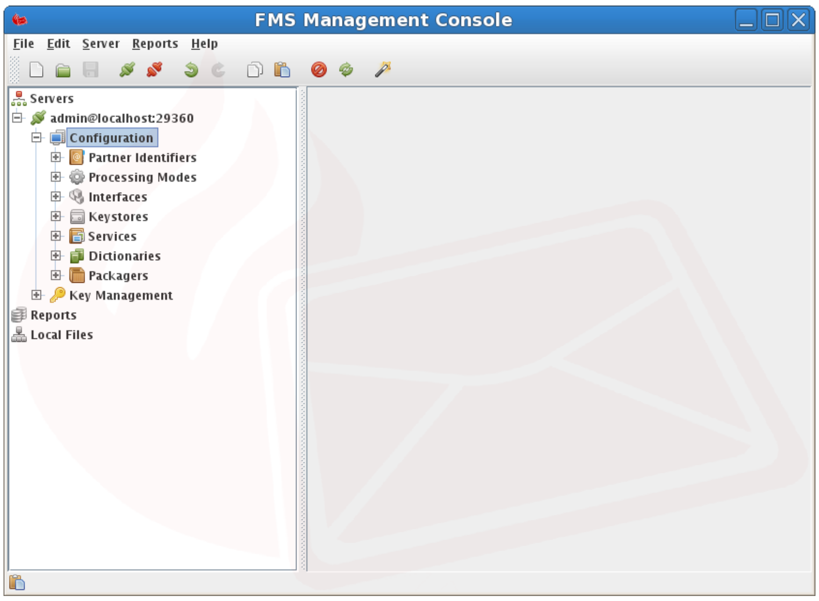
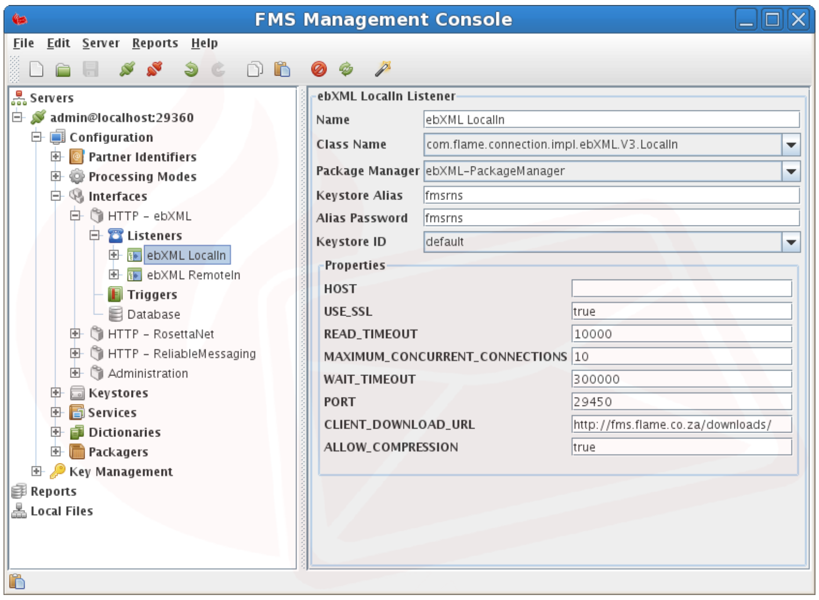


Figure 3-4. Administration Console Configuration screen shot

New Configurations are created by selecting the `New` toolbar button and clicking on the `New Interface` option. This is only necessary for advanced customisation when adding a new protocol to FMS. Typically this is not required because the default interfaces can simply be edited. The security and database settings can be configured on a per Interface Configuration basis as per the above configuration screen shot should these be required. The Interface security level sets a minimum security level for all messages sent using this interface. Note that higher values will override Processing Mode Event Security level settings.

After the server has been started or initialised, all the enabled listener/sender configurations will be populated with default values. This screen is accessible by selecting the node for each listener/sender.



ebXML LocalIn Listener

Name	ebXML LocalIn
Class Name	com.flame.connection.impl.ebXML.V3.LocalIn
Package Manager	ebXML-PackageManager
Keystore Alias	fmsrns
Alias Password	fmsrns
Keystore ID	default

Properties

HOST	
USE_SSL	true
READ_TIMEOUT	10000
MAXIMUM_CONCURRENT_CONNECTIONS	10
WAIT_TIMEOUT	300000
PORT	29450
CLIENT_DOWNLOAD_URL	http://fms.flame.co.za/downloads/
ALLOW_COMPRESSION	true

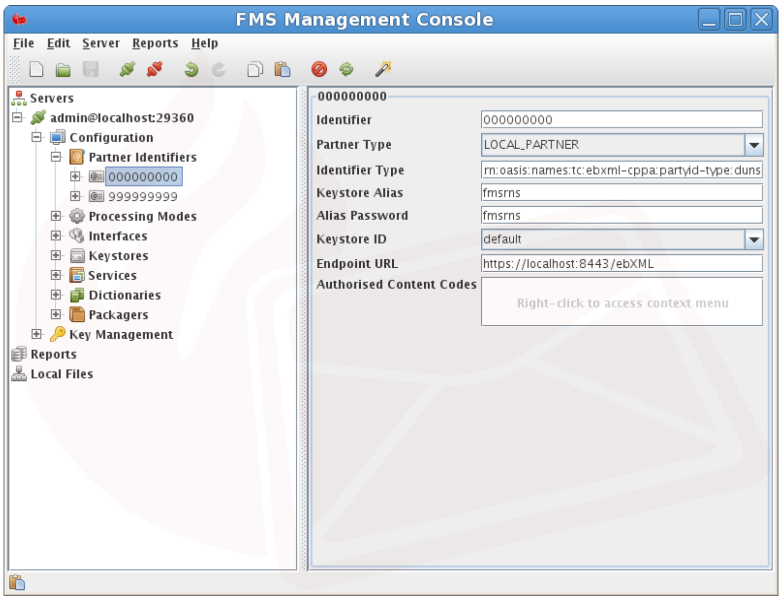
Figure 3-5. Default Configuration example

Each connection has an association to a Packager Configuration Identifier. This allows message transformation by switching between Transport Specifications such as RosettaNet to ebXML.

Partner Identifier Configuration

A mapping of a Partner Identifier (eg. DUNS number) to other relevant partner information needs to be created for use within the system. A default local and remote partner is automatically created by the system on startup. These should be modified according to requirements.

Create or modify an existing `Partner Identifier` entry by selecting *New* from the main menu or toolbar. Select `New PartnerIdentifier`. Complete the required information as follows and select `Save` to finalise the changes.



000000000

Identifier	000000000
Partner Type	LOCAL_PARTNER
Identifier Type	rn:oasis:names:tc:ebxml-cppa:partyid-type:duns
Keystore Alias	fmsrns
Alias Password	fmsrns
Keystore ID	default
Endpoint URL	https://localhost:8443/ebXML
Authorised Content Codes	

Figure 3-6. Partner Identifier Configuration Example

- *Identifier* - The actual value, or name, of the partner identifier. The partner identifier on the local side (producer) must correspond with the partner identifier on the remote side (consumer) and vice versa.
- *Partner Type* - The type of the partner identifier being either a LOCAL_PARTNER or REMOTE_PARTNER.
- *Identifier Type* - The type of the partner identifier (`urn:oasis:names:tc:ebxml-cppa:partyid-type:duns` is used in this example). The type *must* match the type of the PartyID in the PartyInfo section of the CPA if used with the ebMS specification.
- *Keystore Alias* - The Keystore Alias entry stores an Alias name in the Keystore referring to a remote partner's public certificate and the local partners private key. The public key is used to encrypt a message to be sent to the corresponding remote partner where it will be decrypted using the private key. The local partner private key is used to generate digital signature which may be attached to the message[s] and/or to the envelope containing the set of messages and optional attachments. The remote partner MSH uses the local partner public key to verify the signatures.
- *Alias Password* - A password is required if the above alias refers to a Private Key and is used to retrieve the key during envelope and message signing and message decryption.
- *Keystore ID* - The reference to a KeyStore from the server configuration which contains the above Private and/or Public Keys.
- *Partner mailBox ID* - the optional configurable inbound message mailbox for local partners, LOCAL_PARTNER, and outbound message mailbox for remote partners, REMOTE_PARTNER. The mailBox setting will override the interfaceConfig.deliveredContent directory.

  Mailbox directories starting with the directory separator will be treated as absolute. Directories starting without a directory separator will be saved in the FMS installation directory.

  Mailbox directories may either be explicit directory names or customised using the following variables determined from inbound messages for local partners.

  - *$toPartner* - message to partner identifier with any preceding URI elements removed to ensure a consistent directory structure.
  - *$fromPartner* - message from partner identifier with any preceding URI elements removed to ensure a consistent directory structure.
  - *$messageID* - message identifier with any preceding URI elements removed to ensure a consistent directory structure.
  - *$conversationID* - message conversation identifier with any preceding URI elements removed to ensure a consistent directory structure.
  - *$service* - message service with any preceding URI elements removed to ensure a consistent directory structure.

- *$action* - message action identifier with any preceding URI elements removed to ensure a consistent directory structure.

- *Endpoint URL* - The `Endpoint URL` entry stores the endpoint URL of the partner. This being the URL of the remote service that the partner will be invoking. This entry is used in the send or response action during asynchronous service interaction with a remote partner service.

  The actual endpoint used depends on several conditions being as follows

  Processing Mode CPP/A Agreement

  : The endpoint used will be as per the CPP/A agreement value in PRODUCTION mode for the ebXML specification if the processing mode agreement configuration setting is defined.

  Processing Mode Event Endpoint

  : The endpoint defined in the processing mode event will be used if no overriding processing mode CPP/A agreement in PRODUCTION mode is defined.

  Partner Endpoint

  : The endpoint defined for the partner will be used if neither the processing mode event configuration setting is defined nor the CPP/A agreement value in PRODUCTION mode is defined.

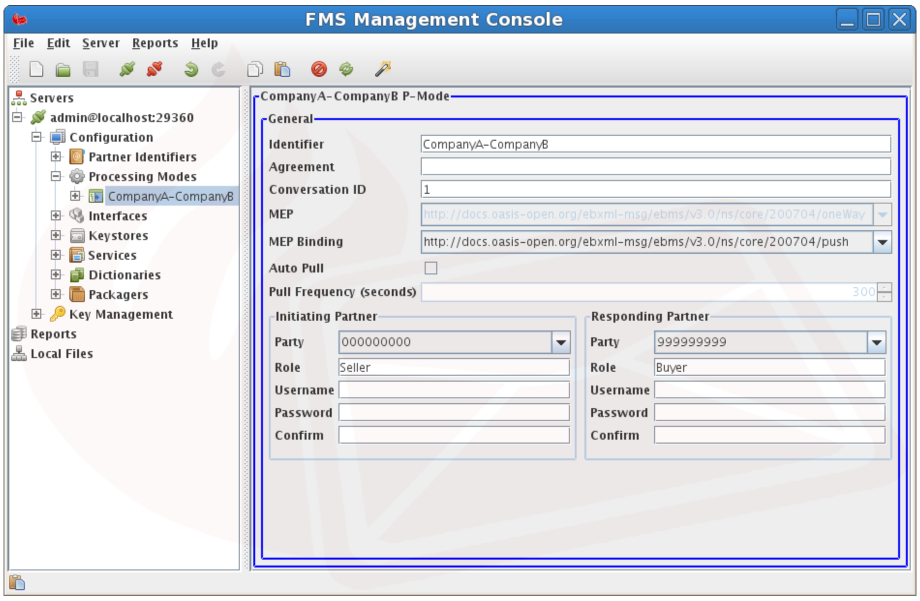
- *Authorisation List* - The section `Authorisation List` stores a list of Schema Codes. A client will be restricted to only send items in this authorisation list. If the list is empty then no restrictions exist.
- *Partner Type* - This is an enumeration of the following values {REMOTE_PARTNER, LOCAL_PARTNER, PULL_PARTNER}.
  - *REMOTE_PARTNER* - Defines that this server *IS NOT* responsible for this Partner Identifier and any incoming messages destined for this Partner Identifier must be delegated to the responsible endpoint URL.
  - *LOCAL_PARTNER* - Defines that this server IS responsible for this Partner Identifier and any incoming messages for this Partner Identifier will be stored locally for retrieval at a later stage.
  - *PULL_PARTNER* - The same as LOCAL_PARTNER, however, the message is stored in a form in which the Partner can perform a Pull Request for retrieval.
- *Triggers* - A list of triggers associated with this PartnerIdentifier, refer to the Section called *Triggers* for more information.

Processing Modes

Processing Modes (as defined by the ebMS Specification) provide a centralized configuration for business messages. Processing Modes consist of the following

- General section - Contains the message pattern, CPA configuration, and partners involved in this processing mode.
- Events section - Defines the endpoint of a message, payload profiles, and other information.
  - Protocol - Contains information for the endpoint, http compression, and message retry configuration.
  - Business Information - Business centric information such as the Service and Action for this message sequence.
  - Security - Security information including digital encryption/signatures, and UsernameToken management.
  - Reliability - WS-ReliableMessaging configuration.
  - Error Handling - Who and where to send error messages to.

ProcessingModes - General Configuration



CompanyA-CompanyB P-Mode

General

Identifier	CompanyA-CompanyB
Agreement	
Conversation ID	1
MEP	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay
MEP Binding	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push
Auto Pull	☐
Pull Frequency (seconds)	300

	Initiating Partner	Responding Partner
Party	000000000	999999999
Role	Seller	Buyer
Username		
Password		
Confirm		

Figure 3-7. Default ProcessingMode-General example

- *Identifier* - A unique identifier for this ProcessingMode, MUST NOT contain spaces and consists of ONLY alpha-numeric characters and hyphens '–'.
- *Agreement* - A URL pointing to a Collaboration Protocol Agreement CPA.
- *Conversation ID* - This element is a string identifying the set of related messages that make up a conversation between Parties.
- *Message Exchange Pattern (MEP)* - This item is not directly configurable since it is defined by the event count in this processing mode.
- *Message Exchange Pattern Binding* - Dependant on the MEP above, if the MEP is OneWay then the MEPBinding may be one of {PUSH, PULL}, however if the MEP is TwoWay then the MEPBinding may be one of {PUSH-AND-PUSH, PUSH-AND-PULL, PULL-AND-PUSH, PULL-AND-PULL, SYNC}. SYNC is only usable if a synchronous trigger is used on receipt of an incoming message.
- *Party* - A dropdown list of Partner Identifiers for defining the respective party for initiating or responding in this ProcessingMode.
- *Role* - The role of the party in this ProcessingMode
- *Username* - The username to use for UsernameToken authentication.
- *Password* - The password to use for the supplied username. Note that confirmation of passwords will need to be performed when adjusting this value.

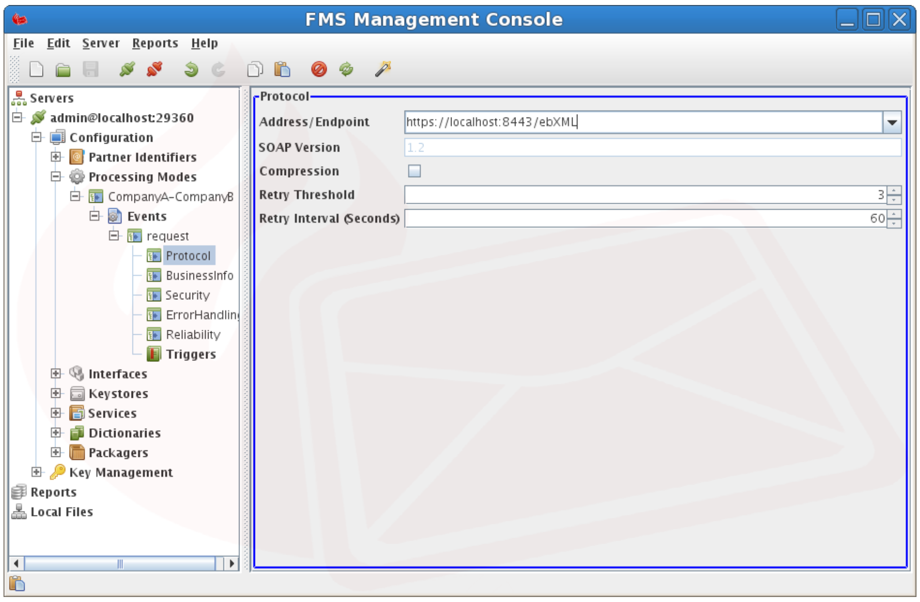
ProcessingModes - Event

An Event is defined as an action in which a message is sent from the initiating party to the responding party (in the first event). The parties are swaped for each subsequent event, i.e. the initiating party in the second event will temporarilly be receiving messages from the responding party.

- *Identifier* - A unique identifier for this ProcessingMode Event, MUST NOT contain spaces and consists of ONLY alpha-numeric characters and hyphens '–'.

Event - Protocol

Defines protocol-related parameters necessary for interoperating, that are associated with a particular message of the MEP.



Protocol

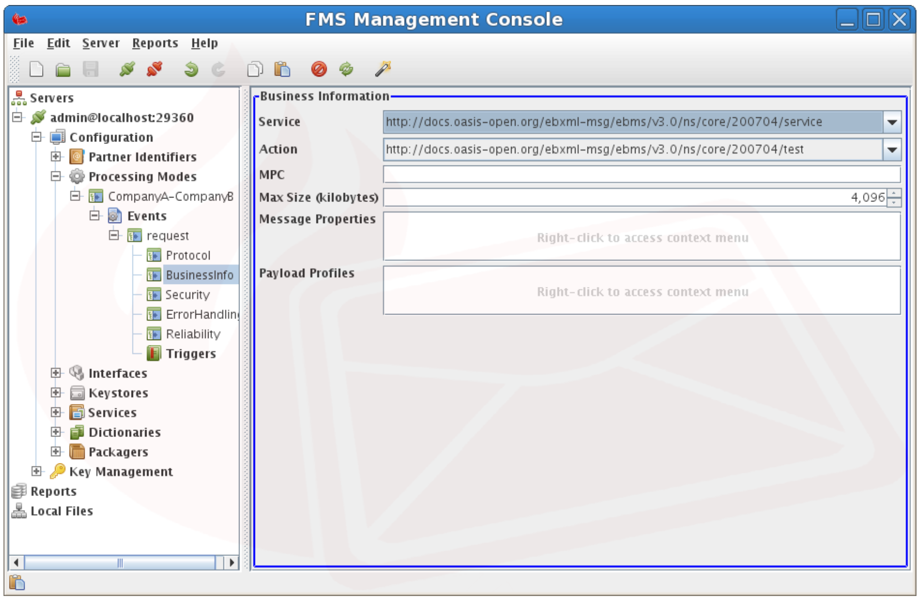
Address/Endpoint	https://localhost:8443/ebXML
SOAP Version	1.2
Compression	☐
Retry Threshold	3
Retry Interval (Seconds)	60

Figure 3-8. Default Event-Protocol example

- *Identifier* - A unique identifier for this ProcessingMode Event, MUST NOT contain spaces and consists of ONLY alpha-numeric characters and hyphens '–'.

Event - Business Information

Defines the business profile of a user message in terms of business header elements and their values (e.g. Service, Action) or other items with business significance (payload profile, MPC).



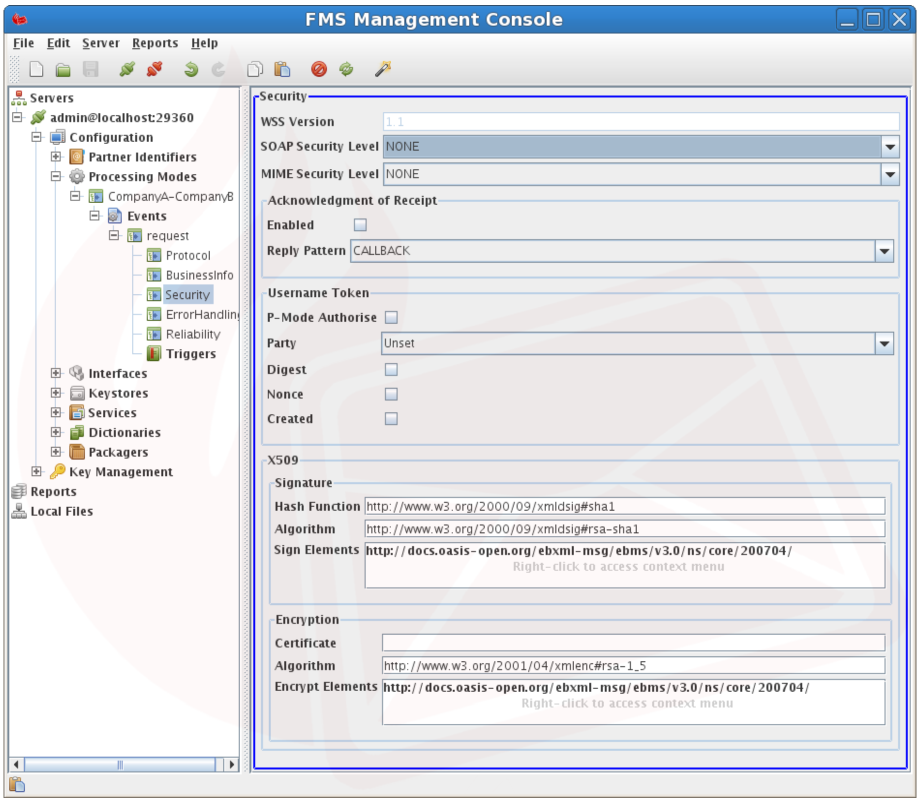
Business Information

Service	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service
Action	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test
MPC	
Max Size (kilobytes)	4,096
Message Properties	Right-click to access context menu
Payload Profiles	Right-click to access context menu

Figure 3-9. Default Event-BusinessInfo example

Event - Security

Defines the security level expected for the message in the exchange, and provides related security context data.



Security
WSS Version: 1.1
SOAP Security Level: NONE
MIME Security Level: NONE

Acknowledgment of Receipt
Enabled: ☐
Reply Pattern: CALLBACK

Username Token
P-Mode Authorise: ☐
Party: Unset
Digest: ☐
Nonce: ☐
Created: ☐

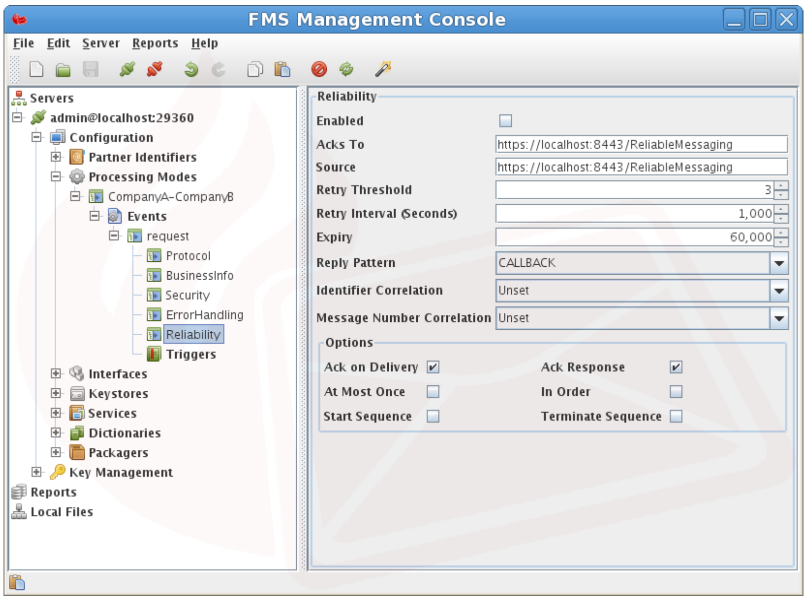
X509
Signature
Hash Function: http://www.w3.org/2000/09/xmldsig#sha1
Algorithm: http://www.w3.org/2000/09/xmldsig#rsa-sha1
Sign Elements: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
Right-click to access context menu

Encryption
Certificate:
Algorithm: http://www.w3.org/2001/04/xmlenc#rsa-1_5
Encrypt Elements: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
Right-click to access context menu

Figure 3-10. Default Event-Security example

Event - ReliableMessaging

Defines the reliability contracts and their parameters, applying to the message in this Event.



Reliability

Enabled	☐
Acks To	https://localhost:8443/ReliableMessaging
Source	https://localhost:8443/ReliableMessaging
Retry Threshold	3
Retry Interval (Seconds)	1,000
Expiry	60,000
Reply Pattern	CALLBACK
Identifier Correlation	Unset
Message Number Correlation	Unset

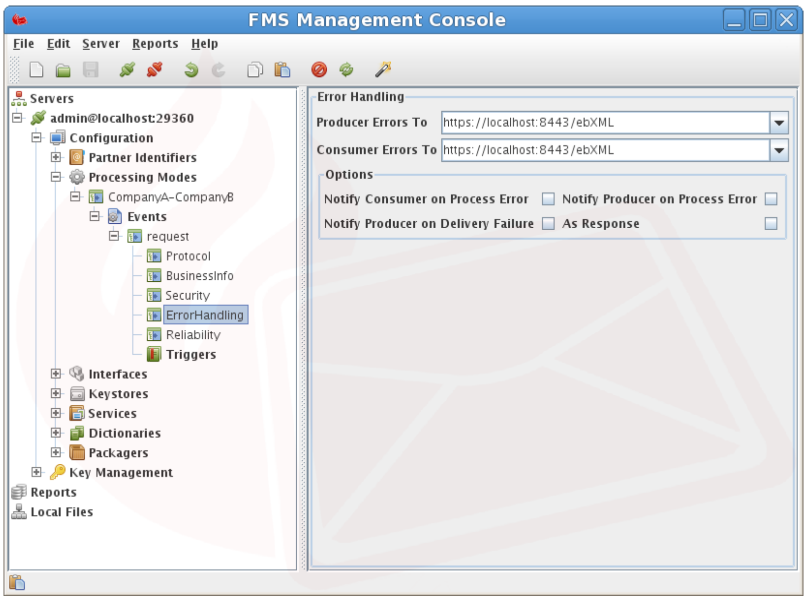
Options

Ack on Delivery	☑	Ack Response	☑
At Most Once	☐	In Order	☐
Start Sequence	☐	Terminate Sequence	☐

Figure 3-11. Default Event-Reliability example

Event - ErrorHandling

Defines the mode of handling and of reporting of errors associated with the message in this Event.



Error Handling

Producer Errors To	https://localhost:8443/ebXML
Consumer Errors To	https://localhost:8443/ebXML

Options

Notify Consumer on Process Error	☐	Notify Producer on Process Error	☐
Notify Producer on Delivery Failure	☐	As Response	☐

Figure 3-12. Default Event-ErrorHandling example

Messaging Security

B2B Messages can be secured using message signing and/or encryption of various elements of both inbound and outbound messages. This section provides the necessary detail of how messages may be sigined and or encrypted.

Digital Signatures

A digital signature is a form of authentication in which message elements such as the message body is digitally signed by a private signing key. When a message is received the receiving agent verifies the digital signature eithe with the enclosed public key, or using an external saved copy of the public key against the various signed message elements. If this verification fails the message will not be accepted since it implies that the original message was tampered with during transmission. Message signing can be configured in the relevant connection configuration as detailed in the Section called *FMS Messaging Configuration*. Refer to the Section called *Certificate Configuration* in Chapter 2 for information on configuring private and public keys.

Encryption

Encyption involves retrieving the recipient's public key (certificate) and encrypting the message content against it. The recipient then uses their private key to decrypt the message. This 'swopping' of certificates normally takes place when obtaining or creating a certificate as discussed in the Section called *Certificate Configuration* in Chapter 2.

A number of steps need to be taken to ensure that encryption will take place. These include the following

1. The unrestricted Java policy files need to be installed to ensure unrestricted algorithm strength used for encryption. This can be achieved by copying the file `local_policy.jar` from the installation directory to the `$JAVA_HOME/lib/security/` directory. The default policy file enforces a keystrength unsuitable for encryption with FMS.
2. A mapping of recipient DUNS number to Keystore alias needs to be assigned for encryption purposes. This is achieved by creating a PartnerIdentifier section by providing focus to the Partner Identifier list and clicking the `Add` button. Complete the required information and click `Save` to finalise the changes. Refer to the Section called *Partner Identifier Configuration*.

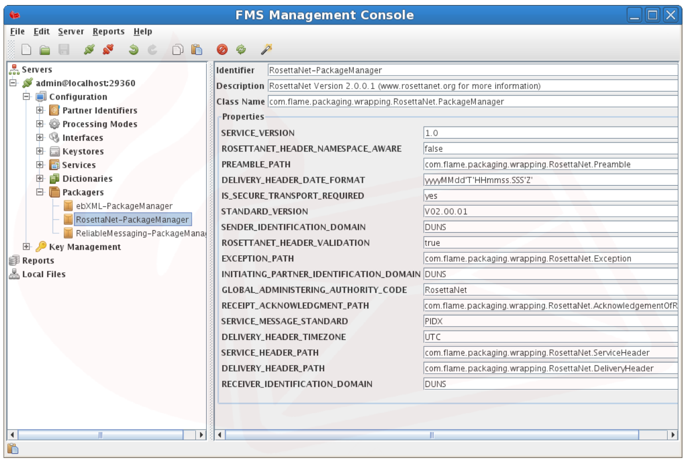
Package Manager Configuration

Additional messaging specifications in the form of Package Managers are supported by FMS. These package managers MUST be configured in the file `ConnectionConfiguration.xml`. The Packager Configuration section contains a simple mapping between a key name and a Java class path denoting the location of the relevant package manager in the classpath. These Package Managers can have their own configuration sections in the `ConnectionConfiguration.xml` file. By default RosettaNet and ebXML packaging specifications are included in FMS, refer to the Section called *RosettaNet Package Configuration* and the Section called *ebXML Package Configuration*.

RosettaNet Package Configuration

The RosettaNet Package Configuration Section is named `RosettaNet-PackageManager`. This section contains RosettaNet® specific options such as `GLOBAL_USAGE_CODE`. `GLOBAL_USAGE_CODE` can be one of two values being `Test` or `Production`.

As well as:



Identifier	RosettaNet-PackageManager
Description	RosettaNet Version 2.0.0.1 (www.rosettanet.org for more information)
Class Name	com.flame.packaging.wrapping.RosettaNet.PackageManager

Properties

SERVICE_VERSION	1.0
ROSETTANET_HEADER_NAMESPACE_AWARE	false
PREAMBLE_PATH	com.flame.packaging.wrapping.RosettaNet.Preamble
DELIVERY_HEADER_DATE_FORMAT	yyyyMMdd'T'HHmmss.SSS'Z'
IS_SECURE_TRANSPORT_REQUIRED	yes
STANDARD_VERSION	V02.00.01
SENDER_IDENTIFICATION_DOMAIN	DUNS
ROSETTANET_HEADER_VALIDATION	true
EXCEPTION_PATH	com.flame.packaging.wrapping.RosettaNet.Exception
INITIATING_PARTNER_IDENTIFICATION_DOMAIN	DUNS
GLOBAL_ADMINISTERING_AUTHORITY_CODE	RosettaNet
RECEIPT_ACKNOWLEDGMENT_PATH	com.flame.packaging.wrapping.RosettaNet.AcknowledgementOfR
SERVICE_MESSAGE_STANDARD	PIDX
DELIVERY_HEADER_TIMEZONE	UTC
SERVICE_HEADER_PATH	com.flame.packaging.wrapping.RosettaNet.ServiceHeader
DELIVERY_HEADER_PATH	com.flame.packaging.wrapping.RosettaNet.DeliveryHeader
RECEIVER_IDENTIFICATION_DOMAIN	DUNS

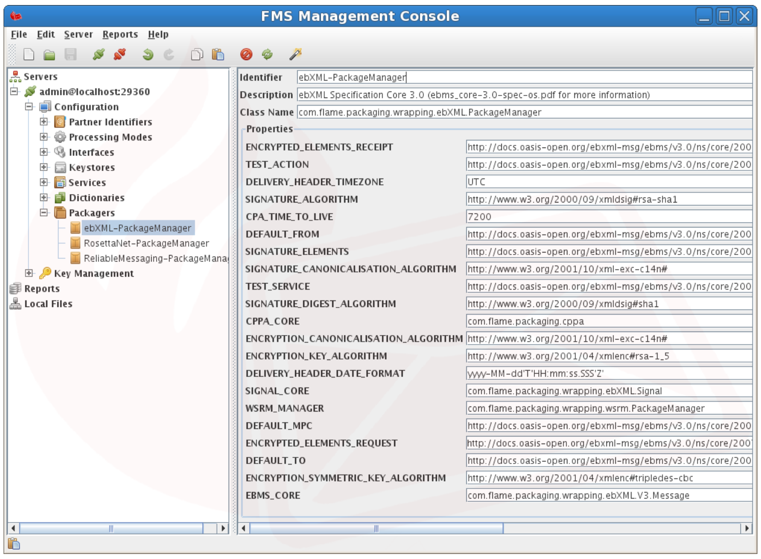
Figure 3-13. RosettaNet Package Configuration example

Description of the RosettaNet Package configuration options:

- *DELIVERY_HEADER_DATE_FORMAT* - The Java based date/time format for use within the packaged RosettaNet DeliveryHeader.
- *DELIVERY_HEADER_PATH* - A Class path to the JAXB created class bindings for the RosettaNet Specification DeliveryHeader Schemas.
- *DELIVERY_HEADER_TIMEZONE* - The Timezone to adjust the current server time to.
- *EXCEPTION_PATH* - A Class path to the JAXB created class bindings for the RosettaNet Specification Exception Schemas.
- *GLOBAL_ADMINISTERING_AUTHORITY_CODE* - Instance from set of codes identifying an administrating authority.
- *INITIATING_PARTNER_IDENTIFICATION_DOMAIN* - The type of Partner Identifier to be used (For example: DUNS)
- *IS_SECURE_TRANSPORT_REQUIRED* - Affirmative (Yes/No) value indicates that the next hub must transmit this message securely.
- *PREAMBLE_PATH* - A Class path to the JAXB created class bindings for the RosettaNet Specification Preamble Schemas.
- *RECEIPT_ACKNOWLEDGMENT_PATH* - A Class path to the JAXB created class bindings for the RosettaNet Specification ReceiptAcknowledgement Schemas.
- *RECEIVER_IDENTIFICATION_DOMAIN* - The type of Partner Identifier to be used (For example: DUNS)
- *ROSETTANET_HEADER_NAMESPACE_AWARE* - Boolean (true/false) value for use during parsing of RosettaNet headers to define XML Namespace awareness.
- *ROSETTANET_HEADER_VALIDATION* - Boolean (true/false) value enabling/disabling RosettaNet header validation.
- *SENDER_IDENTIFICATION_DOMAIN* - The type of Partner Identifier to be used (For example: DUNS)
- *SERVICE_HEADER_PATH* - A Class path to the JAXB created class bindings for the RosettaNet Specification ServiceHeader Schemas.
- *SERVICE_MESSAGE_STANDARD* - The standard with which the Service Content MUST be compliant.
- *SERVICE_VERSION* - The version of the standard with which the Service Content MUST be compliant.
- *STANDARD_VERSION* - Identifies the version number of the standard (RNIF Version).

ebXML Package Configuration

The ebXML Package Configuration Section is named `ebXML-PackageManager`.



Identifier	ebXML-PackageManager
Description	ebXML Specification Core 3.0 (ebms_core-3.0-spec-os.pdf for more information)
Class Name	com.flame.packaging.wrapping.ebXML.PackageManager

Properties

ENCRYPTED_ELEMENTS_RECEIPT	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
TEST_ACTION	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
DELIVERY_HEADER_TIMEZONE	UTC
SIGNATURE_ALGORITHM	http://www.w3.org/2000/09/xmldsig#rsa-sha1
CPA_TIME_TO_LIVE	7200
DEFAULT_FROM	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
SIGNATURE_ELEMENTS	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
SIGNATURE_CANONICALISATION_ALGORITHM	http://www.w3.org/2001/10/xml-exc-c14n#
TEST_SERVICE	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
SIGNATURE_DIGEST_ALGORITHM	http://www.w3.org/2000/09/xmldsig#sha1
CPPA_CORE	com.flame.packaging.cppa
ENCRYPTION_CANONICALISATION_ALGORITHM	http://www.w3.org/2001/10/xml-exc-c14n#
ENCRYPTION_KEY_ALGORITHM	http://www.w3.org/2001/04/xmlenc#rsa-1_5
DELIVERY_HEADER_DATE_FORMAT	yyyy-MM-dd'T'HH:mm:ss.SSS'Z'
SIGNAL_CORE	com.flame.packaging.wrapping.ebXML.Signal
WSRM_MANAGER	com.flame.packaging.wrapping.wsrm.PackageManager
DEFAULT_MPC	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
ENCRYPTED_ELEMENTS_REQUEST	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
DEFAULT_TO	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200
ENCRYPTION_SYMMETRIC_KEY_ALGORITHM	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
EBMS_CORE	com.flame.packaging.wrapping.ebXML.V3.Message

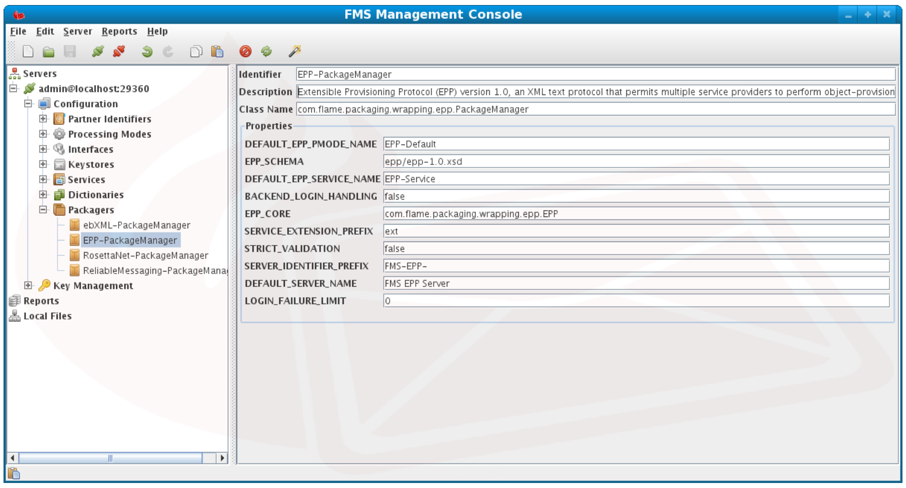
Figure 3-14. ebXML Package Configuration example

Description of the ebMS Package configuration options:

- *CPA_TIME_TO_LIVE* - The lifetime in seconds that the Collaboration Protocol Agreement should live for before an updated copy is retrieved from the provided URL.
- *CPPA_CORE* - A Class path to the JAXB created class bindings for the CPP/A Specification Schemas.
- *DELIVERY_HEADER_DATE_FORMAT* - The Java based date/time format for use within the packaged ebXML SOAP envelope.
- *DELIVERY_HEADER_TIMEZONE* - The Timezone to adjust the current server time to.
- *EBMS_CORE* - A Class path to the JAXB created class bindings for the ebXML V3 Specification Schemas.
- *ENCRYPTED_ELEMENTS* - A comma delimited list of elements appearing in the Request SOAP header that should be encrypted.
- *ENCRYPTION_CANONICALISATION_ALGORITHM* - Web Services Security specific setting.
- *ENCRYPTION_KEY_ALGORITHM* - Web Services Security specific setting.
- *ENCRYPTION_SYMMETRIC_KEY_ALGORITHM* - Web Services Security specific setting.
- *RECEIPT_ENCRYPTED_ELEMENT* - A comma delimited list of elements appearing in the Receipt SOAP header that should be encrypted.
- *SIGNAL_CORE* - A Class path to the JAXB created class bindings for the ebBP Signal Specification Schema.
- *SIGNATURE_ALGORITHM* - Web Services Security specific setting.
- *SIGNATURE_CANONICALISATION_ALGORITHM* - Web Services Security specific setting.

EPP Package Configuration

The EPP Package Configuration Section is named `EPP-PackageManager`.



Identifier	EPP-PackageManager
Description	Extensible Provisioning Protocol (EPP) version 1.0, an XML text protocol that permits multiple service providers to perform object-provision
Class Name	com.flame.packaging.wrapping.epp.PackageManager

Properties

DEFAULT_EPP_PMODE_NAME	EPP-Default
EPP_SCHEMA	epp/epp-1.0.xsd
DEFAULT_EPP_SERVICE_NAME	EPP-Service
BACKEND_LOGIN_HANDLING	false
EPP_CORE	com.flame.packaging.wrapping.epp.EPP
SERVICE_EXTENSION_PREFIX	ext
STRICT_VALIDATION	false
SERVER_IDENTIFIER_PREFIX	FMS-EPP-
DEFAULT_SERVER_NAME	FMS EPP Server
LOGIN_FAILURE_LIMIT	0

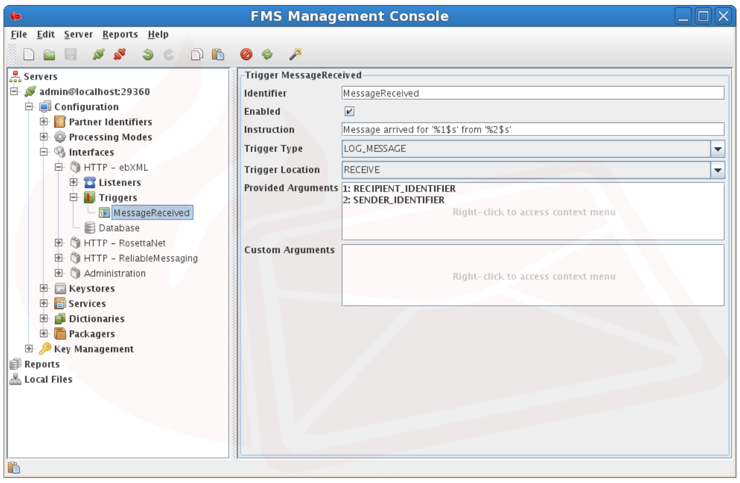
Figure 3-15. EPP Package Configuration example

Description of the EPP Package configuration options:

- *DEFAULT_EPP_PMODE_NAME* - The provided default Processing Mode identifier that the EPP specification should use when utilizing backend authenticaton of client identifiers. The Processing Mode contains default values required for message storage and transmission.
- *EPP_SCHEMA* - The path to the EPP XML Schema, this path is prefixed by the SCHEMA_DIRECTORY system variable.
- *DEFAULT_EPP_SERVICE_NAME* - The provided default Service that will be used when generating EPP Greetings, this service may be extended or customized according to the Listener used.
- *BACKEND_LOGIN_HANDLING* - When set to 'true' a trigger will be invoked to perform backend login authentication. This trigger will be invoked with a METHOD argument (set to 'login') in the Custom Arguments of the specified trigger. Note that when using an Embedded Trigger the METHOD argument is set as a variable.
- *EPP_CORE* - A Class path to the JAXB created class bindings for the EPP Schemas.
- *SERVICE_EXTENSION_PREFIX* - A string prefix used to differentiate internal from external Commands and/or Objects within the EPP Specification. When the Service associated Dictionary Action Code has this prefix (including a semi-colon ':', eg: '`ext:secDNS1.1`') it is assumed to be an extension to EPP and treated accordingly.
- *STRICT_VALIDATION* - This instructs the XML parser and validation system to perform a strict validation of the supplied EPP XML. The file system siblings of the above EPP_SCHEMA are located and used to validate the XML, any web references to imported Schemas will be resolved and cached so an initial load time is expected once per server instance.
- *SERVER_IDENTIFIER_PREFIX* - A string prefix used to customize the server generated transaction identifiers when receiving an optional client transaction identifier.
- *DEFAULT_SERVER_NAME* - The server name to be used when generating the EPP Greeting.
- *LOGIN_FAILURE_LIMIT* - Sets the maximum number of times a login can fail before being disconnected. 0 indicates unlimited failures.

Triggers

Triggers provide a powerful interface into the business environment through the invocation of customizable instructions.



Trigger MessageReceived

Identifier: MessageReceived
Enabled: ✓
Instruction: Message arrived for '%1$s' from '%2$s'
Trigger Type: LOG_MESSAGE
Trigger Location: RECEIVE
Provided Arguments: 1: RECIPIENT_IDENTIFIER
2: SENDER_IDENTIFIER
Custom Arguments:

Figure 3-16. Triggers

Instruction

The Trigger Instruction field contains either the `LOG_MESSAGE` or the executable/script path to be invoked, as well as arguments to be passed to the invoked executable/script.

Trigger instructions are based on Java Formatted Strings. The following link provides more information on formatted strings.

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html#summary

Trigger Types

Triggers are available in 3 types {LOG, EXEC, SYNC_EXEC}

- *LOG* - The instruction value is displayed as a log message in the logging system with the chosen arguments included when formatting.
- *EXEC* - The instruction value is considered a system executable and will be invoked as such.
- *SYNC_EXEC* - Differs from EXEC in the way it handles STDOUT. It is assumed, when setting SYNC_EXEC, that the executable will return a valid XML Object which is considered a synchronous return if the associated P-Mode has an MEP Binding of type sync.

It is critical to note the following when creating EXEC and SYNC_EXEC triggers:

1. Any system executable that is in the environment PATH is invokable.
2. The trigger executable MUST either be in the PATH or MUST be fully qualified. The executable must have execute mode set on UNIX systems.
3. The executable MUST return in a reasonable amount of time
4. The executable MAY return error messages to STDERR which will be displayed if the return value is greater than 0.

Executable Type

When Trigger Type `EXEC` or `SYNC_EXEC` is selected then an `Executable Type` should be selected. `EXTERNAL` is used by default if this argument is not provided.

- *JEP* - Instructs the Trigger mechanisms to utilize the built in JEP (Java Embedded Python) Script Engine. JEP is a JNI implementation of CPython, allowing a native interface to all CPython libraries.
- *JYTHON* - Instructs the Trigger mechanisms to utilize the built in `Jython` (Java Python) Script Engine. `Jython` is a Java implementation of Python, allows for a simple interface to Python but is limited to the core Python mechanisms. Utilizing third party libraries and imports is restricted when using `Jython`
- *EXTERNAL* - Instructs the Trigger mechanisms to invoke a system process enabling maximum compatibility however resulting in a high performance overhead while creating the process.

Further Script Engines will be available in future releases, including but not limited to [Ruby, PHP, JavaScript, AWK, etc]

Trigger Points

Various points exist at which an assigned trigger will be invoked. These are identified as follows

- *SEND*. Triggers when a message is sent from the server or via a synchronous return as provided by a SYNC_EXEC trigger.
- *RECEIVE*. Triggers when a message is received on the server via a Pull Request or an Incoming Request.
- *RESPONSE_SENT*. This trigger fires after writing final signal response (Receipt or Error) for an incoming user message. It may be used for implementing twoWay MEPs by sending response for the 2nd leg of the twoWay MEP by using the `messageID` argument as the `refToMessageID` using the `'-r'` command line argument to the FMS AS4 Light Client. The Configuration Trigger arguments are as follows

```
<tg:Trigger>
  <tg:identifier>SendResponse</tg:identifier>
  <tg:enabled>true</tg:enabled>
  <tg:type>SynchronousExecutable</tg:type>
  <tg:location>RESPONSE_SENT</tg:location>
  <tg:instruction>run/trigger/Response.sh %1$s %2$s %3$s %4$s %5$s %6$s %7$s \
   %8$s %9$s %10$s %11$s %12$s %13$s</tg:instruction>

  <!-- messageID of the receipt or error return else nothing -->
  <tg:providedArguments>messageID</tg:providedArguments>
  <tg:providedArguments>service</tg:providedArguments>
  <tg:providedArguments>action</tg:providedArguments>
  <tg:providedArguments>processingMode</tg:providedArguments>
  <tg:providedArguments>event</tg:providedArguments>
  <tg:providedArguments>senderIdentifier</tg:providedArguments>
  <tg:providedArguments>recipientIdentifier</tg:providedArguments>
  <tg:providedArguments>replyMessageID</tg:providedArguments>
  <tg:providedArguments>conversationID</tg:providedArguments>
  <tg:providedArguments>agreementRef</tg:providedArguments>
  <tg:providedArguments>senderRole</tg:providedArguments>
  <tg:providedArguments>recipientRole</tg:providedArguments>
  <tg:providedArguments>submitted</tg:providedArguments> <!-- signal timestamp else nothing -->
  <tg:providedArguments>payload</tg:providedArguments> <!-- response signal if available else "null" -->
  <tg:executionType>External</tg:executionType>
</tg:Trigger>
```

- *PULL_REQUEST*. Triggers when a message is placed in an inbox for a specified MPC.
- *PULL_REQUEST_EXPIRED*. Triggers if a Pull Request life time has expired at the time of collection. The life time is defined by the retry interval multiplied by the amount of retries as defined in the configuration.

- *ACK*. Triggers when an Acknowledgement is returned to the server either as a response (synchronous) or a callback (asynchronous).
- *NACK*. Triggers when an Error is returned to the server either as a response (synchronous) or a callback (asynchronous).

Arguments

The system provides a list of arguments that could be available when a trigger is invoked. Certain arguments are only available at specific Trigger Points, if a specified argument is not available at the Trigger Point then a list of valid arguments will be logged.

In addition to the system provided arguments it is possible to specify custom arguments. These custom arguments are matched against the MessageProperties of a provided ProcessingMode Event.

When an embedded trigger type is selected such as JYTHON, JEP, or JSON-RPC the arguments (both provided and custom) will be set as environment variables within the embedded system.

Arguments may be referenced using the instruction field. The instruction field is interpreted using Java Formatted Strings and the supplied arguments. Arguments are referenced using the following string `%1$s` where the `1$` is the argument number (seen next to each argument) and the `s` denotes the value is a string.

The following link provides more information on formatted strings:

https://docs.oracle.com/javase/1.5.0/docs/api/java/util/Formatter.html#summary

FMS Logging Configuration

FMS provides extensive logging functionality using both the standard java logging subsystem and the highly flexible Log4j logging subsystem. These logging systems can be adjusted as described in appendix ...

Log4j Configuration

The log4j logging facility requires configuration before it can be used effectively. The configuration files and associated configuration options are as below

- `log4j.properties.logfile` - This file contains the default settings for FMS when logging to a rolling log file. Copy this file to `log4j.properties` in order to send logging information to the file associated with the `log4j.appender.dest2.File` entry.
- `log4j.properties.syslog` - This file contains the default settings for FMS when logging to the system logger. Copy this file to `log4j.properties` in order to send logging information to the system logger associated with the `log4j.appender.Syslog.Facility` entry.
- `log4j.properties.mail` - This file contains the default settings for FMS when logging to SMTP. Copy this file to `log4j.properties` in order to send logging information to the email address associated with the `log4j.appender.mail.To` entry.
- `log4j.properties.html` - This file contains the default settings for FMS when logging to a HTML page. Copy this file to `log4j.properties` in order to send logging information to a HTML page associated with the `log4j.appender.html.File` entry.

Note: Logging in any level below `WARN` is intensive on the system and might slow down message transmission times. For decreased transmission times set the Log Level to either `WARN` or higher (`ERROR`) in a production environment.

The default `log4j.properties` file typically exists in the same directory that the FMS program files are installed. This may be changed by adding `-Dlog4j.configuration=file:/path/to/log4j.properties` to the `java` command line as follows

```
exec java -Dlog4j.configuration=file:///path/to/log4j.properties -server -cp ${FMSLIBDIR} \
  -jar ${FMSLIBDIR}/fms.jar $* 2>> ${FMSLOGDIR}/err.log >> ${FMSLOGDIR}/out.log
```

Further details for the log4j logging facility and configuration are available at http://logging.apache.org/log4j/docs/ and https://www.tutorialspoint.com/log4j/index.htm.

Rotating log4j Files

The log4j log files may be rotated using the configuration directly or alternatively using the `logrotate` facility of the operating system as per the following example typically in `/etc/logrotate.d/fms` for *nix based operating systems

```
# Rotate script for fms
/usr/local/fms/log/fms.log {
  size 10M
  missingok
  rotate 100
  create 644 fms fms
  compress
  sharedscripts
}

/usr/local/fms/log/error.log {
  size 10M
  missingok
  rotate 100
  dateext
  create 644 fms fms
  compress
  sharedscripts
}

/usr/local/fms/log/out.log {
  daily
  missingok
  notifempty
  rotate 356
  dateext
  create 644 fms fms
  compress
  sharedscripts
}

/usr/local/fms/log/err.log {
  daily
  missingok
  notifempty
  rotate 356
  dateext
  create 644 fms fms
  compress
  sharedscripts
}
```

Syslog Configuration

FMS uses the `rsyslogd` logging facility on `linux` operating systems. Other operating systems like `*nix` and `MAC OS X` may use `syslog` for logging purposes which follow a similar approach described here. The `Windows` operating system uses it's own logging framework which is not documented here.

The `rsyslog` facility requires configuration before it can be used. Below are a number of suggested steps to perform for `rsyslog` and `log4j.properties`:

1. By default messages from FMS are logged to files in the `/var/log/fms/` directory on *nix systems. If a different syslog facility, as defined in `log4j.properties`, is required, for example `LOCAL0` then the system logger configuration file, typically `/etc/rsyslog.conf` for Redhat systems or `/etc/rsyslog.d/50-default.conf` for Ubuntu systems , needs to include an entry as follows:

```
# User log files
local0.*                         /var/log/fms/fms-local.log
```

and the configuration `log4j.properties` file, with the `log4j.appender.Syslog.Facility=LOCAL0` definition, typically as follows

```
log4j.rootCategory=TRACE, Syslog
log4j.appender.Syslog=org.productivity.java.syslog4j.impl.log4j.Syslog4jAppender
# The following entry requires an entry as follows in /etc/rsyslog.d/50-default.conf. Ensure fms-local.log has write pe
log4j.appender.Syslog.Facility=LOCAL0
log4j.appender.Syslog.Protocol=unix_syslog
log4j.appender.Syslog.Threshold=TRACE
log4j.appender.Syslog.layout=org.apache.log4j.PatternLayout
log4j.appender.Syslog.layout.ConversionPattern=%d{ISO8601} %-5p SID:%X{sid} ID:%X{msgid} FROM:%X{from} TO:%X{to} %X{cmd
```

2. Alternatively log messages from FMS may be sent to a remote syslog server by using the following `log4j.properties` configuration.

```
log4j.rootCategory=TRACE, Syslog
log4j.appender.Syslog=org.apache.log4j.net.SyslogAppender
log4j.appender.Syslog.Threshold=DEBUG
log4j.appender.Syslog.layout=org.apache.log4j.PatternLayout
log4j.appender.Syslog.layout.ConversionPattern=fms.flame.business %-5p SID:%X{sid} ID:%X{msgid} FROM:%X{from} TO:%X{to}
log4j.appender.Syslog.SyslogHost=syslog.flame.business:6514
```

3. Restart FMS and reload the system logger in order for the changes to be in effect. Also ensure that permissons permit writing to file `fms-local.log` by the `rsyslog` process of logging locally. Note that the facility (`LOCAL0`) is case insensitive.

Altering Log4j Configuration Files

Changing Logging Level

Edit the file `log4j.properties` and adjust the `log4j.rootCategory=` value. log4j logging levels are described in the Section called *Levels of Granularity* in Appendix C.

Note that when altering these configuration files, the server MUST be restarted in order for any changes to take effect. Information on the server can be located in the Section called *FMS Installation* in Chapter 2.

Chapter 4. AS4 Client Utility

FMS is the AS4 light client used for sending messages (push) or retreiving messages (pull) from a FMS or other AS4 compliant server situated at remote business partner.

AS4 Light Client Invocation

Invoke the FMS AS4 push and pull client utility as follows to display usage requirements on a Linux system

```
java -jar fmsclient.as4.jar -?
```

or as follows on MS Windows®

```
\Program Files\fms\client\fms-as4.exe -?

  Usage: java -jar fmsclient.as4.jar <arguments> <XML_File>
  Where: <XML_File> is the optional XML Document to include in the SOAP body
  And <arguments> are:
-a <Attachment>: Repeatable attachment references in the following generic format \
          "key1:value1;key2:value2;key3:value3[;key4:value4]" with example being \
          "filename:mypayload.xml;content-id:ABC123;mimetype:application/xml;encoding:\
          [7bit|quoted-printable|8bit|base64|binary];description:My Payload file;\
          propertyKey:propertyValue" \
          where filename is mandatory and specifies either a local file or URL based file, \
          and the rest optional. Semicolons ";" MUST not be used as part of the keys or values. \
          Use this should an xml payload be required outside the body in the SOAP envelope
-action <Action>: BusinessInfo.Action, overrides the processing mode businessInfo.Action
-ad <Attachment Directory>: Directory in which attachments received will be stored. \
          Default: attachments
-ag <Agreement Reference>: AgreementRef, overrides the processing mode agreementRef
-at <Send Time>: Send message at current time + time in milliseconds or at specified \
          future date in format "yyyy-MM-dd'T'HH:mm:ss.SSS" eg. 2100-12-31T00:00:00.000
-c <Conversation ID>: Conversation ID, overrides the processing mode conversation ID
-batch <Argument File>: Contains a set of arguments for use in batch processing. \
          Arguments in this file will override any argument set on the command line
-bodyID: Enable the SOAP Body PayloadInfo href identifier
-d: Enable DEBUG mode. Default: false
-e <Event>: Processing Mode event for this message
-enc <Encryption Alias>: Alias used to access the remote partner public encryption key \
          from the keystore
-error <Error File>: File to which any error messages and debug output will be written \
          to - default is stderr
-file <Output File>: File to which any received response will be written to - default \
          is stdout
-from <Sender>: Sender identifier to determine which leg of the journey this message \
          is for. Required.
-fromType <Sender@Type>: Sender type attribute - only used if from party is different \
          between the command line and the p-mode
-fromRole <Sender Role>: Sender role - only used if from party is different between \
          the command line and the p-mode
        -h <Host>: Destination host URL. Note that IPv6 addresses must be encapsulated in \
          square brackets Eg. -h https://[fe80:8::106a:9125:18a9:9f64%en0]:6443/as4s
-k <KeyStore Location>: File system path to the KeyStore. Default: certs
-ksp <KeyStore Password>: Password used to access the KeyStore. Default: changeit
-ka <SSL Private Key Alias>: Alias used to access the SSL Private Key in the KeyStore. \
          Default: fmsrns
-kp <SSL Private Key Password>: Password used to access the SSL Private Key in the \
          KeyStore. Default: fmsrns
-l <Licence file>: Path to the licence file. Default: fms.lcn
-m <Message Identifier>: Message identifier, leave blank to auto-generate
-mp <Message Properties>: Optional message properties with format \
          "-mp key1:value1 -mp key2:value2". Eg \
          "-mp originalSender:partyA -mp finalRecipient:partyB"
-messageFile <Message Output File>: File to save outbound message SOAP envelope if specified
-noSystemExit: Disable system exit. If set then System.exit(status) will not be invoked \
          on program termination and return status will be returned to stdout. Default: disabled
-pi <PMode ID>: Processing Mode ID, overrides the processing mode ID
-pm <Processing Mode>: Processing mode configuration file name. Required.
-pp <Part Properties>: Optional repeatable part properties for the main XML payload in \
          the body (if specified) with format "-pp key1:value1 -pp key2:value2". \
          Eg "-pp invoiceID:INV0001 -pp http\://mydomain.com\:invoiceID:INV\:9999"
-r <Reference Message Identifier>: Reference to the message identifier provided
-rc <Receiving Security Context File>: Receiving security context definition \
          file (http://java.sun.com/xml/ns/xwss/config)
-rs: Enable response security requirement, will validate security response and \
          will throw errors if security does not match requirement
-rcpt <In Receipt to SOAP Message>: Create an AS4 receipt for the provided SOAP Message
-s <SSL Context>: Typically one of [SSL, SSLv3, TLS, TLSv1, TLSv1.1, TLSv1.2, TLSv1.3]. Default: TLSv1.2
-sa <Sign Key Alias>: Alias used to access the Private Key for signing in the KeyStore. \
          Default: fmsrns
-service <Service>: BusinessInfo.Service, overrides the processing mode businessInfo.Service
-serviceType <Service@Type>: BusinessInfo.Service@Type, overrides the processing mode \
          businessInfo.Service@Type attribute
-sc <Security Context File>: Sending security context definition file
-scit: Enable Security Context includeTimestamp Flag. Default: false
-suffix: MessageId Suffix. Default: initiating party or localhost
-sp <Sign Key Password>: Password used to access the Private Key for signing in the \
         KeyStore. Default: fmsrns
-st <sslTruststore Location>: Optional file system path to the sslTruststore. \
         May be the same as keystore. Default: java cacerts
-stsp <sslTruststore Password>: Optional password used to access the ssl Trust Store. \
         Default: changeit
-t <Timeout>: Connection timeout in milliseconds (0 disables timeouts). Default: 30000
-T <Timeout>: Response timeout in milliseconds (0 disables timeouts). Default: 30000
-to <Recipient>: Recipient identifier to determine which leg of the journey this \
         message is for. Required.
-toType <Recipient@Type>: Recipient type attribute - only used if to party is \
         different between the command line and the p-mode
-toRole <Recipient Role>: Recipient role - only used if to party is different \
         between the command line and the p-mode
-u: Disable SSL. Default: enabled
-v: Version
        -z: Optional command line option for compressing external payloads. If not set then the processing mode \
         "pmode:useCompression" setting will be used to determine if external payloads must be compressed.
-X: Dump security providers and algorithms
```

AS4 Light Client Examples

The following example illustrates invoking the AS4 light client and pushing a puchase order and associated `README` external payload to a remote server.

```
java -jar fmsclient.as4.jar -h http://remotehost.com:8080/as4 -sc sc.xml -e send -pm as4push.pmode \
 -from POService.flame.business -to POService.flame.business \
 -enc flameserver \
 -a 'filename:README;content-id:id-29996-2011-10-07-15:45:38;mimetype:text/plain; \
 description:Compressed README' ProcessPurchaseOrder.xml
```

A successful message push should return a receipt as follows

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <eb:Messaging xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:Timestamp>2011-10-07T13:45:46.144Z</eb:Timestamp>
          <eb:MessageId>FMS-A-20111007-154542.442-0.5731733724663212@POService.flame.business< \
  /eb:MessageId>
          <eb:RefToMessageId>AS4-132DEA1D1BB-6990F@POService.flame.business</eb:RefToMessageId>
        </eb:MessageInfo>
        <eb:Receipt>
          <ebbpsig:NonRepudiationInformation xmlns:ebbpsig= \
  "http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0">
            <ebbpsig:MessagePartNRInformation>
              <ebbpsig:MessagePartIdentifier/>
            </ebbpsig:MessagePartNRInformation>
          </ebbpsig:NonRepudiationInformation>
        </eb:Receipt>
      </eb:SignalMessage>
    </eb:Messaging>
  </env:Header>
  <env:Body/>
</env:Envelope>
```

The following example illustrates invoking the AS4 client and pulling an acknowledge puchase order from a remote server.

```
java -jar fmsclient.as4.jar -h http://remotehost.com:8080/as4 -e pull -from POService.flame.business \
  -to POService.flame.business -pm as4pull.pmode
```

FMS AS4 Light Client Push PMode

The following example provides a typical p-mode that is used by the send example above. Various p-mode settings are overriden by the command line including both initiating and responding parties.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pmode:ProcessingMode xmlns:tg="http://fms.flame.business/FMS/schema/Trigger"
 xmlns:pmode="http://fms.flame.business/FMS/schema/ProcessingMode">
  <!-- Copyright (c) Flame Computing Enterprises cc 2007 - 2019. All rights reserved -->
  <!-- RCSfile: as4push.pmode,v Revision: 1.1.2.4 Date: 2019-02-14 10:17:34 -->
  <pmode:specification>AS4</pmode:specification>

  <pmode:general>

    <!-- overrride Agreement using -ag <agreementRef> commandLine option -->
    <pmode:Agreement></pmode:Agreement>

    <!-- overrride ConversationID using '-c <conversation id>' commandLine option -->
    <pmode:ConversationID>0001</pmode:ConversationID>
    <pmode:MEP>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay</pmode:MEP>
    <pmode:MEPbinding>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push</pmode:MEPbinding>

      <pmode:initiating>

      <!-- overrride initiating party using '-from <party>' commandLine option -->
      <pmode:party>undefined-initiating</pmode:party>

      <!-- override initiating role using '-fromRole <role>' commandLine option -->
      <pmode:role>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/initiator</pmode:role>
    </pmode:initiating>

    <pmode:responding>

      <!-- overrride responding party using -to <party> commandLine option -->
      <pmode:party>undefined-responding</pmode:party>

      <!-- override responding role using '-toRole <role>' commandLine option -->
      <pmode:role>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/responder</pmode:role>
    </pmode:responding>

  </pmode:general>

  <pmode:event pmode:ID="send">
    <pmode:protocol>

      <!-- overrride address using '-h <remoteHost url>' commandLine option -->
      <pmode:address>http://localhost:8080/as4</pmode:address>
      <pmode:SOAPVersion>1.2</pmode:SOAPVersion>

      <!-- overrride useCompression using '-z' commandLine option -->
      <pmode:useCompression>true</pmode:useCompression>
      <pmode:retryThreshold>3</pmode:retryThreshold>
      <pmode:retryInterval>60</pmode:retryInterval>
```

```
</pmode:protocol>
<pmode:businessInfo>
  <pmode:service>http://docs.oasis-open.org/ebxml-msg/as4/200902/service</pmode:service>
  <pmode:action>http://docs.oasis-open.org/ebxml-msg/as4/200902/action</pmode:action>
  <pmode:maxSize>0</pmode:maxSize>
  <pmode:MPC></pmode:MPC>
  <pmode:MessageProperty>
    <pmode:name>originalSender</pmode:name>
    <pmode:description>original sender</pmode:description>
    <pmode:datatype>string</pmode:datatype>
    <pmode:usage>optional</pmode:usage>
  </pmode:MessageProperty>
  <pmode:MessageProperty>
    <pmode:name>finalRecipient</pmode:name>
    <pmode:description>final recipient</pmode:description>
    <pmode:datatype>string</pmode:datatype>
    <pmode:usage>optional</pmode:usage>
  </pmode:MessageProperty>
  <pmode:PayloadProfile>
    <pmode:ContentID>manifest</pmode:ContentID>
    <pmode:mimeType>application/xml</pmode:mimeType>
    <pmode:schemaFile></pmode:schemaFile>
    <pmode:maxSize>0</pmode:maxSize>
    <pmode:usage>optional</pmode:usage>
  </pmode:PayloadProfile>
</pmode:businessInfo>
<pmode:security>
  <pmode:WSSVersion>1.1</pmode:WSSVersion>

  <!-- defined by -sc sc.xml security context file -->
  <pmode:SOAPSecurityLevel>SIGN_ENCRYPT</pmode:SOAPSecurityLevel>

  <!-- defined by -sc sc.xml security context file -->
  <pmode:MIMESecurityLevel>SIGN_ENCRYPT</pmode:MIMESecurityLevel>
  <pmode:X509>

  <!-- signature settings overriden by the security context file using
   '-sc <securityContext.xml>' -->
    <pmode:signature>

      <!-- CanonicalizationMethod - defaults to  http://www.w3.org/2001/10/xml-exc-c14n# -->
      <!-- defined by server property SIGNATURE_CANONICALISATION_ALGORITHM -->
      <!-- SignatureMethod - defined by server property SIGNATURE_ALGORITHM -->
      <pmode:algorithm>http://www.w3.org/2001/04/xmldsig-more#rsa-sha256</pmode:algorithm>

      <!-- DigestMethod as used for all targets - defined by
       server property SIGNATURE_DIGEST_ALGORITHM -->
      <pmode:hashFunction>http://www.w3.org/2001/04/xmlenc#sha256</pmode:hashFunction>
      <pmode:CryptPart
       pmode:namespace="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
        <pmode:element>Messaging</pmode:element>
      </pmode:CryptPart>
      <pmode:CryptPart pmode:namespace="http://www.w3.org/2003/05/soap-envelope">
        <pmode:element>Body</pmode:element>
      </pmode:CryptPart>
      <pmode:CryptPart>

      <!-- transform algorithm defaults to
       http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform -->
        <pmode:element>cid:*</pmode:element>
      </pmode:CryptPart>
    </pmode:signature>

    <!-- encryption settings overriden by the security context file using
     '-sc <securityContext.xml>' -->
    <pmode:encryption>

      <!-- defined by server property ENCRYPTION_SYMMETRIC_KEY_ALGORITHM -->
      <pmode:algorithm>http://www.w3.org/2009/xmlenc11#aes128-gcm</pmode:algorithm>
      <pmode:certificate></pmode:certificate>
```

```
          <!-- <pmode:minimumStrength>256</pmode:minimumStrength>
           supported but not used here -->
         <pmode:CryptPart
          pmode:namespace="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
            <pmode:element>Messaging</pmode:element>
          </pmode:CryptPart>

          <pmode:CryptPart pmode:namespace="http://www.w3.org/2003/05/soap-envelope">
            <pmode:element>Body</pmode:element>
          </pmode:CryptPart>

          <pmode:CryptPart>
          <!-- transform algorithm defaults to
           http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform -->
            <pmode:element>cid:*</pmode:element>

          </pmode:CryptPart>
        </pmode:encryption>
      </pmode:X509>
      <pmode:usernameToken>
        <pmode:PModeAuthorize>false</pmode:PModeAuthorize>
        <pmode:username></pmode:username>
        <pmode:password></pmode:password>
        <pmode:digest>false</pmode:digest>
        <pmode:nonce>false</pmode:nonce>
        <pmode:created>false</pmode:created>
      </pmode:usernameToken>
      <pmode:sendReceipt> <!-- defined on server -->
        <pmode:enabled>true</pmode:enabled>
        <pmode:replyPattern>response</pmode:replyPattern>
        <pmode:includeUsernameToken>false</pmode:includeUsernameToken>
      </pmode:sendReceipt>
    </pmode:security>
  </pmode:event>
</pmode:ProcessingMode>
```

FMS AS4 Light Client Pull PMode

The following example provides a typical p-mode that is used by the pull example above. Various p-mode settings are overriden by the command line including both initiating and responding parties.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pmode:ProcessingMode xmlns:tg="http://fms.flame.business/FMS/schema/Trigger"
 xmlns:pmode="http://fms.flame.business/FMS/schema/ProcessingMode">
  <!-- Copyright (c) Flame Computing Enterprises cc 2007 - 2019. All rights reserved -->
  <!-- RCSfile: as4pull.pmode,v Revision: 1.1.2.2 Date: 2019-02-14 10:17:34 -->
  <pmode:specification>AS4</pmode:specification>

  <pmode:general>
    <!-- overrride Agreement using '-ag <agreementRef>' commandLine option -->
    <pmode:Agreement></pmode:Agreement>

    <!-- overrride ConversationID using '-c <conversation id>' commandLine option -->
    <pmode:ConversationID></pmode:ConversationID>
    <pmode:MEP>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay</pmode:MEP>
    <pmode:MEPbinding>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pull</pmode:MEPbinding>

    <pmode:initiating>

      <!-- override initiating party using -from <party> commandLine option -->
      <pmode:party>undefined-initiating</pmode:party>
      <!-- override initiating role using -fromRole <party> commandLine option -->
      <pmode:role>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/initiator</pmode:role>
    </pmode:initiating>

    <pmode:responding>

      <!-- override responding party using '-to <party>' commandLine option -->
      <pmode:party>undefined-responding</pmode:party>
      <!-- override responding role using -toRole <party> commandLine option -->
      <pmode:role>http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/responder</pmode:role>
    </pmode:responding>

  </pmode:general>
  <pmode:event pmode:ID="pull">
    <pmode:protocol>

      <!-- overrride address using '-h <remoteHost url>' commandLine option -->
      <pmode:address>https://fmstest.flame.business:8443/AS4-C3</pmode:address>
      <pmode:SOAPVersion>1.2</pmode:SOAPVersion>
      <pmode:useCompression>true</pmode:useCompression>
      <pmode:retryThreshold>3</pmode:retryThreshold>
      <pmode:retryInterval>60</pmode:retryInterval>
    </pmode:protocol>

    <pmode:businessInfo>
      <pmode:service pmode:type="cenbii-procid-ubl">urn:www.cenbii.eu:profile:bii54:ver3.0</pmode:service>
      <pmode:action>pull-action</pmode:action>
      <pmode:maxSize>0</pmode:maxSize>
      <pmode:MPC>Flame.Invoice.MPC</pmode:MPC>
    </pmode:businessInfo>

    <pmode:security>
      <pmode:WSSVersion>1.1</pmode:WSSVersion>
      <pmode:SOAPSecurityLevel>SIGN_ENCRYPT</pmode:SOAPSecurityLevel>
      <pmode:MIMESecurityLevel>SIGN_ENCRYPT</pmode:MIMESecurityLevel>
      <pmode:X509>
        <pmode:signature> <!-- overriden by the security context file using '-sc <securityContext.xml>' -->
          <!-- CanonicalizationMethod - defaults to  http://www.w3.org/2001/10/xml-exc-c14n# -->
          <!-- defined by server property SIGNATURE_CANONICALISATION_ALGORITHM -->
          <!-- SignatureMethod - defined by server property SIGNATURE_ALGORITHM -->
          <pmode:algorithm>http://www.w3.org/2001/04/xmldsig-more#rsa-sha256</pmode:algorithm>
          <!-- DigestMethod as used for all targets - defined by server property SIGNATURE_DIGEST_ALGORITHM -->
          <pmode:hashFunction>http://www.w3.org/2001/04/xmlenc#sha256</pmode:hashFunction>
          <pmode:CryptPart pmode:namespace="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
            <pmode:element>Messaging</pmode:element>
          </pmode:CryptPart>
          <pmode:CryptPart pmode:namespace="http://www.w3.org/2003/05/soap-envelope">
            <pmode:element>Messaging</pmode:element>
          </pmode:CryptPart>
        </pmode:signature>

        <pmode:encryption> <!-- overriden by the security context file using '-sc <securityContext.xml>' -->
          <!-- defined by server property ENCRYPTION_SYMMETRIC_KEY_ALGORITHM -->
          <pmode:algorithm>http://www.w3.org/2009/xmlenc11#aes128-gcm</pmode:algorithm>
          <pmode:certificate></pmode:certificate>
          <pmode:CryptPart pmode:namespace="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
            <pmode:element>Messaging</pmode:element>
            <pmode:element>CollaborationInfo</pmode:element>
            <pmode:element>PayloadInfo</pmode:element>
          </pmode:CryptPart>
          <pmode:CryptPart pmode:namespace="http://www.w3.org/2003/05/soap-envelope">
            <pmode:element>Body</pmode:element>
          </pmode:CryptPart>
          <pmode:CryptPart>
            <!-- transform algorithm defaults to
             http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform -->
            <pmode:element>cid:*</pmode:element>
          </pmode:CryptPart>
        </pmode:encryption>

      </pmode:X509>

      <pmode:usernameToken>
        <pmode:PModeAuthorize>true</pmode:PModeAuthorize>
        <pmode:username>flame</pmode:username>
        <pmode:password>secret</pmode:password>
        <pmode:digest>true</pmode:digest>
        <pmode:nonce>true</pmode:nonce>
        <pmode:created>true</pmode:created>
      </pmode:usernameToken>

      <pmode:sendReceipt>
        <pmode:enabled>false</pmode:enabled>
        <pmode:replyPattern>response</pmode:replyPattern>
        <pmode:includeUsernameToken>false</pmode:includeUsernameToken>
      </pmode:sendReceipt>

    </pmode:security>
  </pmode:event>
</pmode:ProcessingMode>
```

FMS AS4 Light Client Push Security Context

The following example provides a typical security context file that may be used by the send example above when no signing and no encryption is required. The various p-mode security settings are overriden by the security context file including the signature and encryption settings.

```
<xwss:SecurityConfiguration dumpMessages='false'
 retainSecurityHeader='true' enableDynamicPolicy='false' xmlns:xwss='http://java.sun.com/xml/ns/xwss/config'>
<!-- RCSfile: sc.xml,v Revision: 1.1.2.3 Date: 2019-02-13 06:37:50 -->
  <!-- Also see
   http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/XWS-SecurityIntro4.html -->
  <!-- <xwss:Timestamp/> manual signing of timestamp -->
  <!-- No sign and no encrypt -->
</xwss:SecurityConfiguration>
```

The following example provides a typical security context file that may be used by the send example above when signing but no encryption is required. The various p-mode security settings are overriden by the security context file including the signature and encryption settings.

```
<xwss:SecurityConfiguration dumpMessages='false'
 retainSecurityHeader='true' enableDynamicPolicy='false' xmlns:xwss='http://java.sun.com/xml/ns/xwss/config'>
<!-- RCSfile: sc.sign.xml,v Revision: 1.1.2.2 Date: 2019-02-13 07:02:45 -->
  <!-- Also see
   http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/XWS-SecurityIntro4.html -->
  <!-- <xwss:Timestamp/> manual signing of timestamp -->
  <!-- sign soap messaging, body and payload -->
  <xwss:Sign includeTimestamp="false"> <!-- set this to false to ensure no sha1 sig is added -->
    <xwss:X509Token certificateAlias='fmsrns'/> <!-- remove keyReferenceType for bst -->
    <xwss:CanonicalizationMethod algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'/>
    <xwss:SignatureMethod algorithm='http://www.w3.org/2001/04/xmldsig-more#rsa-sha256'/>
    <xwss:SignatureTarget type='uri' value='cid:*' enforce='false'>
      <xwss:DigestMethod algorithm='http://www.w3.org/2001/04/xmlenc#sha256'/>
      <xwss:Transform
       algorithm='http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform'/>
    </xwss:SignatureTarget>
    <xwss:SignatureTarget type='qname'
     value='{http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/}Messaging' enforce='false'>
      <xwss:DigestMethod algorithm='http://www.w3.org/2001/04/xmlenc#sha256'/>
      <xwss:Transform algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
        <xwss:AlgorithmParameter name="InclusiveNamespaces" value="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </xwss:Transform>
    </xwss:SignatureTarget>
    <xwss:SignatureTarget type='qname' value='{http://www.w3.org/2003/05/soap-envelope}Body' enforce="false">
      <xwss:DigestMethod algorithm='http://www.w3.org/2001/04/xmlenc#sha256'/>
      <xwss:Transform algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'>
        <xwss:AlgorithmParameter name="InclusiveNamespaces" value="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </xwss:Transform>
    </xwss:SignatureTarget>
  </xwss:Sign>
</xwss:SecurityConfiguration>
```

The following example provides a typical security context file that may be used by the send example above when encryption but no signing is required. The various p-mode security settings are overriden by the security context file including the signature and encryption settings.

```
<xwss:SecurityConfiguration dumpMessages='false' retainSecurityHeader='true'
 enableDynamicPolicy='false' xmlns:xwss='http://java.sun.com/xml/ns/xwss/config'>
<!-- RCSfile: sc.enc.xml,v Revision: 1.1.2.1 Date: 2019-02-13 06:38:57 -->
  <!-- Also see
   http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/XWS-SecurityIntro4.html -->
  <!-- <xwss:Timestamp/> manual signing of timestamp -->
  <!-- encrypt payload -->
  <xwss:Encrypt>
    <xwss:X509Token certificateAlias='fmsrns'/> <!-- remove keyReferenceType for bst -->
    <xwss:KeyEncryptionMethod algorithm="http://www.w3.org/2009/xmlenc11#rsa-oaep"/>
    <xwss:DataEncryptionMethod algorithm="http://www.w3.org/2009/xmlenc11#aes128-gcm"/>
    <xwss:EncryptionTarget type='qname' value='{http://www.w3.org/2003/05/soap-envelope}Body' enforce="false"/>
    <xwss:EncryptionTarget type='uri' value='cid:*' enforce='true'/>
  </xwss:Encrypt>
</xwss:SecurityConfiguration>
```

The following example provides a typical security context file that may be used by the send example above when both signing and encryption are required. The various p-mode security settings are overriden by the security context file including the signature and encryption settings.

```
<xwss:SecurityConfiguration dumpMessages='false' retainSecurityHeader='true'
 enableDynamicPolicy='false' xmlns:xwss='http://java.sun.com/xml/ns/xwss/config'>
<!-- RCSfile: sc.signenc.xml,v Revision: 1.1.2.1 Date: 2019-02-13 06:38:46 -->
  <!-- Also see
   http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/XWS-SecurityIntro4.html -->
  <!-- <xwss:Timestamp/> manual signing of timestamp -->
  <!-- sign soap messaging, body and payload and encrypt payload -->
  <xwss:Sign includeTimestamp="false"> <!-- set this to false to ensure no sha1 sig is added -->
    <xwss:X509Token certificateAlias='fmsrns'/> <!-- remove keyReferenceType for bst -->
    <xwss:CanonicalizationMethod algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'/>
    <xwss:SignatureMethod algorithm='http://www.w3.org/2001/04/xmldsig-more#rsa-sha256'/>
    <xwss:SignatureTarget type='uri' value='cid:*' enforce='false'>
      <xwss:DigestMethod algorithm='http://www.w3.org/2001/04/xmlenc#sha256'/>
      <xwss:Transform
       algorithm='http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform'/>
    </xwss:SignatureTarget>
    <xwss:SignatureTarget type='qname'
     value='{http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/}Messaging' enforce='false'>
      <xwss:DigestMethod algorithm='http://www.w3.org/2001/04/xmlenc#sha256'/>
      <xwss:Transform algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
        <xwss:AlgorithmParameter name="InclusiveNamespaces" value="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </xwss:Transform>
    </xwss:SignatureTarget>
    <xwss:SignatureTarget type='qname' value='{http://www.w3.org/2003/05/soap-envelope}Body' enforce="false">
      <xwss:DigestMethod algorithm='http://www.w3.org/2001/04/xmlenc#sha256'/>
      <xwss:Transform algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'>
        <xwss:AlgorithmParameter name="InclusiveNamespaces" value="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </xwss:Transform>
    </xwss:SignatureTarget>
  </xwss:Sign>
  <xwss:Encrypt>
    <xwss:X509Token certificateAlias='fmsrns'/> <!-- remove keyReferenceType for bst -->
    <xwss:KeyEncryptionMethod algorithm="http://www.w3.org/2009/xmlenc11#rsa-oaep"/>
    <xwss:DataEncryptionMethod algorithm="http://www.w3.org/2009/xmlenc11#aes128-gcm"/>
    <xwss:EncryptionTarget type='qname' value='{http://www.w3.org/2003/05/soap-envelope}Body' enforce="false"/>
    <xwss:EncryptionTarget type='uri' value='cid:*' enforce='true'/>
  </xwss:Encrypt>
</xwss:SecurityConfiguration>
```

Interpreting AS4 Light Client Results

Once execution has completed, the AS4 Client Utility will return an exit code. These codes are documented in the the Section called *AS4 Client Return Values* in Chapter 5. Further messages are available in `stdout` and `stderr`.

All log messages from the AS4 Client Utility are written to `stderr`. These messages include errors from the Server as well as any transmission or validation errors.

Chapter 5. RosettaNet/ebXML Application Configuration

This section describes the FMS ebXML and RosettaNet server configuration and business application configuration requirements when interfacing to a business application.

ebXML and RosettaNet Client Utility

`fms-client.jar` is an interface utility that must be invoked by the business application to communicate XML business documents via FMS to a remote business partner.

Invocation

Invoke the FMS light client utility as follows to display usage requirements

```
java -jar fms-client.jar -?

Usage: java -jar fms-client.jar
Where:
  -a "<attachment File path>:<attachment mime type>:<attachment description>:<attachmentID>" \
  can appear 0..n times.
  -b <Buffer Size> The send buffer size in bytes to use. Defaults to: 32768
  -c <P-Mode ID> The Processing Mode ID to use as a configuration for this request.
  -d <recipientPartnerIdentifier> The Partner Identifier of the recipient. Defaults to: null
  -e <Event> The Processing Mode event that this message should use. Defaults to the first Event \
    in the specified P-Mode
  -f <XML File path> The file path to the XML file that should be sent. Required...
  -g Enable GZIP Stream Compression for faster upload over slow connections
  -h <Host Name> The final destination of the packaged envelope. Only for Test environments, \
    actual destination in Production environments is defined in the server configuration files.
  -i <Initial Timeout> A timeout in milliseconds that dictates the timeout required before an ACK \
    is received during transmission from the local FMS server, refer to -t <ReceiptTimeout> for the \
    timeout while waiting for a response from the remote server. Defaults to: 10000
  -k <Keystore Location> The location of the keystore that contains the SSL certificates used \
    during SSL communication. Defaults to: /etc/ssl/certs/java/cacerts (ubuntu, debian) or /etc/pki/java/cacerts (redhat, fec
    Not currently used - pending SSL client authentication.
  -l <recipientLocation> The case sensitive recipient location ID. Defaults to: null
  -m <contentID> A content instance tracking identification string.
  -q <messageID> Perform an Acknowledgement query on the provided messageID
  -p <MPC> The Message Partition Channel for use with this PullRequest
  -r <Retry Attempts> The amount of retries that should be attempted before giving up. \
    Defaults to: 3
  -s <Schema Type> The Schema Type to be used for this transaction. As defined in the \
    Collaboration Protocol Agreement (CPA) (If Applicable). (Case Sensitive). Defaults to: null
  -t <Receipt Timeout> A timeout in milliseconds that should trigger if a Receipt is not \
    received from the FMS server for both server and client, refer to -i <Initial Timeout> for \
    upload timeouts. Not required if -w is specified. Defaults to: 300000
  -v Print version and exit.
  -w Do not wait for a Receipt Acknowledgement before exiting, should be specified with -J in case \
    a followup acknowledgement check (-Q) needs to be performed.
  -A <Action Type> The Action Type to be used for this transaction. As defined in the \
    Collaboration Protocol Agreement (CPA) (If Applicable). (Case Sensitive). Defaults to: null
  -C <CPA UID | CPA URL> The Unique ID for the CPA to use for this transaction or directly to \
    the CPA. Defaults to: null
  -D <senderPartnerIdentifier> The Partner Identifier of the sender. Defaults to: null
  -E <ExternalProperties> A comma-delimited Variable=Value list which will be inserted into the \
    transport envelope (if applicable). e.g: -E "MessageIdentifier=ABC123, \
    ContentFilename=InvoiceABC123.xml"
  -H <serverHostName> The server that this client should connect to. Defaults to: localhost
  -I  Ignore the Message ID store '.FMS-ClientMessageID.tmp' when sending requests.
  -J  Append to the unacknowledged Message ID(s) stored in the '.FMS-ClientMessageID.tmp' file if \
    it exists.
  -L <senderLocation> The case sensitive sender location ID. Defaults to: null
  -M <refToMessageID> The reference to message identifier that this message is in response to \
    (If Applicable).
  -O <logLevel> The log level to use when sending requests. One of {OFF, SEVERE, WARNING, \
    INFO, CONFIG, FINE, FINER, FINEST, ALL} (Case InSensitive). Defaults to: OFF
  -Q Perform an Acknowledgement query on the messageID(s) stored in the file \
    '.FMS-ClientMessageID.tmp'. Only available if the configuration database is enabled.
  -P <serverPort> The port number that this client should connect to. Defaults to: 29350
  -R <Retry wait time> The time, in milliseconds, between retry attempts. Defaults to: 1000
  -S  Turn off SecureSocketLayer (SSL) during upload. Not recommended.
  -T <Connection Type> The connection type to use. \
     One of {<acronym>RosettaNet</acronym>, <acronym>ebXML</acronym>} \
     (Case Sensitive). Defaults to: ebXML.

Got: [-?]
```

Interpreting Light Client Results

Once execution has completed, the Client Utility will return an exit code. These codes are documented in the the Section called *ebXML and RosettaNet Client Return Values*. Further messages are available in `stdout` and `stderr`.

All log messages from the Client Utility are written to `stderr`. These messages include errors from the Server as well as any transmission or validation errors.

If transmission of the envelope is successful and a message from the Remote Server has been received it will appear in `stdout`. This message is in XML format and could be a SOAP Message (ebXML) or one of `Exception` or `ReceiptAcknowledgment` (RosettaNet).

The layout of a RosettaNet `Exception` XML object:

```
1   1       Exception
2   1             |-- ExceptionDescription
3   1             |       |-- errorClassification.GlobalMessageExceptionCode
4   1             |       |-- errorDescription.FreeFormText
5   0..1          |       |-- offendingMessageComponent.GlobalMessageComponentCode
6   1             |-- GlobalExceptionTypeCode
```

Where:

- GlobalMessageExceptionCode - Identifies the specific error which occurred during message processing.
- FreeFormText - Unformatted text.
- GlobalMessageComponentCode - Identifies a message component, e.g. Preamble, Delivery Header, Service Header.
- GlobalExceptionTypeCode - A unique identifier specifying the type of exception encountered during message processing, see the Section called *RosettaNet® Error Messages*

The layout of a RosettaNet `ReceiptAcknowledgment` XML object:

```
1       1       ReceiptAcknowledgment
2       0..1        |-- NonRepudiationInformation
3       1           |       |-- OriginalMessageDigest
```

Where:

- OriginalMessageDigest - The base-64 encoded digest of the entire original mime message received. The digest MUST use the same algorithm as the original signed message.

ebXML/RosettaNet Client SSL Configuration

`fms-client.jar` may be used to communicate messages using no encryption to the server or using a secure connection to the server.

Before a secure connection can be made to the server, SSL MUST be configured. Below are a number of steps to follow in order to ensure a secure connection between the client and server:

1. An export of the server certificate must be done in order to import the certificate on the client side. See the Section called *Exporting a Public Key (Certificate)* in Chapter 2
2. Import the server certificate to the default java keystore on the client side. See the Section called *Importing Public Keys (Certificates) into the Client Keystore* in Chapter 2

Attachments

To include an attachment with an XML document include the argument `-a <attachment file path>:<attachment mime type>:<attachment description>` for each attachment at the end of the invocation statement.

Table 5-1. Attachment Usage

Argument	Description
<attachment file path>	The attachment's exact location in URL
<attachment mime type>	The attachment's format. See the table below for which formats are supported and what their mime types are.
<attachment description>	A short description of the attachment. eg: "Price List 2008"

Table 5-2. Attachment Mime Types

Supported Mime Types:

AS4 Client Return Values

`fmsclient.as4.jar` returns the following values on completion

- 0 - Ok
- 1 - Usage - eg when invoking with incorrect command line arguments
- 2 - Connect Exception - occurs if a network connection is not available. Eg. network cable unplugged
- 3 - SSL Exception - could not validate key - only from 5.3.4-2

  If this return value is accompanied with `ClientException - PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target` then this may be due to the SSL certificate verification failing either because the root certificate does not exist or has been revoked.
- 4 - Timeout - occurs on read timeout and on establishing a connection timeout
- 5 - not currently used
- 6 - NPE - internal null pointer error
- 7 - not currently used
- 8 - not currently used
- 9 - not currently used
- 10 - Unknown error - unidentifiable error condition. Use '-d' command line switch to debug.

ebXML and RosettaNet Client Return Values

`fms-client.jar` returns the following values on completion

- 0 - No transmission error.
- 1 - Timeout - Socket timeout occurred, either on client or remote servers.
- 2 - IO Exception - This ranges from failure to connect to errors in the underlying protocol.
- 3 - SSL Exception - An error occurred while configuring or negotiating a SSL connection.
- 4 - XML Parse Exception - An error occurred while attempting to parse and validate the transmitted XML headers and content.
- 5 - Version Error - There is a difference between this client's version and the server's. Read stderr for location to download a new client.
- 6 - File not Found Error - The supplied files could not be located, refer to `stderr` as to the location of the failed files.
- 7 - Security Error - There was a failure in encryption or digital signatures, refer to server logs for more information.
- 8 - CPA Error - The supplied CPA URL/UID was unretrievable or unreadable, refer to server logs for more information.
- 9 - Acknowledgement Query - The result of the Acknowledgement Query
- 10 - Protocol Exception - There was an error in the underlying protocol, refer to server logs for more information.
- 11 - Sent Unacknowledged - The request has been transmitted to the server but the client will not wait for an acknowledgement.
- 12 - Partner Identifier Exception - There was a misconfiguration or error while using the supplied Partner Identifier.
- 13 - Empty Message Partition Channel - There were no messages avilable or the MPC was not found while performing a PullRequest.
- 14 - ProcessingMode Error - The server encountered a misconfiguration or the supplied data does not correlate with the supplied ProcessingMode
- 15 - HTTP Status code not accepted - The server received an error HTTP status code from the remote server.
- 16 - Unknown Error - The server or the client encounted an unknown error, read stderr for error information.

ebXML Error Messages

Even though a return code of 0 was received, an Error might have occurred on the remote servers. If an exception occurred then `fms-client.jar` can return the following ebXML messages in the Error XML element of the SOAP message which is printed to stdout after invocation.

If no error occurred then the SOAP Message returned to stdout should contain the SignalMessage Receipt element.

Table 5-3. ebMS Processing Errors

Error Code	Description	Severity	Category	Semantics
EBMS:0001	ValueNotRecognized	failure	Content	Although the message document is well formed and schema valid, some element/attribute contains a value that could not be recognized and therefore could not be used by the MSH.

Error Code	Description	Severity	Category	Semantics
EBMS:0002	FeatureNotSupported	warning	Content	Although the message document is well formed and schema valid, some element/attribute value cannot be processed as expected because the related feature is not supported by the MSH.
EBMS:0003	ValueInconsistent	failure	Content	Although the message document is well formed and schema valid, some element/attribute value is inconsistent either with the content of other element/attribute, or with the processing mode of the MSH, or with the normative requirements of the ebMS specification.
EBMS:0004	Other	failure	Content	
EBMS:0005	ConnectionFailure	failure	Communication	The MSH is experiencing temporary or permanent failure in trying to open a transport connection with a remote MSH.
EBMS:0006	EmptyMessagePartition Channel	warning	Communication	There is no message available for pulling from this MPC at this moment.
EBMS:0007	MimeInconsistency	failure	Unpackaging	The use of MIME is not consistent with the required usage in this specification.
EBMS:0008	FeatureNotSupported	failure	Unpackaging	Although the message document is well formed and schema valid, the presence or absence of some element/ attribute is not consistent with the capability of the MSH, with respect to supported features.
EBMS:0009	InvalidHeader	failure	Unpackaging	The ebMS header is either not well formed as an XML document, or does not conform to the ebMS packaging rules.
EBMS:0010	ProcessingMode Mismatch	failure	Processing	The ebMS header or another header (e.g. reliability, security) expected by the MSH is not compatible with the expected content, based on the associated P-Mode.
EBMS:0011	ExternalPayloadError	failure	Content	The MSH is unable to resolve an external payload reference (i.e. a Part that is not contained within the ebMS Message, as identified by a PartInfo/href URI).

Table 5-4. ebMS Security Processing Errors

Error Code	Description	Severity	Category	Semantics

Error Code	Description	Severity	Category	Semantics
EBMS:0101	FailedAuthentication	failure	Processing	The signature in the Security header intended for the "ebms" SOAP actor, could not be validated by the Security module.
EBMS:0102	FailedDecryption	failure	Processing	The encrypted data reference the Security header intended for the "ebms" SOAP actor could not be decrypted by the Security Module.
EBMS:0103	PolicyNoncompliance	failure	Processing	The processor determined that the message security methods, parameters, scope or other security policy-level requirements or agreements were not satisfied.

Table 5-5. ebMS Reliable Messaging Errors

Error Code	Description	Severity	Category	Semantics
EBMS:0201	DysfunctionalReliability	failure	Processing	Some reliability function as implemented by the Reliability module, is not operational, or the reliability state associated with this message sequence is not valid.
EBMS:0202	DeliveryFailure	failure	Communication	Although the message was sent under Guaranteed delivery requirement, the Reliability module could not get assurance that the message was properly delivered, in spite of resending efforts.

Table 5-6. FMS Messaging Errors

Error Message	Description
Unable to locate compatible ProcessingMode Event under P-Mode	A compatible MEP being either one of PULL, PULL_AND_PUSH, or PUSH_AND_PULL, for a particular P-Mode event does not exist. No message could therefor be pulled.

RosettaNet® Error Messages

Even though a return code of 0 was received, an Exception might have occurred on the remote servers. If an exception occurred then `fms-client.jar` returns the following RosettaNet® messages in the Exception XML object body which is printed to stdout during execution.

If no exception occurred then the XML object returned to stdout should be ReceiptAcknowledgment.

- PKG.MESG.GENERR Error during packaging. General error
- PRF.ACTN.GENERR Error during action performance. General Error
- PRF.DICT.VALERR Error during action performance. Validating the Service Content against a PIP® specified dictionary
- UNP.MESG.GENERR Error during unpackaging. General error

- UNP.MESG.SIGNERR Error during unpackaging. Verifying the signature of the RosettaNet® Business Message
- UNP.PRMB.READERR Error during unpackaging. Reading the Preamble
- UNP.PRMB.VALERR Error during unpackaging. Validating the Preamble
- UNP.DHDR.READERR Error during unpackaging. Reading the Delivery Header
- UNP.DHDR.VALERR Error during unpackaging. Validating the Delivery Header
- UNP.SHDR.READERR Error during unpackaging. Reading the Service Header
- UNP.SHDR.VALERR Error during unpackaging. Validating the Service Header
- UNP.SHDR.MNFSTERR Error during unpackaging. Verifying Manifest against the actual attachment body parts
- UNP.MESG.SEQERR Error during unpackaging. Validating the message sequence
- UNP.MESG.RESPTYPERR Unexpected Response type in the HTTP header
- UNP.MESG.DCRYPTERR Error Decrypting the message
- UNP.SCON.READERR Error during unpackaging. Reading the Service Content
- UNP.SCON.VALERR Error during unpackaging. Validating the Service Content

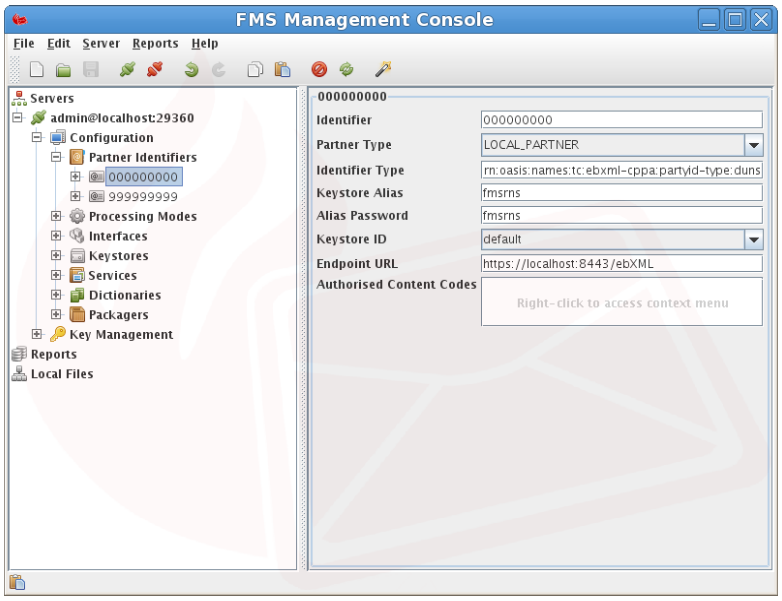
Configuring Pull Requests

Since ebMS version 3 and FMS version 4.1.2.4 it has been possible to generate and receive Pull Requests. Pull Requests are messages that are queued on a remote server waiting to be pulled by a remote MSH. This is especially useful in a situation where a business application is installed on a system that does not have a permanent internet connection permitting remote pull requests to originate private networks behind firewalls, from dynamic IP addresses or a dialup internet connections.

In order to deliver messages to a queue for Pulling the following MUST be performed:

- The destination Partner Identifier type MUST be set to PULL_PARTNER.
- The destination Partner Identifier endpoint must be set to the value of the client Message Partition Channel MPC.

Messages may be transmitted to the PULL_PARTNER Partner Identifier and will be stored in an associated MPC queue.



000000000

Identifier	000000000
Partner Type	LOCAL_PARTNER
Identifier Type	rn:oasis:names:tc:ebxml-cppa:partyid-type:duns
Keystore Alias	fmsrns
Alias Password	fmsrns
Keystore ID	default
Endpoint URL	https://localhost:8443/ebXML
Authorised Content Codes	Right-click to access context menu

Figure 5-1. PartnerIdentifier - Pull Partner

In order to retrieve a message destined for a Partner Identifier MPC the following MUST be performed:

- The MPC value must be known in order to collect messages
- The invocation client must have the Recipient Partner Identifier (-d) as well as the pull request argument (-p <MPC value>) as arguments.

Note: Messages are delivered in a First In First Out order (FIFO).

Note (2): Messages will be saved to the `delivered-content` directory of the local server for collection by the Business Application.

Message Queries

When sending asynchronous messages the FMS client need not wait for an acknowledgement from the remote server. However should an acknowledgement be required or at least message delivery verified then it is possible to use the FMS client to query the acknowledgement status of any message identifier(s) at a later stage.

The FMS Client has the capability of storing a list of Message Identifiers that have been sent, acknowledged or not. By specifying the `-J` argument during FMS Client invocation the Client will append Message Identifiers in the file `.FMS-ClientMessageID.tmp` for later querying/tracking purposes, otherwise only the most recent Message Identifier is stored. This functionality may be disabled by specifying the `-I` argument which sets the Client to ignore the Message Identifier storage file.

There exist two methods to query Message Identifier(s) by using the Client:

- If the Message Identifier storage file is used and populated then the FMS Client may be invoked using `-Q` which will then transfer the Message Identifier storage file to the server and perform an Acknowledgement Query on the Message Identifiers.
- However should just a single Message Identifier require an Acknowledgement Query then it is possible to use the `-q <Message ID>` argument to query just a single Message Identifier.

Once the Acknowledgement check has been performed the Client will return each Message Identifier to stdout in the format `'<MessageID> <Acknowledged> <Time Acknowledged>'`. For example:

```
FMS-20081118-145152.160-0.26004090449932216 true 2008-11-18 14:52:00.503
```

Chapter 6. FMS Tools and Utilites

This section describes various utilities included with the FMS distribution. These include a wrapper for the `fmsclient.as4.jar` push client, a utility for remotely managing the server.

fmsas4lc Client Wrapper

The fmsas4lc wrapper is installed in `/usr/local/bin` for Linux and unix systems. It provides a simple wrapper in the user PATH for invoking the FMS light client Java jar fmsclient.as4.jar installed in a typically non-visible directory.

Invoke the fmas4lc utility as follows to display usage requirements

```
fmas4lc -?
Usage: ./fmsas4lc [--jar client_jar] -- [FMS AS4 Light Client Command Line Options] [XML_PAYLOAD]
  where the optional client_jar is the fully qualified path name of the FMS AS4 Light Client jar file
  and the optional XML_PAYLOAD is the xml payload that will be inserted into the SOAP 1.2 envelope body.
```

The following environment variables may be set according to requirements before invoking fmas4lc

1. FMS_JAVA_ARGS. Override arguments to the Java virtual machine. Defaults to `-Xmx2048m`.

   AS4CLIENTJAR_DIR. Override the directory to the fmsclient.as4.jar jar file. Defaults to `/usr/share/java/fms` for Linux systems.

   AS4CLIENTJAR. Override the name of the client jar. Defaults to `fmsclient.as4.jar`.

fmsconf Server Configuration Utility

Command Line Utility to administer and report on the FMS server. This utility permits an adminstrator to remotely (or locally) send commands to FMS servers and can be used for scripting server administration tasks without user intervention and/or to create customised server management utilities with similiar functionality to the FMC.

fmsconf utility uses default user login credentials of `admin` and password `admin` to connect to the FMS server. Ensure that the necessary credentials are configured prior to connecting to the server as described in section the Section called *Admin User Creation* in Chapter 3.

The following commands may be supplied using fmsconf '-x admin_command' option

- __FMS_Admin_Version. Return the server version.
- __FMS_Admin_ReloadConfigurations. Instruct the server to reload the configuration. This command is used when a new configuration must be reloaded without restarting the FMS server. All active messaging sessions are completed before any reload.
- __FMS_Admin_GetConfiguration. Retrieve the configuration from the server. This command is used to retrieve an FMS server configuration.
- __FMS_Admin_ReloadConnections. Instruct the server to reload any changes to the connection configuration. This command is used when a new configuration file with updated connection settings must be activated without restarting the FMS server. All active messaging sessions are completed before any reload.
- __FMS_Admin_CheckLicense. Instruct the server to display the license details. Useful to determine the period for which the license is active.

- __FMS_Admin_ReloadLicense. Instruct the server to reload the license activation file without restarting the server.
- __FMS_Admin_ConnectionStatistics. Instruct the server to display connection statistics of a running server.
- __FMS_Admin_ListCerts `truststore truststoreType truststorePassword`. Instruct the server to list the certs in any of the various truststores of a running server. The `truststoreType` must be one of `JKS`, `PCKS_12` or `DATABASE`.
- __FMS_Admin_ServerLogLevel `level`. Change the server logging level. The level must be one of WARN, INFO, DEBUG or TRACE.
- __FMS_Admin_GetServerLogLevel `level`. Retrieve the server logging level.
- __FMS_Admin_AddCert `truststore truststoreType truststorePassword alias certificateBase64`. Instruct the server to add a new certificate specified by the `certificateBase64` string and identified by the `alias`.

  The `updateCert.sh` script listed in the Section called *Script to dynamically update FMS public certificates* provides a wrapper for fmsconf to dynamically update certificates on a running FMS server.
- __FMS_Admin_GetCert `truststore truststoreType truststorePassword alias` [`full`]. Instruct the server to return the certificate in base64 encoded format identified by the `alias`.

  The full certificate detail including fingerprint will be displayed if the optional `full` argument is included.
- __FMS_Admin_DeleteCert `truststore truststoreType truststorePassword alias` [`result`]. Instruct the server to delete the cert identified by the `alias` from the keystore identified by `truststore`.

  The deleted certificate alias and fingerprint will be displayed after deletion if the optional `result` argument is included.
- __FMS_Admin_RenameKey `truststore truststoreType truststorePassword keyPassword currentKeyAlias newKeyAlias alias` [`result`]. Instruct the server to rename a certificate or key `alias` in the keystore identified by `truststore`.

fmsconf Requirements

The fmsconf utility uses the following freely available programs which are typically available on any Linux system.

- python
- openssl
- getopt

fmsconf Administrator Configuration

An Administrator listener connection must be configured and enabled in the FMS configuration file. The Administrator configuration is automatically generated and may be adjusted to look similar to the following sample.

```
<cc:listener>
  <cc:name>Administration</cc:name>
  <cc:className>com.flame.connection.impl.admin.Admin</cc:className>
  <cc:aliasRef cc:keystoreID="administrator">
    <cc:alias>fmsrns</cc:alias>
    <cc:password>fmsrns</cc:password>
  </cc:aliasRef>
  <prop:Properties>
    <prop:comment>Configuration options for com.flame.connection.impl.admin.Admin</prop:comment>
    <prop:entry prop:key="HOST">yourdomain.com</prop:entry>
    <prop:entry prop:key="MAXIMUM_CONCURRENT_CONNECTIONS">1</prop:entry>
    <prop:entry prop:key="READ_TIMEOUT">0</prop:entry>
    <prop:entry prop:key="USE_SSL">true</prop:entry>
    <prop:entry prop:key="MOTD"><html>Welcome to the FMS Management Console<br><br> \
      Server listening: %2$s<br>Connection: %1$s@%3$s<br><br> \
      Server Started: %4$tc<br>Uptime: %5$ts seconds</html></prop:entry>
    <prop:entry prop:key="SSL_NEED_CLIENT_AUTH">false</prop:entry>
    <prop:entry prop:key="PORT">29360</prop:entry>
    <prop:entry prop:key="LOG4J_PATTERN">%-5p [%t]: %m%n</prop:entry>
  </prop:Properties>
</cc:listener>
<cc:acl cc:order="allow_deny"/>
```

The keystore configuration is automatically generated at system initialisation and may be adjusted to conform to deployment requirements as follows

```
<cc:keystoreRef cc:ID="administrator">
  <cc:name>admin.jks</cc:name>
  <cc:type>JKS</cc:type>
  <cc:pass>mypassword</cc:pass>
</cc:keystoreRef>
```

fmsconf Administrator User

An Administrator user must also be configured on the server. This should be done as per the instructions in the Section called *Admin User Creation* in Chapter 3.

fmsconf Administrator Certificates

If the Admin listener configuration `USE_SSL` and `SSL_NEED_CLIENT_AUTH` properties are set to '`true`' then before using fmsconf to connect to the FMS server the necessary key and certificates must be created. The public certificate must be imported into the FMS Administrator listener truststore (admin.jks) as defined above.

This may be done by creating the private key in file `mcpriv.pem`, and public certificate in file `mcpub.pem` as per the following instructions.

```
            openssl genrsa -out mcpriv.pem 1024
            openssl req -new -x509 -key mcpriv.pem -out mcpub.pem -days 1095
```

where `mcpriv.pem` is the private key and `mcpub.pem` is the public certificate to use when connecting to FMS.

Import `mcpub.pem` into the FMS truststore as follows

```
keytool -import -keystore server/admin.jks -storepass changeit -file mcpub.pem -alias admin
```

Note: Ensure that the FMS truststore contains a private key else admin ssl connection will cause the following problem

```
140735140426592:error:14094410:SSL routines:ssl3_read_bytes:sslv3 \
  alert handshake failure:s3_pkt.c:1472:SSL alert number 40
140735140426592:error:1409E0E5:SSL routines:ssl3_write_bytes: \
  ssl handshake failure:s3_pkt.c:656:
```

which may be done as follows.

```
            keytool -genkeypair -keyalg RSA -validity 365 -keystore /home/fms/admin.jks  -storepass 123456 -keypass fmsrns
```

Refer to the Section called *Keystore Setup and Examples* in Chapter 2 for further details on key and certificate generation for FMS.

fmsconf Usage

Invoke the fmsconf utility as follows to see the usage

```
fmsconf -?
fmsconf: invalid option -- '?'
Usage: fmsconf [-d] [-c public_certificate_file] [-h fms_host] [-k private_key_file] [-P fms_host_admin_port] [-p password] [
  where
    -d optional - switch debug to s_client on
    -c /path/to/public_certificate_file - only required if the client authentication property 'SSL_NEED_CLIENT_AUTH' is set t
    -h host - optional - defaults to xenialmac
    -H : optional - displays help information
    -k /path/to/public_key_file - only required if the client authentication property 'SSL_NEED_CLIENT_AUTH' is set to 'true'
    -P port - optional - defaults to 29360
    -p password - optional - defaults to admin
    -u admin_username - optional - defaults to admin
    -x admin_command - optional - defaults to '__FMS_Admin_Version'. Can be any one of
'__FMS_Admin_Version'
'__FMS_Admin_ReloadConfigurations'
'__FMS_Admin_GetConfiguration'
'__FMS_Admin_ReloadConnections'
'__FMS_Admin_ReloadLicence'
'__FMS_Admin_CheckLicence'
'__FMS_Admin_ConnectionStatistics'
'__FMS_Admin_ServerLogLevel WARN|INFO|DEBUG|TRACE'
'__FMS_Admin_GetServerLogLevel'
'__FMS_Admin_ListCerts certs.flame JKS|PKCS12|DATABASE keystorepassword'
'__FMS_Admin_GetCert certs.flame JKS|PKCS12|DATABASE keystorepassword mykeyalias [full (FMS 5.4.2+ only)]'
'__FMS_Admin_DeleteCert certs.flame JKS|PKCS12|DATABASE keystorepassword mykeyalias [result (FMS 5.4.2+ only)]'
'__FMS_Admin_AddCert certs.flame JKS|PKCS12|DATABASE keystorepassword mykeyalias certificateBase64'
'__FMS_Admin_RenameKey certs.flame JKS|PKCS12|DATABASE keystorepassword keypassword currentkeyalias newkeyalias'

  Use this utility to dynamically update a running FMS configuration or report on the status of a running FMS instance.
```

fmsconf Environment Variables

The fmsconf utility behaviour may be customised by adjusting the following environment variables

- `ERROR_FILE` defaults to fmsconf.<pid>.
- `OPENSSL`=${OPENSSL:=openssl}
- TLS_ARG Defaults to to an empty string. Results in the openssl s_client process negotiating the highest mutually supported protocol version. May be set up to '-tls1_3' if required to only use TLSv1.3 and if supported in the `java.security` file.
- ADMINUSER Defaults to `admin` and must match the setting in the FMS the administration users `user.cfg` file as per the instructions in the Section called *Admin User Creation* in Chapter 3.
- KEY Defaults to `mcpriv.pem` and only required if SSL_NEED_CLIENT_AUTH is set on the server administration listener properties.
- CER Defaults to `mcpub.pem` and only required if SSL_NEED_CLIENT_AUTH is set on the server administration listener properties.
- ADMINPASS Defaults to `admin` and must match the setting in the FMS the administration users `user.cfg` file as per the instructions in the Section called *Admin User Creation* in Chapter 3.

Script to dynamically update FMS public certificates

The `updateCert.sh` bash script may be used to replace an existing client certificate with a new certificate in the FMS keystore. Prior to running this ensure that the fmsconf utility is installed and working before using this script.

The script makes a backup copy of the existing client certificate being replaced, then deletes it and then loads the new client certificate.

The following environment variables may be set prior to running the script.

- `FMS_HOME` - defaults to the `fms` user directory, typically `/home/fms`.
- `BACKUP_DIR` - defaults to `/home/fms/tmp`.
-

  `TRUSTSTORE_PW` - defaults to `changeit`.
- `TRUSTSTORE_TYPE` - defaults to `JKS`. May be set to `PKCS12`.

```
#/usr/bin/env bash

##
## RCSfile: updateCert.sh,v Revision: 1.1.2.8 Date: 2019-08-01 10:08:45
##
# (c) Flame Computing Enterprises cc - All rights reserved
#
# Update FMS public certificate at specified time
#
# Usage: updateCert.sh truststore cert_file cert_alias [fms_host]
# where
#   truststore: name of the Java Keystore known to FMS in which the certificate must be updated
#   cert_file: file containing the certificate in x format
#   cert_alias: alias to be associated with the certificate
#   fms_host: host on which the FMS administration interface is listening
#
# Eg.
#   updateCert.sh certs.flame /path/to/cert.crt myalias 127.0.0.1
#
# To update a certificate on a specific date use the ’at’ command to schedule this command as follows
#
#   echo "/home/fms/trigger/updateCert.sh flame.jks /home/fms/remotePartner.crt remotePartner" | at ’11:15 Oct 17’
#
# where
#   flame.jks is the FMS truststore in which the certificate must be updated
#   /home/fms/remotePartner.crt is the file containing the public certificate
#   remotePartner is the alias of the certificate to be updated in the truststore
#
# Requirements
#   fmsconf
#   fms
#
# set -x

# Default ENV variables
TRUSTSTORE_TYPE=${TRUSTSTORE_TYPE:="JKS"}

TRUSTSTORE_PW=${TRUSTSTORE_PW:="changeit"}

FMS_HOME=${FMS_HOME:="$( grep ’^fms:’ /etc/passwd |cut -d: -f 6 )"}

if [ "${FMS_HOME}" == "" ]
then
  FMS_HOME="/home/fms" # hardwire it.
fi
```

```
BACKUP_DIR=${BACKUP_DIR:="${FMS_HOME}/tmp"}

Usage () {
  echo <<EOF "Usage: $0 truststore cert_file cert_alias [fms_host]
  where
    truststore: name of the Java Keystore known to FMS in which the certificate must be updated
    cert_file: file containing the certificate in x format
    cert_alias: alias to be associated with the certificate
    fms_host: host on which the FMS administration interface is listening. Default is '127.0.0.1'

  Use this utility from the 'at' command to dynamically update a running FMS certificate on a specified date and time.

  The following environment variables may be set before invoking '${0}'

    FMS_HOME with default '${FMS_HOME}'.

    TRUSTSTORE_TYPE with default '${TRUSTSTORE_TYPE}'.

    TRUSTSTORE_PW with default '${TRUSTSTORE_PW}'.

    BACKUP_DIR with default '${BACKUP_DIR}' for saving certificates being refreshed.
  "
EOF
  exit 1
}

if [ $# -lt 3 -o $# -gt 4 ]
then
  Usage
fi

TRUSTSTORE=$1

CERT_FILE=$2

if ! test -f ${CERT_FILE}; then
  echo "Could not locate certificate file '${CERT_FILE}'."
  Usage
else # Determine certificate
  if grep -q "BEGIN CERTIFICATE" ${CERT_FILE}
  then # Looks like a cert
    CERTIFICATE=$(cat ${CERT_FILE} | grep -v 'CERTIFICATE')
  else
    echo "Could not locate a certificate in ${CERT_FILE}."
    Usage
  fi
fi

CERT_ALIAS=$3

FMS_HOST=${FMS_HOST:='127.0.0.1'}

if [ $# -eq 4 ]
then
  FMS_HOST=$4
fi

if ! test -d ${BACKUP_DIR}
then
  echo "Could not locate the backup directory ${BACKUP_DIR}."
  Usage
fi

echo "Saving certificate with alias '${CERT_ALIAS}' to ${BACKUP_DIR}/${CERT_ALIAS}.$$.crt"

# First save the current cert

fmsconf -h ${FMS_HOST} -x "__FMS_Admin_GetCert ${TRUSTSTORE} ${TRUSTSTORE_TYPE} ${TRUSTSTORE_PW} ${CERT_ALIAS}" |grep -v '^OK
retval=$?
if [ $retval -ne 0 ]
then
```

```
  echo "fmsconf returned error '$retval'. Could not retrieve certificate for '${CERT_ALIAS}' from '${TRUSTSTORE}' at '${FMS_H
  Usage
fi

# Delete the current cert

echo "Replacing certificate with alias '${CERT_ALIAS}' in '${FMS_HOME}/${TRUSTSTORE}' and details as follows ..."
echo
keytool -v -list -keystore ${FMS_HOME}/${TRUSTSTORE} -storepass changeit -alias ${CERT_ALIAS} 2>/dev/null |head -12 |tail -8
echo

fmsconf -h ${FMS_HOST} -x "__FMS_Admin_DeleteCert ${TRUSTSTORE} ${TRUSTSTORE_TYPE} ${TRUSTSTORE_PW} ${CERT_ALIAS}"
retval=$?
if [ $retval -ne 0 ]
then
  echo "fmsconf returned error '$retval'. Could not delete certificate for '${CERT_ALIAS}' from '${TRUSTSTORE}' at '${FMS_HOS
  Usage
fi

# Add the new cert
echo "with new certificate and details as follows ..."
echo
keytool -printcert -file ${CERT_FILE} |head -8
echo

fmsconf -h ${FMS_HOST} -x "__FMS_Admin_AddCert ${TRUSTSTORE} ${TRUSTSTORE_TYPE} ${TRUSTSTORE_PW} ${CERT_ALIAS} ${CERTIFICATE}
retval=$?
if [ $retval -ne 0 ]
then
  echo "fmsconf returned error '$retval'. Could not add certificate for '${CERT_ALIAS}' to '${TRUSTSTORE}' at '${FMS_HOST}'."
  Usage
fi
```

Certificate updates may be scheduled using the `at` command and the above `updateCert.sh` script as per the following example on *nix systems

```
echo "/home/fms/trigger/updateCert.sh '2019-06-11 09:55:00' /home/fms/flame.jks /path/to/NewCert.pem CERTALIAS localhost" | a
```

Appendix A. Server Command Line Options

```
[root@localhost fms]#  java -server -jar fms.jar -?

(c) Flame Message Server usage:
-s >main settings file< -l >licence file< -? --help
Where:
-d provides the location of the installation directory. Default: .
        -i informs FMS to do initialisation. Previous configurations will be backed up. \
          Initialisation of configuration files will occur if these do not exist
-ic informs FMS to do initialisation only and not listen.
-l provides the location to the FMS licence file. Default: fms.lcn
-s provides the location to the main settings file. Default: main.conf
-u provides the location to the administration users file. Default: user.cfg
-v retrieve the build version.
-w do not overwrite the existing server configuration in the event of configuration updates to a running server.
-? or --help displays this message.
```

Appendix B. ebMS Reliable Messaging

To be implemented.

Appendix C. FMS Log4j based Logging Configuration

The FMS logging sub system generates log messages at various levels which can be routed to various log files and various external logging systems. The logs provide a trace of messaging transactions, debug statements and errors.

Log4j logging configuration files are located in the `log` subdirectory of the FMS installation directory.

- RPM based distributions - loggging messages are typically sent to `/var/log/messages` assuming rsyslog has been configured correctly. Typical `log4j.properties` configuration files are illustrated below.
- DEB (debian and ubuntu) based distributions - loggging messages are typically sent to `/var/log/syslog` assuming rsyslog has been configured correctly. Typical `log4j.properties` configuration files are illustrated below.
- ZIP distributions - Log messages will typically be sent to log files located in the directory that FMS was invoked from.

The following Log4j configuration illustrates using the operating system logging infrastructure.

```
! Set root category threshhold to INFO and log to Syslog
log4j.rootCategory=INFO, Syslog
! Syslog is set to be a SyslogAppender.
log4j.appender.Syslog=org.apache.log4j.net.SyslogAppender
! Syslog uses PatternLayout.
log4j.appender.Syslog.layout=org.apache.log4j.PatternLayout
! The logging pattern to use, see below.
log4j.appender.Syslog.layout.ConversionPattern=%-5p %m%n
! Set Syslog properties.
log4j.appender.Syslog.SyslogHost=localhost
log4j.appender.Syslog.Facility=INFO
log4j.appender.Syslog.FacilityPrinting=true

! If Syslog4j is installed:
! Syslog4j provides client (UDP, TCP, TCP over SSL/TLS, Native Unix syslog, and Unix socket) and server (TCP, TCP over SSL/TI
! Also see http://www.productivity.org/projects/syslog4j/old/
! Syslog is set to be a Syslog4jAppender.
! log4j.appender.Syslog=org.productivity.java.syslog4j.impl.log4j.Syslog4jAppender
! log4j.appender.Syslog.Protocol=unix_syslog
! log4j.appender.Syslog.Threshold=INFO
! log4j.appender.Syslog.layout=org.apache.log4j.PatternLayout
! log4j.appender.Syslog.layout.ConversionPattern=%-5p %m%n
```

The following Log4j configuration illustrates using a file based logging infrastructure.

```
!Set root category threshhold to INFO and log to dest2
log4j.rootCategory=INFO, dest2
! Load a Rolling File Appender as our destination
log4j.appender.dest2=org.apache.log4j.RollingFileAppender
! This appender will only log messages with priority equal to or higher than
! the one specified here
log4j.appender.dest2.Threshold=INFO
! Specify the logging file name
log4j.appender.dest2.File=/var/log/fms/fms.log
! Don't overwrite, append
log4j.appender.dest2.Append=true
log4j.appender.dest2.layout=org.apache.log4j.PatternLayout
! The logging pattern to use, see below.
log4j.appender.dest2.layout.ConversionPattern=%d{dd MMM yyyy HH:mm:ss,SSS} %-5p %m%n
! Control the maximum log file size
log4j.appender.dest2.MaxFileSize=512KB
! Keep backup file(s) (backups will be in filename.1, .2 etc.)
log4j.appender.dest2.MaxBackupIndex=10
```

The log file formate may be customised as follows

```
<date> <granularity> [<thread>,<classname>:<lineNumber>]
<message> [<internalTrackingID>, <messageTrackingID>]
```

The logging format can be modified by editing the required `log4j.properties` file.

Table C-1. Table of log4j arguments with their descriptions

Argument	Description
%n	A new line follows.
%m	Log message. <message> <InternalTrackingID> (if applicable) <messageTrackingID> (if applicable)
%p	Priority. The Logging Level of this message.
%r	millisecs since program started running.
%c	name of your category (logger),%c{2} will output the last two components.
%t	Name of current thread.
%d	Date and time, also %d{ISO8601}, %d{DATE}, %d{ABSOLUTE}, %d{HH:mm:ss,SSS}, %d{dd MMM yyyy HH:mm:ss,SSS}
%l	Long. Equivalent to %F%L%C%M
%F	Java source file name.
%L	Java source line number.
%C	Java class name, %C{1} will output the last one component.
%M	Java method name.

Table C-2. Log Message Components.

Component	Description
<message>	The Log Message
<InternalTrackingID>	Used by the FMS server to keep record of where messages were sent from so that returned messages will have correct destination. This is required as the server does not only deal with one message at a time.
<messageTrackingID>	Only applicable in situations where an actual message that was sent to or received by the server(eg: an Invoice) was associated with a particular log message.

Levels of Granularity

Table C-3. Log Granularity

Logging Level	Semantic Level
TRACE	Log TRACE messages and include all messages as below.
DEBUG	Log DEBUG messages and includes all messages as below.
INFO	Log INFO messages and include all messages as below.

Logging Level	Semantic Level
WARN	Log WARN messages and include all messages as below.
ERROR	Log ERROR messages and include all messages as below.
FATAL	Quiet. Only logs fatal errors.

Appendix D. Extending Schema Content Support

What is a PIP®?

RosettaNet® Partner Interface Processes (PIP®s) are specialized system-to-system XML-based dialogs that define business processes between trading partners. Each PIP® specification includes a business document with the vocabulary, and a business process with the choreography of the message dialog.

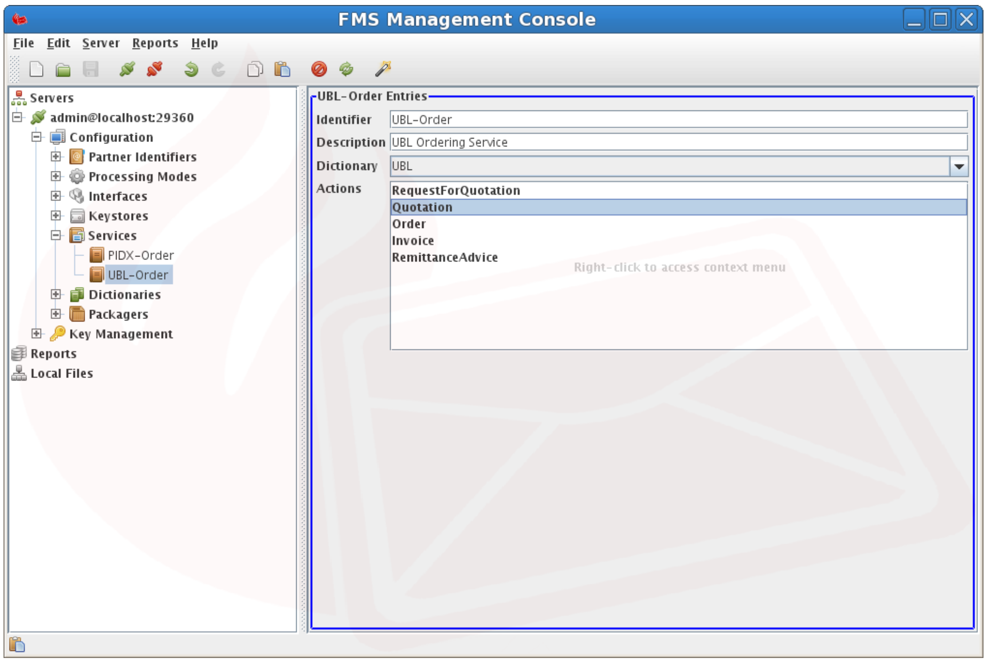
Currently only XML Schema is supported as a vocabulary for the XML based dialogs.

Which Schemas are supported by default?

- Petroleum Industry Data Exchange PIDX, visit http://www.pidx.org/ for more information.
- Universal Business Language UBL The UBL defines a set of standardised XML based vocabularies for business documents in the order-to-invoice cycle. The current 2.0 version of UBL is maintained by the OASIS Universal Business Language Technical Committee. See http://www.oasis-open.org/committees/ubl for further information.
- OAGIS BOD The OAGIS® BOD XML version 9_5 schema documents. Copyright (c) Open Applications Group. All Rights Reserved. See http://www.oagi.org for further information.

Adding a new Schema

Adding a new Schema definition to the configuration is possible through the FMC. Assuming the FMC is open and visible click in the Schema Mapping list on the left hand side of the interface. By doing so the following screen should be in view.



Identifier: UBL-Order
Description: UBL Ordering Service
Dictionary: UBL
Actions: RequestForQuotation, Quotation, Order, Invoice, RemittanceAdvice

Right-click to access context menu

Figure D-1. Schema Mapping UBL

By accessing (right click) the context menu from the list of Schema Entries it is possible to perform a number of tasks

To edit a schema entry just click on the Edit menu item in the context menu to open a popup window.

Table D-1. Schema Entry Explanation

Key	Description
Name	This is the name of the Schema, this would be the same name from the client invocation argument `Schema Type`
Schema	The value is the path to the XML Schema provided for this Schema Type (<The XML Schema>).
Action	The Global Business Action Code as defined by the Schema
To Role	The Global Partner Role Classification Code as defined by the Schema
To Service	The Global Business Service Code as defined by the Schema
From Role	The Global Partner Role Classification Code as defined by the Schema
From Service	The Global Business Service Code as defined by the Schema
Version	The version of this Schema
Code	The Schema Code, normally provided by RosettaNet®

Appendix E. Collaboration Protocol Profile/Agreement

CPP/A Definition

Collaboration Protocol Profile/Agreement (CPP/A) provides interoperability between two parties even though they may use application software and run-time support software from different vendors. The Collaboration Protocol Profile (CPP) defines message-exchange capabilities and the business collaborations that it supports. The Collaboration Protocol Agreement (CPA) defines the way two parties will interact in performing the chosen business collaboration.

CPP/A Configuration

ebXML is currently the only messaging implementation that supports the CPP/A standard. The CPP/A may either be retrieved from a URL or from an ebMS Registry Server such as Omar® or a supported SOAP Repository.

A static CPA may be defined in the PartnerIdentifier section of the FMC with an override switch to ignore a client provided CPA when in a Production environment.

Refer to the Section called *Invocation* in Chapter 5 for the client CPA arguments -o and -C.

Appendix F. Examples and Test cases

ebXML Example

The following example illustrates using FMS between two business trading partners using the ebXML V3.0 message protocol. The example takes a step by step approach in configuring, establishing a link, and communicating a business message between the producer and a consumer trading partner. The business document (invoice) is based on the Universal Business Language UBL version 2.0 format.

The example assumes that the producer machine has a `TCP/IP` number of `192.168.0.102` and that the consumer machine has a `TCP/IP` number of `192.168.0.128`.

ebXML Configuration

The first step is to install FMS as discussed in the Section called *FMS Installation* in Chapter 2. Once FMS is installed and running invoke the FMC as discussed in Chapter 3. Leave any settings not included below at default or blank.

Producer Configuration

The following interface configuration steps are required on the producer side.

1. Set up the keystore including private and public keys for the producer as per the Section called *Keystore Setup and Examples* in Chapter 2.
2. Open the `Interface Configuration` tab in the FMC and select `HTTP - ebXML - UBL` in the `Interface Name` pane.
3. Select `Enabled` in the corresponding configuration form to the right.
4. Select the required `Payload Security Level` eg. `SIGN_ENCRYPT`.
5. Select the required `Transport Security Level` eg. `SIGN_ENCRYPT`.
6. Select the name of the output directory eg. `delivered-content`.
7. Select the `Usage` required eg. `TEST`.
8. Only enable `Non-Repudiation` if the database has been installed and configured as detailed in the Section called *Database Server* in Chapter 2.
9. Select the `ebXML RemoteIn` button in the `HTTP -ebXML - UBL Connections` form and set the `HOST` attribute to the `TCP/IP` number of the machine on which the producer FMS is hosted eg. `192.168.0.102`. Both `TCP/IP V4` and `TCP/IP V6` are supported. If DNS is enabled then set `HOST` to the domain name.

The following partner configuration steps are required on the producer side for the local partner identifier.

1. Select partner identifier `000000000` in the `Partner Identifiers` pane in the `Interface Configuration` section of the FMC.
2. Set the `Identifier Value` to `Producer` in the corresponding form. The partner identifier in the `Partner Identifiers` pane will change accordingly.
3. Set the `Partner Type` to `LOCAL PARTNER` from the drop down list.

The following partner configuration steps are required on the producer side for the remote partner identifier.

1. Select partner identifier `999999999` in the `Partner Identifiers` pane in the `Interface Configuration` section of the FMC.
2. Set the `Identifier Value` to `Consumer` in the corresponding form. The partner identifier in the `Partner Identifiers` pane will change accordingly.
3. Set the `Keystore Alias` to `consumer_fmsrns`.
4. Set the `Endpoint URL/MPC` to the url of the remote host eg.`https://192.168.0.128/ebXML`.
5. Set the `Partner Type` to `REMOTE PARTNER` from the drop down list.

The following processing mode configuration steps are required on the producer side.

1. Select the Processing Modes pane in the Interface Configuration section of the FMC.
2. Select the General form and set the Identifier attribute to inv-producer-consumer. This can be set to any text string but must correspond with the consumer processing mode.
3. Set the Agreement to blank unless a CPA is required.
4. Set the Conversation ID to 1.
5. Set the Initiating Partner Party to Producer.
6. Set the Responding Partner Party to Consumer.

Select the Request tab in the P-Mode Events form and ensure that the following settings are configured. Note that these are also known as P-Mode Legs.

1. Set the protocol Address/Endpoint to https://192.168.0.128/ebXML.
2. Set the business information Service to Service-Mapping-UBL from the drop down list.
3. Set the business information Action to Invoice.
4. Right click in the business information Payload Profiles field and select New to open the Payload Profile selection window.
5. Select UBL-Invoice-2.0.xsd from the Schema File drop down list.
6. Select application/xml from the MIME Type drop down list.
7. Select EXPECTED from the Requirement drop down list.
8. Select the Done button to close the Payload Profile selection window.
9. Select the Error Handling form by scrolling down to the bottom of the P-Mode-Events request tab form.
10. Set the Sender Errors To to https://192.168.0.102/ebXML.
11. Set the Receiver Errors To to https://192.168.0.128/ebXML.
12. Set all 4 Notify selections.

Once all the above has been configured select the Save button. Then select the Administration pane and load the configuartion into FMS as follows.

1. Select the Administration menu option and select Open Admin Connection.
2. Select the Administration menu option and select Connect to Logger.
3. Select the Tools menu option and select Reload Connections.

Consumer Configuration

The following interface configuration steps are required on the consumer side.

1. Set up the keystore including private and public keys for the consumer as per the Section called *Keystore Setup and Examples* in Chapter 2.
2. Open the Interface Configuration tab in the FMC and select HTTP - ebXML - UBL in the Interface Name pane.
3. Select Enabled in the corresponding configuration form to the right.
4. Select the required Payload Security Level eg. SIGN_ENCRYPT.
5. Select the required Transport Security Level eg. SIGN_ENCRYPT.
6. Select the name of the output directory eg. delivered-content.
7. Select the Usage required eg. TEST.
8. Only enable Non-Repudiation if the database has been installed and configured as detailed in the Section called *Database Server* in Chapter 2.
9. Select the ebXML RemoteIn button in the HTTP -ebXML - UBL Connections form and set the HOST attribute to the TCP/IP number of the machine on which the consumer FMS is hosted eg. 192.168.0.128. Both TCP/IP V4 and TCP/IP V6 are supported. If DNS is enabled then set HOST to the domain name.

The following partner configuration steps are required on the consumer side for the local partner identifier.

1. Select partner identifier `000000000` in the `Partner Identifiers` pane in the `Interface Configuration` section of the FMC.
2. Set the `Identifier Value` to `Producer` in the corresponding form. The partner identifier in the `Partner Identifiers` pane will change accordingly.
3. Set the `Keystore Alias` to `producer_fmsrns`.
4. Set the `Partner Type` to `REMOTE PARTNER` from the drop down list.

The following partner configuration steps are required on the consumer side for the remote partner identifier.

1. Select partner identifier `999999999` in the `Partner Identifiers` pane in the `Interface Configuration` section of the FMC.
2. Set the `Identifier Value` to `Consumer` in the corresponding form. The partner identifier in the `Partner Identifiers` pane will change accordingly.
3. Set the `Partner Type` to `LOCAL PARTNER` from the drop down list.

The following processing mode configuration steps are required on the consumer side.

1. Select the `Processing Modes` pane in the `Interface Configuration` section of the FMC.
2. Select the `General` form and set the `Identifier` attribute to `inv-producer-consumer`. This can be set to any text string but must correspond with the producer processing mode.
3. Set the `Agreement` to blank unless a `CPA` is required.
4. Set the `Conversation ID` to `1`.
5. Set the `Initiating Partner Party` to `Producer`.
6. Set the `Responding Partner Party` to `Consumer`.

Select the `Request` tab in the `P-Mode Events` form and ensure that the following settings are configured. Note that these are also known as `P-Mode Legs`.

1. Set the protocol `Address/Endpoint` to `https://192.168.0.102/ebXML`.
2. Set the business information `Service` to `Service-Mapping-UBL` from the drop down list.
3. Set the business information `Action` to `Invoice`.
4. Right click in the business information `Payload Profiles` field and select `New` to open the `Payload Profile` selection window.
5. Select `UBL-Invoice-2.0.xsd` from the `Schema File` drop down list.
6. Select `application/xml` from the `MIME Type` drop down list.
7. Select `EXPECTED` from the `Requirement` drop down list.
8. Select the `Done` button to close the `Payload Profile` selection window.
9. Select the `Error Handling` form by scrolling down to the bottem of the `P-Mode-Events request` tab form.
10. Set the `Sender Errors To` to `https://192.168.0.102/ebXML`.
11. Set the `Receiver Errors To` to `https://192.168.0.128/ebXML`.
12. Set all 4 `Notify` selections.

Once all the above has been configured select the `Save` button. Then select the `Administration` pane and load the configuartion into FMS as follows.

1. Select the `Administration` menu option and select `Open Admin Connection`.
2. Select the `Administration` menu option and select `Connect to Logger`.
3. Select the `Tools` menu option and select `Reload Connections`.

Sending an Invoice

Once the above configuration settings have been made using the FMC and subsequently loaded into FMS, business documents corresponding to the configuration settings can be communicated from the producer trading partner to the consumer trading partner. This may be achieved using the `fms-client` utility from the FMS installation directory as follows.

```
java -jar client/fms-client.jar -c inv-producer-consumer -d consumer -e request \
  -f test/UBL/xml/UBL-Invoice-2.0-Example.xml -P 29450 -O finest
```

where the client utility command line options are described in the Section called *Invocation* in Chapter 5.

RosettaNet Example

Fictional Example for Configuration and Communication between two business partners using FMS and RosettaNet®.

Bob runs a manufacturing company, while Mary runs a retail store which distributes and sells Bob's products.

Since Bob's manufacturing company runs a Business 2 Business (B2B) integration server which supports RosettaNet, Mary has realized that the quoting and product ordering process could be streamlined by using the B2B integration specification RosettaNet provided by FMS-Starter Edition.

Mary initiates the B2B process by communicating with Bob and discussing the integration. Bob sends Mary:

- his supported Parter Interface Process (PIP) list
- the RosettaNet business dictionary for a glossary of terms
- his DUNS number (`000000001`)
- his physical location
- his B2B integration server hostname (`https://ebusiness.bob.org/RosettaNet`)
- as well as his Public Key (Certificate).

Mary subscribes to the RosettaNet PIP cluster 3 which is `'Order Management'` which controls processes such as:

```
Cluster 3: Order Management
  Segment 3A: Quote and Order Entry
    PIP3A1: Request Quote
    PIP3A2: Request Price and Availability
    PIP3A3: Request Shopping Cart Transfer
    PIP3A4: Request Purchase Order
    PIP3A5: Query Order Status
    ...
```

Mary is only really interested in `PIP3A1` and `PIP3A4` but all of the other options are available in this PIP cluster.

Mary then applies for a Data Universal Numbering System or DUNS number from http://www.dnb.com/US/duns_update/ for use with the RosettaNet specification which uniquely identifies her company by a number. Mary receives the DUNS number `000000002` for her company.

Mary now installs the FMS distribution on her company's server. She refers to the manual for installation steps:

1. Install FMS
2. Obtain a licence key from the FMS distributer
3. Generate a Private and Public Key pair in the keystore 'certs'

4. Import Bob’s Public Key (Certificate) into the server keystore ’certs’ under the alias ’bob.alias’
5. Add Bob's PartnerIdentifier (`000000001`), alias (`bob.alias`), and endpoint hostname (`https://rn.bob.org`) to the PartnerIdentifier section in the configuration file `ConnectionConfiguration.xml` for return mapping purposes
6. Copy the unrestricted Java policy files into the `$JAVA_HOME/lib/security/` directory to allow for 1024 bit encryption keys
7. Configure the logging system
8. Start the server and check logs for error messages
9. Export the server's Public Key (Certificate) for use with the client application during SSL transmission
10. Copy the server's Public Key to the client machine running the Management Application that will be sending the RosettaNet requests
11. Install the Client distribution on the client machine
12. Import the server's Public Key (Certificate) into the client machine's keystore
13. Configure the client script with the correct DUNS numbers (Recipient `'000000001'` and Sender `'000000002'`), Bob's server hostname `'https://rn.bob.org'`, the local server hostname to connect to, and the physical location names for both Mary and Bob
14. Send Mary's Public Key (Certificate), her physical location, and her DUNS number to Bob who then inputs them into his system
15. Modify the Business Application to generate a Quote into the XML standard provided by the `PIP3A1` specification
16. Execute the client script from the Business Application with the XML, Message Tracking ID, supporting documents, and Schema Type (`'Request Quote'`) as arguments and wait for a response from Bob's B2B integration server

Mary then repeats the last two steps but with `PIP3A4` for `'Request Purchase Order'` for ordering of stock.

Mary now enjoys a seamless quoting and ordering experience with Bob's Manufacturing Company, thereby increasing productivity and reducing stress levels at work.

Appendix G. Frequently Asked Questions

Server

1. Starting the server fails with a configuration parse error where a configuration element in `ConnectionConfiguration.xml` is valid but with an error similar to the following

   ```
   main: Configuration Parse failure: L529:C31 cvc-complex-type.2.4.d: Invalid content was found starting with element ’pm
   ```

   Ensure that all elements are ordered according to the schema files in the FMS install schema/FMS directory.

2. Sending a message with required encryption using the ’-enc alias’ command line argument fails with following error message

   ```
   SEVERE com.sun.xml.wss.impl.misc.DefaultSecurityEnvironmentImpl getCertificate() - WSS0221: Unable to locate matching c
   ```

   This is an indication that the alias matching the required partner public certificate in the keystore to encrypt an outbound message cannot be located in the keystore. Ensure that the certificate is installed.

3.

4. Is it possible to restrict TLS ciphers ?

   FMS does not have a configuration option for enabling and disabling TLS ciphers as that is handled by the default `java.security` configuration file which will override any setting configured in applications.

5. I upgraded my Java version to 1.8.0 release 222 or later and now the FMS server starts but just hangs.

   This is due to a missing class file within the standard java library. A typical error output in the FMS server logs could look as follows:

   ```
   Exception in thread "main" java.lang.NoClassDefFoundError: sun/security/validator/KeyStores
   at com.flame.utils.FMSTrustManager$InternalX509TrustManagerImpl.<init>(FMSTrustManager.java:254)
   ```

   Either rollback the upgrade for Java to a previous working release or add the missing class file as follows:

   ```
   # Extract KeyStores.class from openjdk 8u212 and import into update 8u222 rt.jar
   cd ~/tmp # directory where previous version of sun/security/validator/KeyStores.class was extracted
   cp -pv /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar \
     /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar.8u222.backup
   jar uf /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar \
     sun/security/validator/KeyStores.class
   ```

6. How many messages can the server handle concurrently?

   This is largely dependent on the hardware deployed, the speed of the network and the size of the messages including any attachments. Suffice it to say that it can handle significant message volumes per minute.

7. What messaging protocols does the FMS support.

   Currently the ebXML Messaging Services Version 3, the AS4 Profile of ebMS 3.0 Version 1.0, the IETF STD69 Extensible Provisioning Protocol (EPP), and RosettaNet® Implementation Framework Version 2.

8. I've installed the FMS server but I can't start it up and have no idea what is happening!

   Locate the log file typically located at `/var/log/fms/fms.log` or other log files in the same directory for Linux and unix and and at `Programs/FMS/log/serviceinfo.log` or other log files in the same directory for Windows.

   Ensure that the certificate keystore has been setup, as per the Section called *Importing Public Keys (Certificates) into the Server Keystore* in Chapter 2 and that syslog is configured before startup as per the Section called *Syslog Configuration* in Chapter 3.

9. I've setup the syslog handler but I don't see anything in the log file!

   Ensure that syslog itself is configured correctly, refer to the Section called *Syslog Configuration* in Chapter 3

10. I've started up the FMS server and see that `/var/log/fms/fms-stderr.log` contains the following error.

    ```
    [INFO] is an unknown syslog facility. Defaulting to [USER].
    ```

    Ensure that the syslog configuration matches the `log4j.appender.Syslog.Facility` configuration in file `log4j.properties`. Refer to the Section called *Syslog Configuration* in Chapter 3 for further details.

11. I've started up the FMS server and see that `/var/log/fms/fms-stderr.log` contains the following error.

    ```
    log4j:WARN No appenders could be found for logger (com.flame)
    log4j:WARN Please initialize the log4j system properly
    ```

    Ensure that the `log4j.properties` configuration is installed in the same directory as the `fms.jar` file. Refer to the Section called *FMS Logging Configuration* in Chapter 3 for further details.

12. I can't make a connection to the remote server! Help!

    I get FMS ERROR: Connection interface failed to bind to address when starting FMS.

    Make sure the remote server host can be pinged first, if so, then attempt to `telnet` or `openssl` to that server on the port specified. If the telnet or openssl session succeeds then it should work, if not then ensure that a firewall is not blocking the port and that the server is indeed listening by checking the logs for any errors. Verify SSL Certificates have been set up correctly.

    ```
    telnet remote.server.com 443
    openssl s_client -connect remote.server.com:443
    ```

13. I receive a cipher suite error in the logs or on FMC when trying to connect to the FMS server.

    SSLHandshakeException occurred while negotiating a connection: no cipher suites in common.

    Ensure that the java `java.security` file entry for `jdk.tls.disabledAlgorithms` on both the local and remote sides do not include the required protocol. For FMC revisions prior to 5.4.2 build 7 the required transport protocol is TLSv1.

    Ensure that the FMS server private certificate for the relevant listener is valid and has not expired.

14. I receive the following error in the FMS logs when trying to connect to the server using the fmsconf utility.

    SSLHandshakeException occurred while negotiating a connection: Client requested protocol TLSv1.2 is not enabled or supported in server context.

    SSL Handshake Exception occurred while listening from 'localhost/127.0.0.1'. SSL_VERSION = 'TLSv1.3', SSL_NEED_CLIENT_AUTH = 'false' : Error is `'The client supported protocol versions [TLSv1.2, TLSv1.1, TLSv1] are not accepted by server preferences [TLS13]'`.

    Ensure that the FMS server configuration listener property SSL_VERSION is set to include TLSv1.2. Also ensure that the default `java.security` file does not have TLSv1.2 disabled. Note that this property can be set up to TLSv1.3 if supported by the `java` runtime.

15. I receive the following error when trying to connect to the FMS server using the fmsconf utility.

    `SSL routines:ssl3_read_bytes:sslv3 alert handshake failure:s3_pkt.c:1487:SSL alert number 40`

    Ensure that the FMS server private certificate used by the Administration listener is valid and has not expired.

16. I receive the following error when trying to connect to the FMS server using the fmsconf utility.

    `ssl handshake failure`

    This indicates that the server has client authentication set to true for the Administration interface and therefore requires the necessary certificates on the fmsconf side. For details on key and certificate generation invoke fmsconf as follows

    ```
    fmsconf -H
    ```

17. I receive the following error when using the fmsconf `__FMS_Admin_ListCerts` command as follows

```
fmsconf -x '__FMS_Admin_ListCerts certs JKS'
OK
ERROR
An error occurred while reading the keystore: Keystore was tampered with, or password was incorrect
```

The `__FMS_Admin_ListCerts` requires a password field to read the contents the certificate store as follows

```
fmsconf -x '__FMS_Admin_ListCerts certs JKS password'
```

The keystore name (just the file name excluding the directory), type and password must be the same as specified for the keystore in the FMS `ConnectionConfiguration.xml` file. Eg.

```
<cc:keystoreRef cc:ID="default">
  <cc:name>certs</cc:name>
  <cc:type>JKS</cc:type>
  <cc:pass>password</cc:pass>
</cc:keystoreRef>
```

18. I have SSL problems connecting to a remote server using the fmsclient.as4.jar and would like to see more detail in the logs

    Invoke the client as follows to provide low level debugging information.

```
java -Djavax.net.debug=ssl:handshake:verbose \
  -Dcom.sun.xml.wss.impl.MessageConstants.debug="true" -jar fmsclient.as4.jar
```

19. I can't make a connection to a remote FMS server! Help!

    Has the remote FMS server started up correctly. Look carefully at the logs of the remote FMS server for any ERROR notifications and address these first.

    If the local FMS reports an `IOException while writing error` followed by an `Unrecognized SSL` message then it is likely that the local FMS server is attempting to send an encrypted message to a remote server that is not expecting an encrypted message.

20. I get the following message in my server log when initiating a request or executing a synchronous trigger.

    [Fatal Error] :1:1: Content is not allowed in prolog.

    What is in all likelihood happening is that the the content being handed to the server is not valid XML.

21. I get the following message in my log when attempting to send to a remote destination using HTTPS.

    `PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target`

    Ensure that the associated root certificate is up to date in the Java cacerts keystore and confirm that the certificate being used validates against it's root certificate path, and that the root certificate exists and has not expired or been revoked.

    The remote server SSL certificate can also be validated as follows using the `openssl` utility

```
openssl s_client -connect remote.as4.server.com:443 -showcerts
# or if testing with a new root certificate
openssl s_client -CAfile /path/to/RootCA.pem -connect remote.as4.server.com:443 -showcerts
```

or as follows using the FMS light client

```
java -Djavax.net.debug=ssl:handshake:verbose .. fmsclient.as4.jar ...
```

which may provide the following detail

```
***
%% Invalidated:  [Session-1, TLS_RSA_WITH_AES_256_CBC_SHA256]
main, SEND TLSv1.2 ALERT:  fatal, description = certificate_unknown
main, WRITE: TLSv1.2 Alert, length = 2
main, called closeSocket()
main, handling exception: javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: \
  PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certifica
```

```
main, called close()
main, called closeInternal(true)
2019-06-17 15:57:06,006 SEVERE com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnection post() \
 - SAAJ0009: Message send failed
2019-06-17 15:57:06,007 SEVERE com.flame.client.as4.api.Client transmit() - PKIX path building failed: \
  sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path \
 to requested target
```

22. I get a message send fail in my log when attempting to send to a remote destination using HTTPS.

```
2019-08-01 21:40:05,122 SEVERE com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnection post() \
 - SAAJ0009: Message send failed
2019-08-01 21:40:05,123 SEVERE com.flame.client.as4.api.Client transmit() - Received fatal alert: handshake_failure
```

    This error occurs when the remote endpoint is not available or not in a state where it can not accept a secure connection.

23. I get the one of following messages in my server log when attempting to listen from a remote destination.

    `SSL Handshake Exception occurred while listening from 'remote_host/remote_host_address' : Received fatal alert: certificate_unknown    SSL Handshake Exception occurred while listening from 'remote.host' : null cert chain.`

    Ensure that the `SSL` client certificate chain is in the server truststore and that the certificate validates against it's root certificate path in the server truststore. The certificate chain is presented to the remote client from the server and the client can only respond with the appropriate certificate if the certificate authority is correctly presented to it. The `null cert chain` error will occur when the listener property `SSL_NEED_CLIENT_AUTH` is set to `true` for the server listener interface on which an incoming connection occurs.

    The `certificate_unknown` error can also occur if the keystore contains two private keys with one of the private keys having expired.

    `TLSv1` requires a certificate in its keystore that was signed directly or indirectly by any of the signers mentioned in the `SSL CertificateRequest` handshake message.

    For `TLSv1.1` or later if a `certificate_authorities` list is empty then the client MAY send any certificate of the appropriate `ClientCertificateType`, unless there is some external arrangement to the contrary.

    Also see the paragraph under `certificate_authorities` at https://tools.ietf.org/html/rfc4346#section-7.4.4.

24. I get the following error in my server log when attempting to start the server when I have two or more private keys in the keystore.

    `Failed to load FMS KeyManager. Could not recover key with password "keypassword": Cannot recover key`

    Ensure that all private keys share the same password.

25. I can't start the server because the `java` command cannot be found!

    Ensure Java is installed and setup correctly. Open a console and type `java -version` it should return version 1.8.0 or higher.

26. I'm getting validation errors on the RosettaNet DTD headers (Preamble, Delivery, and Service), but my XML looks perfectly valid, whats wrong?!

    The DTD validation of RosettaNet headers is very strict. To force validation to relax disable it by setting the configuration option `ROSETTANET_HEADER_VALIDATION` to `false`. This configuration option is located in the section `Package Configuration` in the `ConnectionConfiguration.xml` file.

27. I receive namespace errors on the RosettaNet DTD headers (Preamble, Delivery, and Service), but my XML looks perfectly valid, whats wrong?!

    The DTD namespace implementation is probematic and should be disabled, by setting the configuration option `ROSETTANET_HEADER_NAMESPACE_AWARE` to `false` it will be disabled, this configuration option is located in the section `Package Configuration` in the `ConnectionConfiguration.xml` file.

28. I get a "com.flame.client.as4.api.Client transmit SEVERE: For input string:" error when specifying an `IPv6` address for the host argument.

    Ensure that the `IPv6` address is encapsulated within square brackets as follows

    ```
    -h https://[fe80:8::106a:9125:18a9:9f64%en0]:6443/as4s
    ```

29. I get a "java.net.BindException?: Cannot assign requested address exception"

    If you get an exception like this, then, switch to IPv4 by assigning an IPv4 address in the configuration file. This is quite likely due to trying to use IPv6 in Linux but Sun's JDK (pre version 6) has a bug.

30. I get a `SecurityException: cannot verify signature block file META-INF/BCKEY` exception on the server.

    This exception generally occurs when verifying the signature of a signed jar. Try restarting the server to reload the library jar files.

31.

    I get a `java.security.NoSuchProviderException: JCE cannot authenticate the provider BC` error log entry.

    This exception generally occurs when verifying the signature of a signed jar. Try restarting the server to reload the library jar files.

32. I get an `UnrecoverableKeyException` when starting the server yet the alias and the password are correct.

    Ensure that the password is not shared in that keystore, the SSL Key management requires a master key for encryption, this master key's password must be unique.

33. I get an exception `Key inappropriate for algorithm` or `Illegal key size or default parameters`. What causes it and how do I fix it?

    This means that the unrestricted Java policy files have not been installed. Refer to the Section called *Encryption* in Chapter 3

34. Various default passwords are used in generating keys and keystores. Do I need to stick to these?

    It is highly recommended that no default passwords are used. Remember to first change the configuration files before invoking the key generation utility and the server.

35. I have a problem importing certificates supplied in `pkcs7` format into our java key store (`JKS`).

    It seems that to import it using the Java `keytool` you need to use the exact alias as supplied in the `pkcs7` format but which is unfortunately not visible when viewing the certificate.

    Use the following process to import it.

    Convert supplied `pkcs7` certificate to an `x509` certificate

    ```
    openssl pkcs7 -print_certs -in pkcs7certs.txt -out x509cert.cer
    ```

    Import supplied converted `x509` certificate

    ```
    keytool -import -trustcacerts -keystore certs -storepass changeit -file x509cert.cer -alias remotepartneralias
    ```

Client and FMC

1. I receive a 'Response: No appropriate protocol (protocol is disabled or cipher suites are inappropriate). Server might require mutual SSL authentication' message when trying to connect to and FMS server from FMC.

   Ensure that the client side java `java.security` file entry for `jdk.tls.disabledAlgorithms` does not include the required protocol. For FMC revisions prior to 5.4.2 build 7 the required transport protocol is TLSv1.

   Ensure that the FMS server private certificate for the relevant listener is valid and has not expired.

2.

I get the following SAXParseException when trying to send a message.

```
2019-04-30 13:30:04,637 WARNING com.flame.client.as4.AS4ClientAPI <init>() \
  - Preparing message for sending to host https://ebis-int.open-grid-europe.com/ibis/as4/sync
2019-04-30 13:30:04,637 INFO com.flame.client.as4.api.Client createEnvelope() \
  - Creating Envelope with messageID 'AS4-16A6E0248DC-D7C8C@undefined'
2019-04-30 13:30:04,669 INFO com.flame.client.as4.api.Client createPayload() \
  - Creating Payload for messageID 'AS4-16A6E0248DC-D7C8C@undefined'
[Fatal Error] :1:1: Content is not allowed in prolog.
ERROR:  'Content is not allowed in prolog.'
2019-04-30 13:30:04,675 SEVERE com.flame.client.as4.api.Client transmit() \
  - org.xml.sax.SAXParseException; lineNumber: 1; columnNumber: 1; Content is not allowed in prolog.
```

Ensure that all arguments are correctly set when invoking the AS4 light client. Eg. ensure that arguments such as `'-ag'` are followed by a value which in this case would be the Agreement Reference.

3. When trying to send a message with the client process I get an unknown connection issue message.

   Try to connect to the remote FMS server using `openssl` or `telnet` on port `443`. If no log messages appear at the remote FMS then the remote server is not listening on port `443`. Ensure that no other process such as a web server is listening on port `443` on the remote server. Also ensure that the user starting up FMS has permissions to bind to port `443`.

4. I can't start the client because the Java command cannot be found!

   Ensure Java is installed and setup correctly. Open a console and type `java -version` it should return version 1.8.0 or higher.

AS4 Client

1.

   When trying to send a message with the client process I get the following response

   PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target

   The client has received an SSL certificate chain that it does not recognise. This implies that the certificate validation path does not exist in the default `cacerts` java keystore. If the certificate exists in the client keystore then try re-sending with the `'-st'` command line switch set to the client keystore.

2. When trying to send a message with the client process I get the following response

```
com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnection post
SEVERE: SAAJ0009: Message send failed
SOAP Error: : com.sun.xml.internal.messaging.saaj.SOAPExceptionImpl: Message send failed:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: \
  unable to find valid certification path to requested target:
```

   Export the server certificate and install it on the client. See the Section called *Keystore Configuration* in Chapter 2 for further details on this.

3.

   How do I define the security context file as required by the as4 client?

   The client security context as set using the '`-sc security.xml`' command line option determines the signing and encryption security of any sent messages. Full details on the syntax and settings are available at http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/XWS-SecurityIntro4.html

4.

   I get the following error in my client log when attempting to send to a remote destination.

   `com.flame.client.as4.api.Client transmit - SEVERE: Received fatal alert: bad_certificate` or `com.flame.client.as4.AS4ClientAPI <init> - INFO: ClientException - Received fatal alert: bad_certificate`.

Ensure that the `SSL` client certificate chain has been provided and installed in the server truststore and that the certificate validates against it's root certificate (if a CA signed cert) path in the server truststore. The certificate chain is presented to the remote client from the server and the client can only respond with the appropriate certificate if the certificate authority (CA) is correctly presented to it. The `bad_certificate` error will occur when the client continues a connection to a remote server without presenting it's certificate but when client authentication is required.

Appendix H. Version History

This section details the differences between the various versions of FMS.

FMS Version History

- Version: 5.4.3 Release 3

  1. fmsconf - updated to use python3. This can be overriden by setting environment variable PYTHON to the required version before invoking fmsconf.

  Version: 5.4.3 Release 2

  1. Server - removed install requirement for 'PostgreSQL'.
  2. Server - removed 'fmsas4lc' from the server installation. It can be installed as part of the client distribution.
  3. Server 5.4.3 - default keystore type is now '`PKCS12`'. This means that all keytool examples in this document require '-storetype PKCS12' instead of '-storetype JKS' and changing '-keypass fmsrns' which must either be removed or made the same as '-storepass changeit'.

  Version: 5.4.3 Release 1 pre-release

  1. Server - Improved warning log message for '`Referenced partner not found`'.
  2. Server - Migrate from `log4j` to `log4j2`.
  3. Server Partner Configuration `ConnectionConfiguration.partnerIdentifier.organisation` - added optional organisation element.
  4. Server Partner Configuration `ConnectionConfiguration.partnerIdentifier.address` - added optional address element.
  5. Server Partner Configuration `ConnectionConfiguration.partnerIdentifier.phone` - added optional phone element.
  6. Server Partner Configuration `ConnectionConfiguration.partnerIdentifier.email` - added optional email element.
  7. Server Partner Configuration `ConnectionConfiguration.partnerIdentifier.email` - added optional email element.
  8. FMC - Bug fixes and improved certificate export functionality for both DER and PEM formats.

  Version: 5.4.2 Release 9

  1. Server - removed install requirement for 'PostgreSQL'.
  2. Server - removed 'fmsas4lc' from the server installation. It can be installed as part of the client distribution.

  Version: 5.4.2 Release 8

  1. 5.4.2-8 p-mode search caching fix (unreproducible).
  2. documentation - general editorial updates,
  3. FMC - soften TLS protocol - default to TLSv1.2.

  Version: 5.4.2 Release 7

  1. FMC - add `html` directory to doc and include support jars including `webservices-rt` to the management console build distributions.
  2. FMC - general improvements including keystore management.
  3. documentation - general editorial updates including reference to new directory structures,

  Version: 5.4.2 Release 6

  1. Server - improved logging.

2.

   fmsconf - TLS_ARG now defaults to an empty string.

3.

   Server - fixed `insert into request` query syntax error. This occured in previous versions if the `interfaceConfig.databaseConfig.enabled` property is set to `true` resulting in failed inbound message processing.

4.

   Server - corrected audit log tables `/home/fms/schema/postgresq/databaseAudit.sql` to match latest database schema.

5.

   Server - updated comments in `/home/fms/schema/postgresq/databaseCreation.sql`.

6.

   Server - updated comments in `/home/fms/schema/postgresq/databaseConfig.sql`.

7.

   Server - updated comments in `/home/fms/schema/postgresq/fms.sql`.

8.

   Console - improved error response messages.

9.

   Server - implemented listening on all IP address when the configuration `interfaceConfig.listener.Properties.HOST` property is set to `'*'`.

10.

    Server - Configurable `ConnectionConfiguration.partnerIdentifier.mailBox` implemented as described in the Section called *Partner Identifier Configuration* in Chapter 3.

    Used for alternate messsage delivered directory based on the `partnerIdentifier` definition in the server configuration.

11. Server - Configurable `ConnectionConfiguration.interfaceConfig.deliveredContentDir` with similar functionality as the `ConnectionConfiguration.partnerIdentifier.mailBox` configuration setting.

    Used for alternate messsage delivered directory based on the `interfaceConfig` definition in the server configuration.

12.

    Server - Support for TLSv1.3 `ConnectionConfiguration.interfaceConfig.listener.Properties.ENABLED_` if supported by the `java` runtime.

13. FMC Management Console - support for exporting certificates in PEM and DER (binary) format.
14. FMC Management Console - improved display of certificate details.

Version: 5.4.2 Release 5

1. Server - added command line option `'-D53'` to revert to the FMS version 5.3 `deliveredContentDir` format as follows

   ```
   fms_installation_dir/<cc:deliveredContentDir>/toPartner.URLtoPath()/conversationID/messageID
   ```

   instead of the FMS version 5.4 `deliveredContentDir` format as follows

   ```
   fms_installation_dir/<cc:deliveredContentDir>/fromPartner.URLtoPath()/toPartner.URLtoPath()/messageID
   ```

Version: 5.4.2 Release 4

1. Server - fix missing `sun/security/validator/KeyStores.class` message on FMS server startup after upgrading Java® 8 release 212 or earlier to a later version. This resulting in FMS server not starting up.

   Also see 5 for details on circumventing this problem with previous versions of the FMS server.

2. Server - fix incorrect log message `Unable to load interface 'HTTP - AS4': NullPointerException while reading FMS Licence`. This was caused by incorrect or unresolvable listener HOST property in the Administration interfaceConfig section.

3. Server - Various logging improvements - display stack trace only when FMS LOGGING_LEVEL is set to TRACE as set in the server main.conf settings.

Version: 5.4.2 Release 3

1. Server - Include fmsas4lc in the debian and ubuntu distribution.
2. Server - New log message '`Using default p-mode`' in `debug` log level, if no p-mode match was found and `DEFAULT_PMODE` is set in the package configuration properties.
3. Server - Change log level from debug to info for '`Setting session pmode`' log message
4. Server - Fixed configuration `schema` path. No longer needs to be terminated with a path separator.

Version: 5.4.2 Release 2

1. Server - Fixed `metadata.fmd` <fmd:Location> from relative to absolute path. Was broken in 5.4.2 Release 1.
2. Server - Fixed configuration delivered-content path. Was broken in 5.4.2 Release 1.
3. Server - enhanced file move capabilities both within and across file systems including versioning (backups) for duplicates.
4. Server - all logging messages for received messages saved now show normalised and absolute path.
5. Server - fixed an issue where a SOAP fault indicating an internal server error was returned with an HTTP 500 error on receiving an empty payload. This caused the following error in the server logs

   `com.sun.xml.messaging.saaj.SOAPExceptionImpl: java.lang.NegativeArraySizeException`

   and `SEVERE com.sun.xml.messaging.saaj.soap.AttachmentPartImpl getRawContentBytes() - SAAJ0577: Exception while trying to get the Raw content for this attachment` sent to stderr.
6. Console - provided management console support for local `PKCS12` keystores.
7. server - fixed support for `PKCS12` keystores.
8. server - renamed fmsdaemon.deb to fmsdaemon to avoid confusion with debian and ubuntu install packages.
9. server - removed inadvertent log message displaying certificate alias names introduced in version 5.4.2 release 1.
10. server - database table for storing any payload URL associated with a message implemented.
11. server - database table for storing partner_agreements associated with a message implemented.
12. server - updated bouncy castle libraries.
13. server - fixed permission changes on directory `/home/fms` during installation.
14. server - fixed unnecessary call to `savelog` in server startup.
15. server - fixed `ConversionPattern` for `dest3` in `log4j.profile.as4`.
16. server - clear previous session logging details such as `ID`, `FROM` and `TO`.
17. fmsconf - `__FMS_Admin_DeleteCert` command extended with optional `result` argument.
18. fmsconf - `__FMS_Admin_GetCert` command extended to return the full certificate detail if the optional `full` argument is provided.

Version: 5.4.2 Release 1

1. Server - Optional partner mailBox (inbox/outbox) for local/remote partners or the other way around for inbound/outbound messages implemented in `schema/FMS/ConnectionConfiguration-1_0.xsd`. Used for alternate delivered directory based on partnerIdentifier.
2. Server - added command line option '`-w`' to not write a new configuration in case of updates.
3. Server - Message payloads are now correctly moved from temp to destination across file systems.
4. client - New '`-z`' command line option for compressing payloads. If included on the command line then payloads will be compressed overriding the p-mode setting.
5. client - The "`mimeType`" (non compressed payloads), "`schema`" and "`characterset`" properties may now be set in the pmode `businessInfo.PayloadProfile.mimeType`, `businessInfo.PayloadProfile.schemaFile` and/or the `businessInfo.PayloadProfile.CharacterSet` settings where these will be used

if not included in the properties passed in the `'-a'` command line argument. The appropriate `businessInfo.PayloadProfile` will be matched on the `"content-id"` property that SHOULD be included in any properties passed in the `'-a'` command line argument. Eg. the following illustrates a typical pmode businessInfo.PayloadProfile configuration

```
<pmode:PayloadProfile>
  <pmode:ContentID>sbdh-order</pmode:ContentID>
  <pmode:mimeType>application/xml</pmode:mimeType>
  <pmode:schemaFile>testSbdh.xsd</pmode:schemaFile>
  <pmode:CharacterSet>utf-8</pmode:CharacterSet>
  <pmode:maxSize>0</pmode:maxSize>
  <pmode:usage>expected</pmode:usage>
</pmode:PayloadProfile>
```

6. Server - table `insert` SQL statements now include the attributes being inserted. This to permit table attribute extension without affecting internal code.
7. Server - `ConnectionConfiguration.partnerIdentifier.errorEndpointURL` endpoint is now supported as follows

   This endpoint indicates the address to which to send ebMS errors generated on the receiving MSH that receives a message that caused an error. This is typically the endpoint address of the MSH sending the message that caused an error on the receiving side. It will only be used if the `asResponse` is set to `true` and if the FMS connection interface database is configured. If `ConnectionConfiguration.partnerIdentifier.errorEndpointURL` is not set the `P-Mode.errorHandling.receiverErrorsTo` setting will be used. This element was ignored prior to FMS version 5.4.2.
8. Server - P-Mode Error handling endpoints are now supported as follows
   a. `P-Mode.errorHandling.asResponse`

      This boolean parameter indicates whether (if `true`) errors generated from receiving a message in error are sent over the back-channel of the underlying protocol associated with the message in error, or as a callback message in the case that the FMS connection interface is configured with a database connection. All error signals will be sent back on the backchannel if no database connection is configured. SOAP Faults are always sent back to the remote partner on the back channel. This element was ignored prior to FMS 5.4.2.

   b. `P-Mode.errorHandling.receiverErrorsTo`

      This endpoint indicates the address to which to send ebMS errors generated by the MSH that receives a message in error. This is typically the endpoint address of the MSH sending the message that caused an error. It will only be used if `asResponse` is set to `true` and if the FMS connection interface database is configured. This endpoint is overriden by the optional connection configuration `partnerId errorEndpointURL` if configured. This element was ignored prior to FMS version 5.4.2.

9. Server - Receipt acknowledgement triggers have been extended with the following available arguments
   a. `SERVICE`.
   b. `ACTION`.
   c. `SENDER_IDENTIFIER`.
   d. `SENDER_ROLE`.
   e. `RECIPIENT_IDENTIFIER`.
   f. `RECIPIENT_ROLE`.
   g. `SUBMITTED`.
   h. `CONVERSATION_ID`.
   i. `PATH`.

10. Server - Improved message file saving permitting across filesystem saving of payloads, receipts and other signals between `MESSAGE_TMP_DIR` and `delivered-content` directories.

11. Server - `interfaceConfig.deliveredContent` directory structure changed from

    `../delivered-content/initiatingParty/message-id`

    to

    `../delivered-content/initiatingParty/respondingParty/message-id`

    where the partner strings have any preceding URI elements removed to ensure a consistent directory structure. Eg.

    `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultFrom`

    is transformed to just

    `defaultFrom`
12. Server - connection configuration `deliveredContent` directory support for absolute paths for both *nix and Windows operating systems.
13. Server - Use the responding (local) partner keystore for determining the private decryption key. This permits the use of multiple private keys each associated with a separate initiating (remote) partner.
14. Server - Use the initiating (remote) partner keystore for determining the key to sign a receipt signal response for a received user message. This permits the use of multiple private keys where remote partners can then be issued with separate public keys as associated with their respective keystores. Note that keystores can be shared across partners permitting a single keystore with a single private/public key across any number of remote and local partners.
15. Server - The `listener.aliasRef.alias` value is now used to determine the SSL private key alias in the associated listener keystore if set. If not set (empty) then the SSL private key will default to the first private key located in the keystore.
16. Server - Listener activation now picks up the correct interface properties from the server configuration including the following.
    a. `<prop:entry prop:key="SSL_PROVIDER">SunJSSE</prop:entry>`.
    b. `<prop:entry prop:key="SSL_NEED_CLIENT_AUTH">true</prop:entry>`.
    c. `<prop:entry prop:key="SSL_VERSION">TLSv1</prop:entry>`.
17. Server - Listening sockets are now closed down when shutting the server down and when closing and reloading interfaces from the management console.
18. Server - Logging now includes the thread id.
19. Server - Inbound messages using security setting `keyReferenceType="IssuerSerialNumber"` comparison on certificate issuer improved and is now based on RFC 3280 compliant comparison.
20. Client - improved usage message for the '`-h`' host command line option when using a TCP IPv6 address.
21. Server - A Receipt for a user message that could not be signed due to the private certificate alias being incorrectly configured on the server and as associated with the remote (previously local) partner configuration resulted in an `EBMS:0102` error returned to the initiating party. The server now logs the error and responds with a SOAP fault indicating a server side problem to the client.
22. FMC - improved support for TCP IPv6 addresses when connecting to a server.
23. fmsconf - general improvements and no longer requires certificates if the server admin listener `SSL_NEED_CLIENT_AUTH` property is set to false.
24. Server - Implement multiple private certificates for signing outbound messages by locating the private signing certificate using the initiating (remote) partner aliasRef alias in the initiating partner keystore for signing receipts.

    If the certificate alias could not be located in the initiating partner keystore search the responding (local) partner keystore using the responding partner aliasRef alias.

    Previously only the responding partner keystore was searched permitting only a single private signing certificate for outbound receipts.
25. Server - Use the responding (local) partner keystore for locating the private certificate for decrypting inbound messages and not the listener keystore. If not located then try the listener keystore as previously done.

26. Server - Verify signature for inbound user message by searching the responding (local) partner keystore for a match. If not found search the listener keystore. If not found fail.
27. Server - Improved default configuration and p-mode creation on startup. Now based on default properties
28. Windows install - include `send.bat` and sample XML payload
29. Server - Fixed receiving message with unknown encryption key and unknown partner results NPE.
30. Server - Fixed error if `pmode:queueMessages` is missing from the configuration.
31. Server - Logging for Linux changed as follows
    a. `/var/log/fms/debug.log` renamed to `/var/log/fms/fms.log`. General FMS server logs. Configured in `/etc/fms/log4j.properties.as4`.
    b. `/var/log/fms/out.log` renamed to `/var/log/fms/fms-stdout.log`. Configured in `/etc/fms/fmsdaemon` (ubuntu, debian) or `/etc/fms/fms.conf` (RedHat, CentOS).
    c. `/var/log/fms/err.log` renamed to `/var/log/fms/fms-webservices.log`. Detailed webservices soap and security logs. Configured as `java.util.logging.FileHandler` in `/etc/fms/logging.properties`
    d. `/var/log/fms/err.log` renamed to `/var/log/fms/fms-stderr.log`. General webservices soap and security logs. Configured in `/etc/fms/logging.properties` (`java.util.logging.ConsoleHandler` aka stderr) and `/etc/fms/fmsdaemon` (ubuntu, debian) or `/etc/fms/fms.conf` (RedHat, CentOS).
    e. `/var/log/fms/error.log`. Removed from `log4j.properties.as4`. Can be re-configured as `dest3` in `/etc/fms/log4j.properties.as4`
32. Server logging configurations for Linux are now as follows
    a. `/etc/fms/log4j.properties.as4`. FMS Server general logging configuration.
    b. `/etc/fms/logging.properties`. FMS Server Webservices logging configuration.
33. FMC Logging
    a. `/etc/fms/log4j.properties.mc`. FMC logging configuration (goes to stdout).
34. FMC - Fixed error not displaying `cc:interfaceConfig`. Due to invalid setting for `cc:interfaceConfig.Optimisations.retryFrequency = 300000`
35. Fixed `java.lang.ClassCastException: com.sun.xml.messaging.saaj.soap.impl.SOAPTextImpl cannot be cast to javax.xml.soap.SOAPElement` due to tabs or spaces in security encryption `<xenc:CipherReference...>` element of an encrypted incoming user message.

Version: 5.4.1 Release 3

1. client - improved logging
2. client - move atTime to before preparing message to be sent
3. client - optional `'-messageFile soapFile.out'` command line option to save outbound soap envelopes
4. client - fmsclient.as4.jar `'-d'` command line switch sets 'ALL' for logging.properties but no longer 'OFF' if not set. This so logging can be configured in logging.properties using `'java -Djava.util.logging.config.file=/home/fms/trigger/logging.properties'`.
5. server - messages are now saved with thread_id in the case of no DB connections as follows as per directory setting in main.conf

   ```
   <entry key="HTTP_REQUEST_STORAGE_DIRECTORY">/home/fms/http_requests/ </entry>
                 OUTGOING-01112017-074737.582-13_sig.dat
                 INCOMING-01112017-074737.253-13_um.dat
   ```

   where `'-13_'` indicates the thread id. This to ensure uniqueness in filenames
6. server - fixed `FMS_ARGS` in `/etc/fms/fms.conf`

7. client - the following attachment partproperties set on the `'-a'` command line option are removed from the part properties for compressed payloads.

    a. partProperties.remove("mimetype");
    b. partProperties.remove("description");

   This as the following duplicate properties are added when the payload is compressed

   `MimeType, Description, CompressionType`

   resulting in duplicate part properties.
8. server - Default properties updated to use latest signing and encryption algorithms.
9. server - Default configuration generation updated to use default properties.
10. server - Improved logging

Version: 5.4.1 Release 2

1. client - atTime `'-at msecs'` now works when timeouts are not set.
2. server - improve debug logging, and throw certificate error reason to be used in response if certificate validation failed due to a problem with the root certificate.
3. server - remove call to undocumented jdk.nashorn.internal.ir.debug.ObjectSizeCalculator as that causes errors (com.flame.shared.exceptions.PackingException: Unhandled exception Exception: Internal Server Error) with later versions of java (1.8.0_131...)

Version: 5.4.1 Release 1

1. client - new '-batch batchArgs.txt' command line option

   The new optional command line option `'-batch file'` permits multiple messages to multiple locations during a single session. Each line contains separate client arguments for separate messages.

   Invoke `fmsclient.as4.jar` as follows when using the new `'-batch'` command line option

   ```
   java -jar fmsclient.as4.jar -batch batchArgs.txt -k ./fmstestcerts  -ksp changeit -d
   ```

   where `batchArgs.txt` contain lines with `fmsclient.as4.jar` arguments for each separate message. Lines starting with a `'#'` character are ignored.
2. Improved logging - includes detail when compressing and securing messages.

Version: 5.4.0 Release 4

1. server - further improvements to large payload (up to 1.8GB) handling. Includes updates to webservices-rt including saaj-impl-1.3.28, mimepull to 1.9.1 and org.jvnet.staxex to 1.7.8
2. server and client - removed NLs surrounding <wsse:Security> ... </wsse:Security> tags - this in an attempt improve .NET interoperability.
3. server and client - HTTP_BAD_REQUEST (400) Responses eg. SOAP faults no longer blocked.
4. client - Improved truststore private key retrieval logging

Version: 5.4.0 Release 3

1. server - further improvements to large payload (up to 1.8GB) handling. Includes updates to webservices-rt including saaj-impl-1.3.28, mimepull to 1.9.1 and org.jvnet.staxex to 1.7.8
2. Also includes changes to removing NLs from encryption CipherValue element. SignatureValue still contains NLs.
3. Externalised setting of -Dsaaj.mime.optimization=true -Dsaaj.use.mimepull=true -Dsaaj.lazy.mime.optimization=true. These java args must now be set on FMS startup

Version: 5.4.0 Release 2

1. server - large payload (up to 1.8GB) handling. Includes updates to webservices-rt including saaj-impl-1.3.28, mimepull to 1.9.1 and org.jvnet.staxex to 1.7.8
2. server - ubuntu and debian package fixes and improvements.
3. server - logging improvements.

Version: 5.4.0 Release 1

1. server - xmlsec 2.0.7 based webservices server
2. server - optimisation if certificate matches entry in keystore
3. server - fixed problem with saving inbound binary compressed attachments
4. server - fixed activation problem with java-1.8 on some VMs

Version 5.3.4 Release 2

1. client - Certificate chain validation for validating incoming signing certs implemented.
2. client - Support for separation of ssl trust store (default java cacerts) from key store by default.

   New optional `'-st sslTruststore'` and `'-stsp sslTruststorePass'` command line options to set the ssl truststore to an alternate truststore. Set `'-st'` to the same as the keystore location for pre 5.3.4-2 behavior.
3. client - Command line option `'-u'` now works as expected. Ie. SSL support is not loaded if set.
4. client - Certificate chain validation for validating incoming ssl certs implemented.
5. client - Return code 3 in case of SSL certificate error.
6. client - Implemented `'-noSystemExit'` command line option. If set then program return status will be sent to `stdout` instead of the invoking shell.
7. client - Copyright notice now goes to `stderr`.

Version 5.3.4 Release1

1. client - Dynamic p-mode support for setting initiating and responding partner p-mode settings. Includes setting `partner.party@type` and `partner.role` on the command line.
2. client - New command line args include `-toType`, `-toRole`, `-fromType`, `-fromRole`.

   Use these command line options when any of `'-from'` or `'-to'` do not match what is in the p-mode. Used to override p-mode settings particularly when dynamic p-modes are required without having to create new p-mode files.

Version 5.3.3 Release 17

1. client - migrate to java 1.8.
2. client - further encryption support for `gcm` encryption algorithms.
3. client - support for EncryptedKey.EncryptionMethod.DigestMethod
4. client - support for key encryption method `"http://www.w3.org/2009/xmlenc11#rsa-oaep"`
5. client - fixed part properties usage example by escaping the first colon.
6. client - improved error trapping on non-existing attachments.
7. client - added stack trace on null pointer exceptions (return code 6).

Version 5.3.3 Release 16

1. client - fixed spec in `URL` instantiation. Was set to path instead of the full spec as returned by `url.getFile()`
2. client - no need to override `URLConnection()` if `connectTimeout` and `responseTimeout` is not set.
3. client - allow `pmode:party/@pmode:type` and `pmode:party/@type`. Fixed regression `pmode:party/@type` since 5.3.3-13
4. client - allow `pmode:service/@pmode:type` and `pmode:service/@type`. Fixed regression `pmode:service/@type` since 5.3.3-13
5. client - implemented underlying encryption support for `gcm` algorithms. Only available with java 1.8.
6. client - improved logging.

Version 5.3.3 Release 15

1. client - log for message transmit time (based on com.flame.client.as4 logger) when `'-d'` command line option is used.
2. client - set return code to `TIMEOUT` (4) on read or connect timeout.
3. client - set return code to `CONNECTION_EXCEPTION` (2) on failing to establish a connection.

Version 5.3.3 Release 14

1. client - Improved error message when failing to connect to remote host. Eg. `'network unavailable'`
2. client - New command line options

   `-at <Send Time>` - Send message at current time + time in milliseconds or at specified future date in format `yyyy-MM-dd'T'HH:mm:ss.SSS` eg. `2100-01-01T00:00:00:000`

   `-t` <Timeout> - Connection timeout in milliseconds. Defaults to 30000.

   `-T` <Timeout> - Response timeout in milliseconds. Defaults to 30000.

   Set both `-t` and `-T` to `0` to disable timeouts.

Version 5.3.3 Release 13

1. client - New command line option `'-X'` for dumping security providers.
2. client - Unused command line option `'-pr'` removed.
3. client - Command line option `'-r refToMessageId'` now also works for user messages.
4. client - New repeatable command line option `'-mp'` for setting MessageProperties.
5. client - messageID `'-m messageID'` now functions as expected.
6. client - Use of `p-mode.id` and `agreementref` now optional if not defined in the p-mode.
7. client - Only default `pmode:party/@type` to `"string"` if party is not a URI - also see RFC 2396

Version 5.3.3 Release 12

1. client - Improvements to SSL context handling.
2. client - Internal improvements and bug fixes.

Version 5.3.3 Release 11

1. client - Added 400 response code as SOAPFault (even though it's against SOAP 1.2 over HTTP) - also see https://java.net/jira/browse/SAAJ-74 and https://issues.jboss.org/browse/WFLY-3966
2. client - Strip path from attached payload files as appears in `Content-Disposition`
3. client - Internal improvements and bug fixes.

Version 5.3.3 Release 10

1. client - fixed bug saving payload attachments

Version 5.3.3 Release 9

1. client - remove schema from part properties.

Version 5.3.3 Release 8

1. client - bug fix for `'-service'` and `'-serviceType'` command line options when sending messages with attached payloads

Version 5.3.3 Release 6 and 7

1. client - debug flag now sets security config `dumpMessage='true'` for inbound messages.
2. client - attachment filename can now be either a filename or external payload pointed to by a url which will be retrieved
3. client - remove filename,content-id,mimetype,description and encoding from part properties
4. client - do not gzip payload if the payload is already compressed as per gzip sig bytes ID1 and ID2 from RFC 1952. Requires `<pmode:useCompression>true</pmode:useCompression>` in the p-mode.

Version 5.3.3 Release 4 and 5

1. client - Internal improvements including bug fixes for partner type.

Version 5.3.3 Release 3

1. client - Internal improvements and bug fixes.

Version 5.3.3 Release 2

1. client - Do not with an empty messageID

Version 5.3.3 Release 1

1. client - First release of fmsclient.as4.jar based on sas4client.jar

Version: 5.3.2 Release 19

1. server - dynamic partner support - create initiating partner from the 'from' party field in message if it does not exist. Note no @type as yet.
2. server - fixed NRR receipts not being signed for incoming signed messages from unknown remote partners
3. server - fix retrieving private key for the local partner - was set to attempting to obtain private key using the remote partner alias.
4. server - fixed debug message output
5. server - Certificate chain validation for validating incoming SSL certificates implemented. Requires root CA certificate either in the truststore or in `cacerts`.
6. FIXES for CA certs

Version: 5.3.2 Release 18

1. fmsconf - updated usage
2. server - improved checking on service, action, agreementref and mpc if these are empty strings
3. server - Dynamic p-modes. Set p-mode to `packagerConfig.Properties.DEFAULT_PMODE` if no p-mode match. If no `DEFAULT_PMODE` property match then no match.
4. server - improved logging
5. server - now validates sign certificates against issuer CA as well.

Version: 5.3.2 Release 17

1. Server - webservice-rt.jar
   a. Support for Java® 8.
   b. Support for AES_GCM_BLOCK_ENCRYPTION_128, AES_GCM_BLOCK_ENCRYPTION_192, AES_GCM_BLOCK_ENCRYPTION_256 with Java® 8.
   c. Improved logging.
   d. Attachment "Content-Transfer-Encoding" changed from base64 to binary.
2. Server - RESPONSE_SENT trigger improvements. This trigger fires after writing final response for incoming user message. It may be used for implementing twoWay MEPs.
3. Server - improved logging for cert handling - now shows keystore used.
4. Server - partner identification now done prior to the security processing. This requires that <eb:Messaging> -> <eb:UserMessage> -> <eb:PartyInfo> not be encrypted as per 5.1.6 of the AS4 Profile and as recommended in 7.4 of the core spec.
5. Server - Locate the private decryption and signature verification key in the local partner keystore if not found in the connection keystore. The connection private key is used for SSL so the private decryption and signature verification key can now be separate.
6. Server - Return EBMS:0303 on failure to un-compress a compressed incoming attachment.
7. Server - mark message as failed in case of internal Exception (Session.run())
8. Server - improved logging (include class and method name in TRACE mode)
9. Server - improved checking for non-existant messageID
10. Server - SignatureKeyCallback.AliasPrivKeyCertRequest - improved logging and set keystore referred to by initiatingParty (was respondingParty) as required when separating keystores based on partners.
11. Server - Trigger position RESPONSE_SENT include path
12. Server - if a test message with service set to '`http://.../service`' and action set to'`http://.../test`' then process complete message and now also creates a receipt if specified in the p-mode, but do not writePayload. e-SENS requirement
13. Server - throw packingException on missing MimeType PayloadInfo PartProperty - Compressed payload requirement from section 3.1 of the AS4-Profile.

14. Server - metadata.fmd support for `PartInfo/PartProperties/Property/@name="CharacterSet"`. Added MimeCharacterSet to `schema/FMS/FMSMetadataDocument-1_0.xsd` - not written to metadata.fmd as yet
15. Server - throw PackingException with ValueNotRecognized (EBMS:0001) on invalid character set `(eb:PartInfo/eb:PartProperties/eb:Property/@name="CharacterSet")` for a compressed xml SWA payload. Also see Section 3.1 of the AS4 Profile.
16. Server - Admin protocal command for __FMS_Admin_ReloadLicence. Also reloads connections.
17. Server - Fixed logging and management console viewing and updating of license details
18. Schema - FMSMetadataDocument-1_0.xsd support for the following Eg. metatdata.fmd may now include

```
<fmd:MessagePayloads>
 <fmd:Payload>
  <fmd:MimeContentID>xmlpayload@minder</fmd:MimeContentID>
  <fmd:MimeContentType>application/xml</fmd:MimeContentType>
  <fmd:MimeCharacterSet>utf-16</fmd:MimeCharacterSet> <!-- optional -->
  <fmd:Location>/home/fms/delivered-content/flame-c2/703131/6622e78c@mindertestbed.org/attachment-0</fmd:Location>
 </fmd:Payload>
```

19. Schema - FMSMetadataDocument-1_0.xsd support for the following
    a. metadata.fmd support for eb:CollaborationInfo/eb:AgreementRef Eg. metatdata.fmd may now include

    ```
    <fmd:AgreementRef>test-agreement</fmd:AgreementRef>
    ```

    b. metadata.fmd support for

    `eb:MessageInfo/eb:Timestamp`
    Eg. metadata.fmd now includes

    ```
    <fmd:Timestamp>2016-02-12T12:30:22.552Z</fmd:MessageID>
    ```

20. fmsconf implemented.

Version: 5.3.2 Release 16

1. Server - Close SSL socket bugfix for 5.3.2-14 and 5.3.2-15
2. Fixed non-blocking server mode as reported for 5.3.2-15 (hanging sockets) - 5.3.2-16

Version: 5.3.2 Release 15

1. Server - Fixed NPE bugfix for 5.3.2-14 on returning a response when attempting to fire RESPONSE_SENT trigger. Occurs when pulling from non-existing MPC or pushing to an non-existing p-mode.

Version: 5.3.2 Release 14

1. message.message_type - OTHER=7 for soap faults
2. Server - much improved log messages - TRACE will generate huge logs including stack traces on error conditions, DEBUG is useful for debugging and recommended in a running test environment and INFO appropriate for a long running production server.
3. Server - SSL protocol property

    ```
    <prop:entry prop:key="ENABLED_SSL_PROTOCOLS">TLSv1.2</prop:entry>
    ```

4. Server - improved socket connection threading and handling - new `LISTENER_COUNT` property with default set to 5 plus the value of `LISTENER_COUNT`. Used for number of listeners.

    ```
    <prop:entry prop:key="LISTENER_COUNT">5</prop:entry>
    ```

5. Server delivery directory and file handling improvements

    User Messages
    `fms_installation_dir/<cc:deliveredContentDir>/toPartner.URLtoPath()/conversationID/messageID/fromPartner.URLtoPath()`

Meta file delivery to
```
fms_installation_dir/<cc:deliveredContentDir>/toPartner.URLtoPath()/conversationID/messageID/metadata.fmd
```

Receipts
```
fms_installation_dir/<cc:deliveredContentDir>/fromPartner.URLtoPath()/message_id/receipt.soap
```

Meta file delivery to
```
fms_installation_dir/<cc:deliveredContentDir>/fromPartner.URLtoPath()/message_id/metadata.fmd
```

6. Server message storage directory improvements (used if no db)

   user messages to fms_installation_dir/HTTP_REQUEST_STORAGE_DIRECTORY/[INCOMING|OUTGOING
   mmddyyyy-hhmmss.SSS_um.dat (no mime messages so no attachments) signal messages to fms_installation_dir/HTTP_REQUEST_STORAGE_DIRECTORY/[INCOMING|OUTGOING]-mmddyyyy-hhmmss.SSS_sig.dat

7. Server interfaceConfig acls implemented for AS4 listeners.

8. Server `RECEIVE` trigger argument 10 now contains PATH indicating full path name of delivered messages and metadata.

   Requires change to `ConnectionConfiguration.xml` triggers as follows

   Change
```
<tg:Trigger>
  <tg:identifier>ReceiveMessage</tg:identifier>
  <tg:enabled>true</tg:enabled>
  <tg:type>SynchronousExecutable</tg:type>
  <tg:location>RECEIVE</tg:location>
  <tg:instruction>./trigger/Receive.sh %1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s</tg:instruction>
  <tg:providedArguments>action</tg:providedArguments>
  <tg:providedArguments>conversationID</tg:providedArguments>
  <tg:providedArguments>event</tg:providedArguments>
  <tg:providedArguments>messageID</tg:providedArguments>
  <tg:providedArguments>MPC</tg:providedArguments>
  <tg:providedArguments>processingMode</tg:providedArguments>
  <tg:providedArguments>recipientIdentifier</tg:providedArguments>
  <tg:providedArguments>senderIdentifier</tg:providedArguments>
  <tg:providedArguments>service</tg:providedArguments>
  <tg:executionType>External</tg:executionType>
</tg:Trigger>
```

   to
```
<tg:Trigger>
  <tg:identifier>ReceiveMessage</tg:identifier>
  <tg:enabled>true</tg:enabled>
  <tg:type>SynchronousExecutable</tg:type>
  <tg:location>RECEIVE</tg:location>
  <tg:instruction>./trigger/Receive.sh %1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s %10$s</tg:instruction>
  <tg:providedArguments>action</tg:providedArguments>
  <tg:providedArguments>conversationID</tg:providedArguments>
  <tg:providedArguments>event</tg:providedArguments>
  <tg:providedArguments>messageID</tg:providedArguments>
  <tg:providedArguments>MPC</tg:providedArguments>
  <tg:providedArguments>processingMode</tg:providedArguments>
  <tg:providedArguments>recipientIdentifier</tg:providedArguments>
  <tg:providedArguments>senderIdentifier</tg:providedArguments>
  <tg:providedArguments>service</tg:providedArguments>
  <tg:providedArguments>path</tg:providedArguments>
  <tg:executionType>External</tg:executionType>
</tg:Trigger>
```

9. Server returns `HTTP CODE 400` regression fixed - this bug crept in with the -14 build updates. May cause the pulling partner to drop the associated EBMS response message

eg. `EBMS:0006:EmptyMessagePartitionChannel`.

10. Server now returns 200.

Version: 5.3.2 Release 13

1. server - provide support for absolute URIs as per RFC 2616 (http://tools.ietf.org/html/rfc2616#section-9.5)
2. server - improved usage message
3. rpm package - include Envelope directory

Version: 5.3.2 Release 12

1. server and as4 client - Java® 7 compatibility, tweak on socket close
2. server - Java® 1.6.0_31 problem with attachments - Changed saaj.lazy.contentlength from true to false

Version: 5.3.2 Release 11

1. Server - messageid suffix configured as $host gives wrong host - fixed
2. Server - Locate the private decryption and signature verification key in the local partner keystore if not found in the connection keystore.
3. SSL key customisation - `ConnectionConfiguration.xml`

   Change

   ```
       <cc:listener cc:packagerConfigID="AS4-PackageManager">
         <cc:name>AS4</cc:name>
         <cc:className>com.flame.connection.impl.ebXML.AS4.RemoteIn</cc:className>
         <cc:aliasRef cc:keystoreID="default">
           <cc:alias>fmsrns</cc:alias>
           <cc:password>fmsrns</cc:password>
         </cc:aliasRef>
   ```

   to

   ```
       <cc:listener cc:packagerConfigID="AS4-PackageManager">
         <cc:name>AS4</cc:name>
         <cc:className>com.flame.connection.impl.ebXML.AS4.RemoteIn</cc:className>
         <cc:aliasRef cc:keystoreID="flamessl">
           <cc:alias>flamessl</cc:alias>
           <cc:password>fmsrns</cc:password>
         </cc:aliasRef>
   ```

   and add the following

   ```
     <cc:keystoreRef cc:ID="flamessl">
       <cc:name>ssl_certs</cc:name>
       <cc:type>JKS</cc:type>
       <cc:pass>changeit</cc:pass>
     </cc:keystoreRef>
   ```

4. Server - Implement signing messages and NRR (non repudiation receipts) with separate signing cert instead of the partner SSL cert

   Gen the new private key into a separate keystore as follows - required to avoid java.security.InvalidKeyException: Not an RSA key: DSA when do the default keytool -genkey ???

   ```
   keytool -genkeypair -keyalg RSA -sigalg MD5withRSA -alias flamesign -keypass fmsrns \
     -keystore sign_certs -storepass changeit -keysize 2048 -storetype JKS
   ```

   Then set the partner and keystore as follows

   ```
   <cc:partnerIdentifier cc:value="flame" cc:type="string">
     <cc:endpointURL>https://localhost:6444/AS4</cc:endpointURL>
     <cc:partnerType>LOCAL_PARTNER</cc:partnerType>
     <cc:username>flame</cc:username>
     <cc:password>flame</cc:password>
     <cc:aliasRef cc:keystoreID="flamesign">
       <cc:alias>flamesign</cc:alias>
       <cc:password>fmsrns</cc:password>
   ```

```
</cc:aliasRef>

<cc:keystoreRef cc:ID="flamesign">
  <cc:name>sign_certs</cc:name>
  <cc:type>JKS</cc:type>
  <cc:pass>changeit</cc:pass>
</cc:keystoreRef>
```

5. Server and client swa 11 encryption compliant transform fix (metro webservices jars), and wsse:mustUndertand = "true" instead of "1"
6. Server - the messageid suffix not included for errors - fixed 5.3.2-11
7. Client - Softened up service and action - can now be set via the command line
8. Client - fix schema option for attachments, set via the `'-a schema:location;...'` command line option per attachment
9. Client - '-suffix' command line option to set the messageid suffix
10. Client - new exception that can be used for invalid arguments
11. Client - Allow escape characters to escape required characters such as ':' in the `'-a'` attachment key value pairs

Version: 5.3.2 Release 9

1. Server - NPE on storing invalid messagees
2. Update FMS server p-mode search algorithm
   i. Using the optional `@pmode` attribute in the `agreementref` - if set but does not match return `EBMS:0010`
   ii. combination of to/from/service/action - if to/from/service/action are set but no match goto next step
   iii. combination of to/from/agreementref - if to/from/agreementref set but no match goto next step
   iv. Just agreementref - if agreementref set but no match then return EBMS:0010.
   v. combination of from/MPC for pulls - if from/MPC set but no match then return EBMS:0010. and finally
   vi. just MPC for pulls
3. New configuration properties

```
<prop:entry \
prop:key="EBXML_SOAP_EXTENSION_NS">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/</prop:entry>
<prop:entry \
prop:key="DEFAULT_INITIATOR_ROLE">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/initiator</prop:entr
<prop:entry \
prop:key="DEFAULT_RESPONDER_ROLE">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/responder</prop:entr
<prop:entry \
prop:key="DEFAULT_SERVICE">http://docs.oasis-open.org/ebxml-msg/as4/200902/service</prop:entry>
<prop:entry \
prop:key="DEFAULT_ACTION">http://docs.oasis-open.org/ebxml-msg/as4/200902/action</prop:entry>
<prop:entry \
prop:key="ONEWAY_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay</prop:entry>
<prop:entry \
prop:key="TWOWAY_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay</prop:entry>
<prop:entry \
prop:key="PUSH_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push</prop:entry>
<prop:entry \
prop:key="PULL_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pull</prop:entry>
<prop:entry \
prop:key="PUSH_PUSH_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPush</prop:entry>
<prop:entry \
prop:key="PUSH_PULL_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPull</prop:entry>
<prop:entry \
prop:key="PULL_PUSH_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pullAndPush</prop:entry>
<prop:entry \
prop:key="SYNC_MEP">http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/sync</prop:entry>
```

Version: 5.3.2 Release 9

1. Client - create attachment directory if it does not exist. Was broken for default case.
2. Client - fixed ignoring of suffix - caused light client to be non RFC 2822 compliant for MessageId
3. Client - attachment part properties are not included in PartInfo from the p-mode or from the `'-a'` command line option - done
4. Client - Allow pmode:ID to be set to an empty string eg. <pmode:ProcessingMode pmode:ID="" xmlns:tg="http://fms.flame.business/FMS/schema/Trigger" xmlns:pmode="http://fms.flame.business/FMS/schema/ProcessingMode"> if empty do not include @pmode in CollaborationInfo.AgreementRef

Version: 5.3.2 Release 7

1. Server - table message add column event - provides missing event from old configuration table
2. Server -table dispatch add column retry_threshold - provides missing retry_threshold from old configuration table
3. Server -table dispatch add column retry_interval - provides missing retry_interval from old configuration table
4. Server -index dispatch_attempt_dispatch_id_idx - missing index - used for DatabaseHandler.isMessageRetriable
5. Server Configuration - Security Encryption Certificate - make it minOccurs="0" - done 5.3.2-6
6. Server - add event to message table (was originally in configuration which now no longer exists). - done 5.3.2-6
7. Server - remove journal trace logs - done
8. Server - customise the prefix/suffix for messageid - done
9. server - toRole incorrect - set to fromRole - fixed problem xpath on the as4client
10. Server - SID, FROM and TO log4j settings are not reset on new connections made to the server - these must be cleared. - fixed 5.3.2-7
11. Server - MessageId for empty MPC messages are wrong - contain the to party instead of the from party. Also if the party is a url then characters to the right of the @ do not conform to RFC 2822 - need to only get the last bit or use host - fixed 5.3.2-7
12. Server - Receipt message ids have $to instead of $from - fixed

Version: 4.3.0 Release 1

1. Server - Added message direction to PullMessagesForMPC query to prevent incorrect messages being sent to pull requester (Incorrect private key for message error)
2. Server - All interfaces now have access to all p-modes.
3. Server - Log error when service is not found.
4. Server - Removed configuration limit on listeners.
5. Server - Fixed bug during dispatch update when transaction is locked.
6. Server - Fixed configuration refresh not refreshing listener configuration.
7. Server - Fixed endpoint/MPC in partner identifier references.
8. Server - Fixed incorrect connection path being printed on bind.
9. Server - Fixed message transaction locking.
10. Server - Fixed null P-Mode message referring to old interface association.
11. Server - Fixed possible null pointer when no packagers exist.
12. Server - Fixed server classpath after BC library update.

13. Server - Fixed synchronous reliable messaging with and without receipts.
14. Server - Implemented auto pull features in processing modes.
15. Server - Implemented service and action property searching for Rosettanet.
16. Server - Implemented service mapping location using service identifier rather than associated service mapping
17. Server - First write messages to temp directory before moving to `delivered-content` directory.
18. Server - Fixed potential for duplicated http response send.
19. Server - Adjusted service names to UBL and PIDX.
20. Console - Added FMC drag and drop transfer handlers.
21. Console - Confirm whether the administrator wants to refresh the connections on save.
22. Console - Added splashscreen to jar load.
23. Console - Improved administrator connection error message.
24. Console - Fixed handling of modified nodes on exit.
25. Console - Moved location of the help dialog to be on the left if there is no space on the right.
26. Console - Set modality type for the wizard dialog.
27. Console - Paste now appends `Copy Of` in front of pasted node.
28. Console - Adjusted paste to insert nodes at an index.
29. Console - Adjusted properties so it uses old properties when resaving.
30. Console - Implement shortcuts for management console
31. Console - Adjusted transport and payload security levels to be MIME and SOAP security levels respectively.
32. Console - Adjusted wizard to run without a connection to a server.
33. Console - Fixed P-Mode duplication copy-of-copy-of issue.
34. Console - Fixed clone bugs.
35. Console - Fixed cloning of P-Mode conversationID.
36. Console - Fixed help file referencing.
37. Console - Fixed tree not modifying the correct nodes.
38. Console - Fixed null variables causing XML parse failure when saving.
39. Console - Fixed null pointer for null outgoing connection.
40. Console - Fixed partnerIdentifier modification causing null pointer when handler node is not selected.
41. Console - Fixed template re-assignment issue when going back to beginning without changing template types.
42. Console - Fixed unmodified tree method, forward scan on handler node first, then backward to root.
43. Console - Implemented undo editor, resets all edits with each new panel that is opened.
44. Console - Implemented administrator MOTD command.
45. Console - Implemented Tree Drag and Drop.
46. Console - Implemented multiple administrator connections.
47. Console - Implemented report graphs (Messages/Minute over Time, Messages/Minute meter, Message Times, Message Composition)
48. Console - Implemented report message search queries.
49. Console - Implemented empty file handlers for easy access (cached).
50. Console - Implemented imports of configurations to an administration connection.
51. Console - Implemented reports node for message queries with menu accessors.
52. Console - Implemented unified wizard dialog.
53. Console - Moved `Quit` from `File` menu and `About` from `Help` menu for Mac OS X

54. Console - Moved content identifier in `PayloadProfile` to below schema selection.

Version: 4.2.7 Release 1

1. System - Upgraded compiler to Java 1.6
2. Server - Implement extended protocol for the administration connection to the server. This to enable multiple concurrent connections from the FMC to FMS servers.
3. Server - Removed all connection restrictions. Licensing restrictions are implemented at the partner level.
4. Console - Deprecated Configuration Editor and replaced it with FMC.
5. Console - Implemented authentication for the administration connection.

- Version: 4.2.6 Release 18
  1. Server - Implemented FMSMetadataDocument to be deposited in the delivered content directory.
  2. Server - Added application client correlation code for correlation of sequenceID and sequenceNum from client properties.
  3. Server - Implemented WS-ReliableMessaging
  4. Server - Implemented configurable Trigger system which can invoke a system executable or log a custom message when trigger location is encountered.
  5. Server - Implemented custom HTTP compression as well as client compression for slower connections.
  6. Server - ebXML - Implemented UsernameToken authentication.
  7. Server - Implemented database pooling and queuing system.
  8. Server - Implemented system network optimisations and prioritizing of TCP/IP packets.
  9. Server - Adapted protocolListener to allow handles with sub handles in the format https://host:port/handle/subhandle
  10. Server - Added log message to log time/size/bandwidth after upload/download.
  11. Server - Adjusted storage of messages to indicate the direction (Incoming or Outgoing).
  12. Server - Added thread names for logging purposes.
  13. Configuration - Fixed memory leaks after opening popup panels.
  14. Configuration - Tightened up GUI controls and consistancy.
  15. Configuration - Added configurable tooltips with HTML ToolTip viewer to FMC.
  16. Many other bug fixes and enchancements.

- Version: 4.1.3 Release: 1
  1. Server - Implemented Processing Modes (P-Mode).

- Version: 4.1.2 Release: 10
  1. Server - Include message content in SOAP Body and implement namespace separation for WSS encryption and signatures.
  2. Server - Library update to fix external namespace bug in SOAP Envelope when marshalling complex XML objects.

- Version: 4.1.2 Release: 9
  1. Server - Adjusted Database connection handling to abort connection instantiation if database connection fails unless in INIT mode so as to generate the connection specific settings.
  2. Server - Adjusted destinationAddress usage.
  3. Server - Updated transmission message for pull requests.
  4. Server - Added MPC attribute value to UserMessage.

5. Server - Adjusted exception handling for unknown MPC
6. Server - Allowed all interfaces to be instantiated during INIT mode to allow for non enabled connection specific configuration entries to be populated during first init to prevent confusion during initial configuration.
7. Server - Rethrow SQL exception during connection should failure occur for underlying layer to handle.
8. Server - ebXML Package Manager - Adjusted so that a client specified CPA in a Test environment will not be cached.
9. Server - ebXML Package Manager - Removed Production environment check when checking for Override Client CPA.
10. Server - ebXML Package Manager - Adjusted CPA TTL to be in seconds instead of milliseconds.
11. Server - ebXML Package Manager - Removed Production environment check when checking for Override Client CPA.
12. Server - ebXML Package Manager - Added null checks when recipient is unknown but error message must be generated.
13. Client - Implemented new exit status code handling.
14. Client - Added -q for a single messageID query instead of using -Q for testing multiple messageIDs from the messageID temp store file.
15. Client - Added EMPTY_MESSAGE_PARTITION_CHANNEL exit code for PullRequest errors.
16. Client - Removed RosettaNet specific PIP references.
17. Configuration - Adjusted alignment of buttons in button panel, adjusted alignment of administration menu bar, added administration version to menu About.
18. Configuration - Changed CPA config `Override Client CPA in Production` to `Override Client CPA`, removed defunct CPA registry URL field.
19. Configuration - Set usage to only display Production and Test, admin connections are fixed as Usage: Admin.
20. Configuration - Adjusted admin console colours for better readability, recalled last viewed configuration on reload.
21. Administration - Added server log level notification on new admin connection to adjust gui controls.
22. Administration - Fixed menu options availability when disconnected.

- Version: 4.1.2 Release: 4
  1. Server - Added readMimeMultiPart from ebMS connections to PackageManager, unsynchronized some methods to prevent deadlock.
  2. Server - Added better logging for pullrequests.
  3. Server - Adjusted readHTTPBoundary finishing check to only check if the line ends with boundary + "--" rather than more explicit equals.

- Version: 4.1.2 Release: 3
  1. All - Implemented Pull Requests.
  2. Server - Added findPartnerIdentifierByEndpoint method for use with PullRequests.
  3. Client - Fixed required argument checking arrays.

- Version: 4.1.2 Release: 2
  1. Server - Adjusted ebXML.RemoteOut to use custom Socket instead of HttpsURLConnection due to synchronous transmission problems.
  2. Server - Shifted message construction around, took Receipt, Error and PullRequest construction out of createV3Envelope and placed construction code into createV3Receipt, createV3Error and createV3PullRequest respectively, renamed createV3Envelope to createV3Request.

3. All - Added PartnerType enum for PartnerIdentifiers one of {LOCAL_PARTNER, REMOTE_PARTNER, PULL_PARTNER}.

- Version: 4.1.1 Release: 3
  1. All - Renamed Packing to Package and PIP to Schema.
  2. All - Renamed PipMapping and PIPEntry to SchemaMapping and SchemaEntry.
  3. Administration - Added expiry date to Licence check.
  4. Server - Rebuilt with new Metro Webservices libraries.

- Version: 4.1.1 Release: 1
  1. Configuration - Centered Connection Config arrow between buttons, Added delayed draw of watermark when resizing or moving to prevent 100% CPU usage due to downscaling of watermark image during continuous resize.
  2. Server - Implemented chained EntityResolver as well as deeper search mechanisms for the location of XML Schemas.

Appendix I. Glossary

- CPA

  Defined as a Collaboration Protocol Agreement and derived from a CPP. It is an agreement between business partners which defines the way partners interact in performing a chosen set of business collaborations. It is defined in XML. Further details available at http://www.oasis-open.org/committees/ebxml-cppa/documents/ebcpp-2.0.pdf

- CPP

  The Collaboration Protocol Profile (CPP) defines the message exchange capabilities of a business partner and the business collaborations that it supports. Further details available at http://www.oasis-open.org/committees/ebxml-cppa/documents/ebcpp-2.0.pdf

- CP

  Conformance Profile - the gateway profile that lists the features expected of a MSH acting as an e-Business gateway to backend systems.

- DUNS

  Data Universal Numbering System (DUNS) is a system developed and regulated by Dun & Bradstreet which assigns a unique numeric identifier to a single business entity. Further details available at http://www.dnb.com

- FMS

  Flame Message Server.

- MEP

  Message Exchange Pattern (MEP) is defined as a message sequence that follows a defined message exchange pattern required by a communications protocol. There are two major message exchange patterns being one-way and two-way. MEPs are manifested in the P-Mode.

- MPC

  A Message Partition Channel (MPC) is a flow of messages from a set of sending MSHs to a set of receiving MSHs. An MSH may have an associated priority determined by agreement between business partners.

- MSH

  Message Service Handler (MSH) provides the interface and services required for encapsulating messages in in an electronic envelope and securely forwarding the envelope to a remote destination possibly via an MSH chain. FMS is an instance of an MSH.

- MOM

  Message Oriented Middleware (MOM) is a client/server infrastructure that enables the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms (Wikipedia).FMS falls in this class of software.

- *Package Manager*

  A mechanism used to encapsulate messages according to the various requirements of a published messaging specification, such as ebXML or RosettaNet. Implemented as a Java class file stored in a Jar archive.

- P-Mode

  An MSH operates either for sending or receiving messages with knowledge of some contextual information that controls the way messages are processed. This contextual information that governs the processing of a particular message is called Processing Mode (P-Mode). Because different messages may be subject to different types of processing, an MSH generally supports several P-Modes. The set of all P-Modes that are supported by an MSH during operation, is called the P-Mode operation set of the MSH.

  The association of a P-Mode with a message may be based on various criteria, usually dependent on header data (e.g. Service/Action, Conversation ID, or other message properties). Which security and/or which reliability protocol and parameters, as well as which MEP is being used when sending a message, is determined by the P-Mode associated with this message.

- *Partner Identifier*

  A unique identifier associated with a Business Partner.

- *Reliable Messaging*

(a) WS-Reliability 1.1 and (b) WS-ReliableMessaging. (a) has been an OASIS standard for several years, has been tested and implemented by communities of users, notably in Asia. (b) is a more recent standard, still awaiting for WS-I interoperability guidance, but enjoying a broad support among US-based companies. This fulfills the Quality of Service requirements for a message.

- SOAP

  SOAP (from wikipedia) is a protocol for exchanging XML based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built.

- WSS1.0/1.1

  Web Services Security Encryption/decryption of any SOAP message content and generation/verification of any digital signatures forms part of this specification.

Appendix J. Bibliography

- AS4-profile-v1.0.pdf http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/os/AS4-profile-v1.0-os.html

  OASIS AS4 Profile of ebMS 3.0 Version 1.0 Committee Specification 03 dated 30 April 2012.

- ebCPP-2_0.pdf

  OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee

- ebms_core-3.0-spec.pdf http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.html

  OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features, 1 October 2007.

- rosettanet.org

  RosettaNet Implementation Framework: Core Specification Version: V02.00.01 Revised: 6 March 2002

- wikipedia.org

  Definitions of messaging terminology and acronyms.

Index

Notes

1.
2.
3.
4.
5.
6.
7.
8.
9.