OpenAS2 Server Application

Table of Contents

1. Introduction

The OpenAS2 application enables you to transmit and receive AS2 messages with EDI-X12, EDIFACT, XML, or binary payloads between trading partners. The AS2 implementation conforms with [RFC4130](#) supporting the 1.1 specification.

This document describes how to install, configure and use OpenAS2. An appendix provides information on upgrade procedures as new versions of the application are released.

In this document a partner can be either your own company or a company you will be exchanging data with using AS2.

The sample configurations in this document are based on Unix type OS but in general the only significant difference is that it may be necessary to use "\" instead of "/" for folder name separators on Windows based machines but because the application is Java it should work fine leaving the "/" for the most part as Java will do the conversion if necessary.

This document is valid for version 2.6.2 and up.

2. Glossary

EDI – Electronic Data Interchange

MDN - Message Disposition Notification

JCE - Java Cryptography Extension

3. Basic Functional Overview

The OpenAS2 application provides the following mechanisms for sending and receiving files with a vanilla deployment:

- Files to be sent are placed in folders that are detected by directory polling modules and sent to the relevant destination AS2 partner using the AS2 protocal. These polling modules are configured in XML and provide the ability to determine the target partner either based on the file being in a specific folder that is specifically designated for a target partner or a generic folder where the file name contains the relevant information to determine the target partner.
If configured for an MDN response, the MDN response from the partner is stored in a folder defined by the MDN storage module

- Received files from partners are placed a location defined the the message storage module. The default configuration for OpenAS2 creates dedicated inbox folders per partnership.
 If configured for an MDN response, the application will automatically send an MDN that is by default signed.

All sent and received files are tracked in a database tracking system that updates the progress of the message state in the database at key points in the life cycle of message.

4.   Installing OpenAS2

4.1. System Requirements

To be able to run the OpenAS2, you will need:

1. Java™ installed on the machine you intend to run the OpenAS2 server on – this document uses Java 1.6.

2. The OpenAS2 package version you wish to use. The downloadable packages can be found here: <https://sourceforge.net/projects/openas2/files>

3. Java Cryptography Extension (JCE) policy files - you can download the correct version from the Java website. Search "Java Cryptography Extension Unlimited Strength" to find the right cryptography extension for your version of Java. The current link for Java8 is here.

4.2. Installing Application

The following steps will provide an installed app on a target machine:

1. Unzip the downloaded OpenAS2 package into a suitable location, which we will refer to as `<install_dir>`.

    NOTE: Typical values for `<install_dir>` locations are `/opt/OpenAS2` under Linux®/Unix or `C:\OpenAS2` under Microsoft® Windows®.

2. For the encryption and certificate management to work correctly, you must have the proper JCE policy files installed in your version of Java (see system requirements above). The downloaded zip archive contains the two files `local_policy.jar` and `US_export_policy.jar`. Install them into your Java location under `<JAVA_HOME>/lib/security`. Back up the existing files before installing these new ones. There are numerous detailed articles on the web for installing these files if you need more information.

The file structure will look something like the figure below without the data and logs folders which are created automatically by the server when it starts based on configuration if they do not exist.

Name	Date Modified	Size	Kind
▼ 📁 bin	Today 13:52	--	Folder
commons-logging.properties	3 August 2015 23:38	66 bytes	Java p...ies
gen_p12_key_par.sh	27 July 2015 19:07	2 KB	shell script
start-openas2.bat	Today 00:06	3 KB	MacVi...ume
start-openas2.sh	2 August 2015 22:55	999 bytes	shell script
build.xml	4 August 2015 17:52	3 KB	XML text
▼ 📁 config	4 August 2015 22:53	--	Folder
as2_certs.p12	27 July 2015 19:17	5 KB	person...S#
commands.xml	16 August 2010 09:58	1 KB	XML text
config.xml	3 August 2015 23:21	4 KB	XML text
emailtemplate.txt	16 August 2010 09:58	166 bytes	text
partnerships.xml	3 August 2015 22:41	2 KB	XML text
▼ 📁 data	Yesterday 23:38	--	Folder
▶ 📁 OpenAS2A_OID-OpenAS2B_OID	Yesterday 23:38	--	Folder
▶ 📁 OpenAS2B_OID-OpenAS2A_OID	4 August 2015 22:56	--	Folder
▶ 📁 resend	4 August 2015 22:55	--	Folder
▶ 📁 temp	Yesterday 23:38	--	Folder
▶ 📁 toAny	4 August 2015 22:55	--	Folder
▶ 📁 toOpenAS2A	Yesterday 23:37	--	Folder
▶ 📁 toOpenAS2B	Yesterday 23:38	--	Folder
▶ 📁 lib	4 August 2015 22:53	--	Folder
▼ 📁 logs	Today 00:05	--	Folder
log-08042015.txt	4 August 2015 22:56	3 KB	text
log-08052015.txt	Yesterday 23:38	5 KB	text
log-08062015.txt	Today 14:03	216 bytes	text
manifest.mf	1 August 2015 08:46	68 bytes	Unix E...le F
▶ 📁 src	1 August 2015 22:34	--	Folder

4.3. Tuning Java

The default settings for the Java virtual machine in the startup script (start_openas2.sh or start_openas2.bat) will work for installations on most machines for low volume/small file size transfers. However, if your system will be transferring large files you will need to increase memory allocation. If you expect to support very high AS2 traffic you will need to increase memory allocation and possibly tune the garbage collector to get reasonable performance.

How much you can increase memory allocation to Java will depend on how much RAM is installed on the system running OpenAS2 and how many other processes will be running concurrently that will also require memory. Most systems deploy with at least 8GB RAM these days so increasing memory allocation from the default amount in the startup script should not cause adverse affects to the system.

To increase memory allocation you need to increase the heap space. This is set using the -Xmx option. You could increase this from the 384m (m = MB) default setting to 1g or 2g to get good performance for larger files or busy systems and for very large files given enough RAM you can set it to 6g or 8g. Search for "-Xmx" in the startup script and adjust accordingly.

For garbage collection you may want to allocate a more appropriate garbage collector than the default parallel collector that is the default in Java. In Java 7 and up, the G1 collector is ideal if you use large heap space allocation. To enable it add this to the command line parameter:

–XX:+UseG1GC

Based on basic internal testing and user feedback, the following are guidelines for setting your heap space (Xmx):

- files up to 50MB – 384m
- files up to 150MB – 756m
- files up to 300MB - 2g
- files up to 500MB – 3g
- files up to 750MB - 4g

5. Configuration Overview

This section explains the details of the configuration files and how they link together.

The OpenAS2 server uses four files to configure and execute:

1. **config.xml** – configures the application such as the types of modules that are started, the logging systems, command processors and global properties. This is the key file for starting OpenAS2 and must be passed in either as a command line option or as an environment variable. If the application cannot find the file it will fail to start.
   NOTE: This is the default name for the file and is referred to in this document by that name but you can use any name you like since it is passed in from the command line by the invoking batch script or can be set as an environment variable. For example in the nix shell script it is this line that determines the configuration file name and location:
   EXTRA_PARMS="$EXTRA_PARMS -Dopenas2.config.file=${binDir}/../config/config.xml"

2. **partnerships.xml** – configures the partners and partnerships. Provides the ability to specify different signing and encryption algorithms, message compression, MDN handling etc
   NOTE: This is the default name for the file and is referred to in this document by that name but you can use any name you like – see the Partner Configuration section below for how to set a different name.

3. **as2_certs.p12** – a PKCS12 keystore that stores the SSL certificates used to secure the messages for all partners. It contains the primary key for your own company as well as the public keys for all your trading partners
   NOTE: This is the default name for the file and is referred to in this document by that name but you can use any name you like – see the Certificate Configuration section below for how to set a different name.

4. **commands.xml** – the application provides a way to enter commands to control and configure the system whilst it is running either via the console or a remote tool (configured in the config.xml file above). This file stores the commands that the application will support.
   This file should not be modified

IMPORTANT: A restart of the application is required to load any configuration changes.

5.1. "home" Configuration Parameter

The folder containing the config.xml file defines the *home* configuration parameter that can be used to reference other files on the file system relative to a known base folder in the app. This is done by encapsulating *home* in percentage signs (*%home%*). All files can be referenced relative to this

parameter and it is how the default config.xml file defines the location of other configuration and data file locations used by the OpenAS2 application.

Therefore the default home location for %home% with the current default OpenAS2 folder structure will be [InstallDir]/config

6.   Application Configuration

The file named "config.xml" configures the modules that will be activated by the AS2 server when it starts up. This file can be located anywhere within the disk subsystem on which the OpenAS2 application runs as it is passed into the application as a startup parameter.

Some of the key configuration settings in the config.xml file are:

- define the modules to be activated in the OpenAS2 application

- override module default classes in the AS2 code base

- enhance or change behaviour of modules and the inputs and outputs of the modules.

- define the location of the certificates keystore and password

- define the location of the partnerships configuration file

- specify the listening ports

- enable support for high availability/load balanced environments

- change system behaviour via properties

See appendices for a detailed definition of the config.xml file structure.

There are 2 listening ports for inbound connections (see partnerships.xml config for outbound connections) used for:

1. receiving messages and synchronous MDN's – default port number 10080

2. receiving asynchronous MDN's - default port number 10081

The port numbers are arbitrary and defaulted to a number above 1024 that does not require root access to listen on (normally on Unix type systems any port below 1024 requires root access). The port values are important to the partner you will be communicating with if they will be sending AS2 messages to your system. For outbound only systems, it is only necessary to have a listener for asynchronous MDN's if using that mechanism for MDN's.

Each module has a number of attributes that can be configured on the module element to control and change how the module behaves.

All network modules that listen for inbound HTTP requests can be configured to bind to a psecific IP address (or host name) on the server using the "address" attribute. These modules by default will bind to localhost (127.0.0.1).

6.1. AS2 Message Tracking

As of version 2.1.0 the system will track key events in the message transmission and reception process and invokes any configured action handlers that can then process the information in some way. The default deployment of OpenAS2 supports a database tracking module that will write the message state to an embedded H2 database. As an AS2 message is processed, key points are logged to the database for a given message as a single record. As the message reaches the next state, the system overwrites the previous state.

The database tracking uses the module "org.openas2.processor.msgtracking.DbTrackingModule".

Configuration parameters for the database tracking functionality are shown in the table below.

Function	Attribute Name	Default Value
Database name	db_name	openas2
Database user name	db_user	sa
Database password	db_pwd	OpenAS2
Database file directory – used for embedded database	db_directory	`%home%/config/DB`
Use Embedded Database	use_embedded_db	true
Escape character for SQL strings	sql_escape_character	' (single quote)
JDBC connect string	jdbc_connect_string	jdbc:h2:$component.db_directory$/ $component.db_name$
JDBC Driver – not necessary if using at least JDBC 4.0	jdbc_driver	org.h2.Driver
Force loading of the JDBC driver class. Use if there are issues with JDBC	force_load_jdbc_driver	false
Provide JDBC access to H2 via TCP	tcp_server_start	true
H2 Listening port for the JDBC access	tcp_server_port	9092
H2 Password for the TCP server	tcp_server_password	openas2

Use of an external database can be configured for any database that has a JDBC driver such as Oracle, MySql or Postgresql.

To use an external database:

- put the appropriate JDBC driver jar for the SQL system you want to use into the "lib" folder of the OpenAS2 install
- the "**use_embedded_db**" attribute must be set to "**false**" and the appropriate settings changed in the database tracking module XML.

Below is a sample configuration for using Postgresql database:

```
<module classname="org.openas2.processor.msgtracking.DbTrackingModule"
   use_embedded_db="false"
   force_load_jdbc_driver="false"
   db_user="sa"
   db_pwd="OpenAS2"
   db_name="openas2"
   db_directory="%home%/DB"
```

```
    jdbc_driver="org.postgresql.Driver"
    jdbc_connect_string="jdbc:postgresql://localhost:5432/$component.db_name$"
    sql_escape_character="'"
/>
```

The user name and password can be changed using either a JDBC connection or the command line tool as described below. It is recommended that a readonly user is added for reading data from the database.

To connect to the H2 database whilst OpenAS2 is running use this JDBC connect string:

> jdbc:h2:tcp://localhost:9092/openas2

To query the database from the command line, you must have OpenAS2 running then use this command:

java -cp *[path to OpenAS2 install]*/lib/h2-1.4.192.jar org.h2.tools.Shell -user sa -password OpenAS2 -url jdbc:h2:tcp://localhost:9092/openas2

There is a file named db_ddl.sql file located in the config folder that can be used to create the necessary table structure if your DB becomes corrupted. The simplest way to recreate the database table is using this command whilst OpenAS2 is running:

java -cp *[path to OpenAS2 install]*/lib/h2-1.4.192.jar org.h2.tools.RunScript -user sa -password OpenAS2 -url jdbc:h2:tcp://localhost:9092/openas2 -script *[path to OpenAS2 install]*/config/db_ddl.sql

The above is for the version of H2 deployed with OpenAS2 version 2.1.0. If you use a different version of H2 then change the jar name to reflect this.

NOTE: The version of H2 deployed with the application only works in Java 7 or higher. Download the older version of H2 that was compiled with support for Java 1.6 if you wish to use Java 6: https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/h2database/h2-2014-01-18.zip (Replace the jar in the OpenAS2 lib folder with that version and change the startup script to include the replaced version of H2, delete the files in <installDir>/config/DB, restart OpenAS2 server and recreate the database using the command referenced above).

See appendixes for information on creating external database DDL statements and updating the existing database if the schema is changed in a new release.

6.2. Overriding Certificate Store Password

The certificate store password is stored as an XML attribute "password" on the <certificates> element. This can be overridden using the system property "***org.openas2.cert.Password***". For improved security, it may not be desired to store the password in the XML file.

This can be passsed into the application by adding the following to the java command:

- ***-Dorg.openas2.cert.Password=myCertificateStorePassword***

This can be set by using an additional parameter to the batch script file so that it can be set as part of invoking the script. The UNIX shell script will support the password as a parameter. The Windows bat file will need to be enhanced.

6.3. Resend Retry Configuration

When failures occur transferring a message to a trading partner, the system will automatically try to resend the message. By default the system will retry indefinitely.

IMPORTANT: A message that is put into the retry queue will use exactly the same parameters as when it was first sent. Therefore any changes to partnership such as destination URL, Async MDN response URL, signing algorithm etc. for that message after the first attempt to send the message will NOT be picked up by the message. The message must be deleted from the resend queue if changes were needed in the partnership definition and the message resent by passing in the file it was supposed to send again.

Restricting the retry attempts can be done at the processor level (applies to all partnerships configured on the server) and at the partnership level. Partnership configuration will override processor settings.

To define the processor level retry count, set the "**resend_max_retries**" attribute on the processor element to a valid integer.

> Example snippet:
>
> <processor classname="org.openas2.processor.DefaultProcessor"
> pendingMDN="%home%/../data/pendingMDN3"
> pendingMDNinfo="%home%/../data/pendinginfoMDN3"
> resend_max_retries="10" >

To define the partnership level retry count, set an attribute element on the partnership with ***name*** attribute value as "**resend_max_retries**" and a ***value*** attribute element to a valid integer.

> Example snippet:
>
> <partnership name="OpenAS2A-to-OpenAS2B">
>   <attribute name="resend_max_retries" value="3"/>
>   <sender name="OpenAS2A"/>

In the case of asynchronous MDN responses, the application will wait for a fixed amount of time (default is 300 seconds) for the partner to respond with an asynchronous MDN and then produce a fail message and move the sent file to the error directory as specified in the configuration.
To change the wait time before the application decides the partner will not respond can be done using a property in the config.xml file usinmg the attribute name
`"as2_mdn_response_max_wait_seconds"`. For example:
` <properties as2_mdn_response_max_wait_seconds="100" />`

6.4. File Name Parsing

The name of the file passed into the OpenAS2 application for sending to a remote partner can be used to provide information to the AS2 handler that can be used to affect various aspects of how the file is handled.

When the file is picked up for processing by the directory poller, it will look for an attribute named "**format**" on the AS2DirectoryPollingModule component for that directory in the config.xml file. This attribute is used to break the actual file name into multiple parts and the values assigned to the relevant objects defined in the "**format**" attribute.

An example format in the config.xml is:

> format="sender.as2_id, receiver.as2_id, attributes.filename"

The above format will set the AS2 sender and receiver ID as well as the name of the file sent in the AS2 message if configured to send a file name by tokenizing the file name using delimiters defined in the "**delimiter**" attribute. The delimiters for tokenizing defaults to "-." so the actual name of the file as picked up from the file system will be parsed and split into tokens using either a dash (-) or a period (.) and therefore using the above format, the name of the actual file would have to be in the format XXX-YYY-ZZZ or XXX.YYY.ZZZ or any combination of dash or period and then the AS2 sender would be set to XXX, AS2 receiver to YYY and the name of the file as ZZZ. Any extra string tokens will be discarded so for instance a file name of "X-Y-Z.edi" parsed against the format string above would simply discard the ".edi" part of the file name. If you want the file name sent to the remote partner to be Z.edi then the delimiter attribute value would just be "-".

The file name can also be configured to be parsed on a per partnership basis if a partner requires a different file name format to be sent or custom headers added to the AS2 message using partnership attributes as defined in section 7.7.2Dynamic Header Values From File Name.

6.5. Using A Proxy Server

The application uses the java.net.HttpURLConnection class for HTTP communication and as such should automatically use a proxy server if the appropriate system properties are set.

As of version 2.3.0, OpenAS2 also supports proxy server authentication and the setting of the proxy server host, port, username and password via the properties in the config.xml file. The settings in the config.xml will override any system properties passed in to the Java virtual machine from the command line.

The following properties are supported in the config.xml file:

- http.proxyHost
- https.proxyHost
- http.proxyPort
- https.proxyPort
- http.proxyUser
- http.proxyPassword

To bypass the proxy for certain destination hosts you will need to use the system property

This is an example of a proxy server configuration using OpenAS2 properties for HTTP connections:

```xml
<openas2>
  <properties
    http.proxyHost="192.168.1.1"
    http.proxyPort="1099"
    http.proxyUser="acme"
    http.proxyPassword="secret"
  />
  <certificates classname="…
  ….
  ….
</openas2>
```

This article provides the basics:
https://blogs.oracle.com/wssfc/entry/proxy_server_authentication_in_java

6.6. Health Check For High Availability Deployment

There are 2 ways to implement a health check.

1. Use a specific URI on the existing AS2 receiver HTTP listener – provides a simple HTTP 200 response but does not do any checking of application status

2. Use a dedicated healthcheck module that runs on a separate HTTP port and provides extended checking of the status of the application and its modules.

6.6.1. Healthcheck URI On Exisitng AS2 Listener

This mechanism simply provides validation that the application is listening on the specified socket and does not provide any verification of the health of other components of the application.

The AS2 receiver module can be configured to recognize a specific URI as a health check request instead of a normal AS2 message. By default, the healthcheck URI is set to "/healthcheck". The full URL would be the combination of host name/ip address and port number with the URI as suffix.

e.g http://localhost:10080/healthcheck

6.6.2. Dedicated Healthcheck Module

As of version 2.4.0, a health check module is included in the OpenAS2 deployment.

This module adds a listener on a specified HTTP or HTTPS port that can be invoked by a load balancer or other application to check if the application is running.

It is invoked by any GET request on the specified host and port irrespective of URI

e.g http://localhost:10080/ or http://localhost:10080/some/random/uri

Currently the module will invoke a health check method for all active modules and returns HTTP 200 OK if there are no errors. It checks all configured listeners that the socket is active and responding as well as relevant health checks for other modules such as the database tracking module, resender modules and directory polling modules.

If errors are detected, it will return HTTP 500 Internal Error with a list of errors detected in the body of the response.

This allows you to configure the load balancer to send an HTTP request to the configured port and will return a 200 OK response.

To enable the module you need to uncomment the definition at the bottom of the config.xml file setting the port as appropriate for your environment. Below is the sample entry in the config.xml file:

```
<module classname="org.openas2.processor.receiver.HealthCheckModule" port="10099"/>
```

If desired you can bind the healthcheck module to a specific IP address using the "address" attribute:

```
<module classname="org.openas2.processor.receiver.HealthCheckModule" address="10.0.2.1"/ port="10099"/
>
```

HTTPS transport can be configured as per section Configuring HTTPS Transport in this document.

See the appendix for notes on deploying OpenAS2 in a clustered/load balanced environment.

6.6.3. HTTP User Agent Header

By default, the "User-Agent" header sent in the HTTP requests will contain the application title and version along with the module name sending the request. This can be overridden using the property name "http.user.agent". For example:

```
<properties
    http.user.agent="AS2 1.1 Compliant Server"
    log_date_format="yyyy-MM-dd HH:mm:ss.SSS"
    sql_timestamp_format="yyyy-MM-dd HH:mm:ss.SSS"
/>
```

7.   Partner Configuration

The file named partnerships.xml configures all the information relating to the partners you will be exchanging data with. See the appendix for information on the structure of this file.

The "partnerships" element in the application configuration file (described in section Application Configuration above) causes the partnerships to be loaded by the application. The default entry in the application configuration file for this element is as below:

```
<partnerships classname="org.openas2.partner.XMLPartnershipFactory"
    filename="%home%/partnerships.xml"
    interval="120"/>
```

The "interval" attribute specifies the number of seconds between each check for a changed partnership file. If a change of the partnership file is detected, the partnership file will be automatically reloaded.

It is important to keep in mind that the word *partner* refers to any entity specified as a recipient or sender of AS2 messages and includes your own company that you might be configuring the application for.

Each partner will require the following entries in the file:

- a **\<partner\>** element – key information defining the partner

- a **\<partnership\>** element - key information for defining a partnership between 2 partners
  Separate **\<partnership\>** elements are required for inbound and outbound data for a specific partner pairing.
  NOTE:It is necessary to have 2 elements even if data transfer is unidirectional.

7.1. Partner Definition

The **<partner>** element requires 3 attributes to enable AS2 partner identification:

1. partner name – this is the key to connect partnerships to a partner definition

2. AS2 identifier – this is the key for identifying the target/source partner and is included in AS2 message headers to allow the receiving partner to identify the source of the message and verify the target partner for the AS2 message. It is also used by the general directory polling module to look up the partner names and hence the partnership definition where the as2_id of the sender and receiver are part of the transferred file name.

3. X.509 certificate alias – identifies the alis of the certificates for this partner in the keystore. The encryption and decryption of messages requires the partners public or private key as appropriate

7.2. Partnership Definition

The <partnership> element identifies a **specific direction** of AS2 message transfer **from** one partner **to** another. The "name" attribute on the <partnership> element is not important but should be used to clearly identify the intended use of the partnership definition. It is suggested the name value uses the names of the source and destination partners something like xxx-to-yyy.

The <partnership> element encapsulates a number of child elements that are necessary to properly configure a partnership:

- <sender name="*xxx*"> - identifies the sending partner definition such that *xxx* must match the "name" attribute of a <partner> element

- <receiver name="*yyy*"> - identifies the receiving partner definition such that *yyy* must match the "name" attribute of a <partner> element

- <as2_url> - a fully qualified URI that provides the connection string to the remote partner for sending AS2 messages. If sending to another OpenAS2 server then the port number must match the value configured in the config.xml file of the remote OpenAS2 server.

- <as2_mdn_to> - neccesary if an MDN response is required and can be any random string but is most commonly configured with an email address

The partnership element attribute values supports using a key value string that can be used to reference other attributes within the same partnership element to ensure consistent configuration. The value field in an attribute can use the format *$attribute.XXX$* to reference the value of the attribute name "XXX". The value in the attribute value containing the reference to another attribute will be replaced at load time with the referenced attributes value.

For instance you can have :

```
<attribute name="some_attrib_name" value="My value: $attribute.other_attrib"/>
<attribute name="other_attrib" value="bingo"/>
```

The value for `"some_attrib_name"` will be `"My value: bingo"` after the application starts up.

For an implemented example see the MDN MIC Algorithm section below.

NOTE: This feature does cascade the replacement. ie it will NOT replace values in attribute values that themselves contain references to other attributes. It may appear to cascade depending on the order in which attributes are processed but the processing order is not guaranteed so will not produce a reliable result

7.2.1. Signing

Signing is controlled by the "sign" attribute in the "partnership" element. Remove this element from the partnership to send a message without signing.

7.2.2. Encryption

Encryption is controlled by the "encrypt" attribute in the "partnership" element. Remove this element from the partnership to send a message without encryption.

7.2.3. MDN MIC Algorithm

The MDN must be signed using the same algorithm that the sent message was signed with. The recipient partner is told the signature algorithm via a field in the "**as2_mdn_options**" attribute and the value uses the **$attribute.sign$** dynamic variable to ensure it matches the "**sign**" attribute as shown in the example below:

```
<attribute name="as2_mdn_options"
              value="signed-receipt-protocol=optional, pkcs7-signature; signed-
receipt-micalg=optional, $attribute.sign$"/>
```

If you receive errors something like "mismatched Message Digest" or similar messages then ensure you have this attribute set correctly.

If you need to send an unsigned MDN then set the attribute on the partnership as below:

```
<attribute name="as2_mdn_options" value="none"/>
```

7.3. Configuring the AS2 Message ID

The message ID used for uniquely identifying the message sent to a partner defaults to the following format:

OPENAS2-$date.ddMMyyyyHHmmssZ$-$rand.UUID$@$msg.sender.as2_id$_$msg.receiver.as2_id$

To change this globally for all partnership definitions, change the property in the config.xml file to the desired format string using the attribute name "**as2_message_id_format**" in the "**properties**" element.

```
<properties
    log_date_format="yyyy-MM-dd HH:mm:ss.SSS"
    sql_timestamp_format="yyyy-MM-dd HH:mm:ss.SSS"
    as2_message_id_format="&lt;OPENAS2-$date.ddMMyyyyHHmmssZ$-
$rand.UUID$@$msg.sender.as2_id$_$msg.receiver.as2_id$>"
/>
```

The message ID that is generated can be overridden on a partnership by partnership basis. To set it for a particular partnership, add an attribute named "**as2_message_id_format**" to the partnership

definition and use any dynamic parameters as specified in the dynamic parameters section of this document.

Something like this:

```
<partnership name="OpenAS2B-to-OpenAS2A">
      <sender name="OpenAS2B"/>
      <receiver name="OpenAS2A"/>
      <attribute name="protocol" value="as2"/>
      <attribute name="content_transfer_encoding" value="binary"/>
      <attribute name="as2_message_id_format" value="ACME-$date.yyyyMMddHHmmssZ$-
$rand.UUID$"/>
```

In some cases, the choice of the Message ID will not be suitable for generating an MDN message Id because the parameter strings will not have a value value such as when the message ID references some parts of the parsed file name and would return a "null" value in the MDN context. In this case you can specify a different format for the MDN message ID that is generated and this can be set at partnership or system level as above.

```
<properties
    log_date_format="yyyy-MM-dd HH:mm:ss.SSS"
    sql_timestamp_format="yyyy-MM-dd HH:mm:ss.SSS"
    as2_mdn_message_id_format="&lt;OPENAS2-MDN-
$rand.UUID$@$msg.sender.as2_id$_$msg.receiver.as2_id$>"
    />
```

To set it for a particular partnership, add an attribute named "**as2_mdn_message_id_format**" to the partnership definition and use any dynamic parameters as specified in the dynamic parameters section of this document.

Something like this:

```
<partnership name="OpenAS2B-to-OpenAS2A">
      <sender name="OpenAS2B"/>
      <receiver name="OpenAS2A"/>
      <attribute name="protocol" value="as2"/>
      <attribute name="content_transfer_encoding" value="binary"/>
      <attribute name="as2_mdn_message_id_format" value="ACME-MDN-
$date.yyyyMMddHHmmssZ$-$rand.UUID$"/>
```

7.4. Content Transfer Encoding

As of version 1.3.7, the default content transfer encoding uses "**binary**" if not explicitly overwritten in the configuration. The default can be changed using the "*content_transfer_encoding"* attribute in the partnership.xml file. If you experience issues with failing to verify a partners AS2 inbound message because the message contains CR/LF data in it then you should switch to using "binary" for the transfer encoding. The sample partnership file sets the transfer encoding to "binary" for both partners.

IMPORTANT NOTE: The Content-Transfer-Encoding header is a restricted HTTP header and will not be sent in the HTTP headers but some systems will fail if not sent – see the trouble shooting section for restricted HTTP headers to manage this

7.5. *Supported Encoding Algorithms*

The currently supported encoding algorithms are:

- MD5
- SHA1
- SHA224
- SHA256
- SHA384
- SHA512
- CAST5
- 3DES
- IDEA
- RC2_CBC
- AES128 (CBC mode)
- AES192 (CBC mode)
- AES256 (CBC mode)
- AES256_WRAP

7.6. *Message Compression*

The application supports inbound compression automatically. There is no configuration for this option. To enable outbound compression requires setting "***compression_type***" attribute on the partnership definition for the outbound configuration. The only supported compression/decompression at this time is "***ZLIB***". The default is no compression of sent messages.

By default compression will occur on the message body part prior to signing. The compression can be configured to occur after signing using the "***compression_mode***" attribute on the partnership definition for the outbound configuration. Set the attribute to "***compress-after-signing***" to enable this.

See partnership.xml appendix for configuration details.

7.7. *Custom Mime Headers*

Mime headers can be added to the outermost Mime body part for outbound messages and additionally added to the HTPP headers. The outermost Mime body part will depend on configuration of the partnership and could be the compressed, signed or encrypted part. In the case of the encrypted part being the outermost mime body part, the HTTP headers will not be visible until after decryption of the body part since encryption protects the content and the headers.

7.7.1. Static Header Values

Custom headers can be added as statically defined name/value pairs in a partnership attribute where the name and the value are separated by a colon. Multiple static headers are added using a semi-colon separated list between each name/value pair. The attribute name for this is "***custom_mime_headers***" and a sample entry of 2 static headers is shown below:

```
<attribute name="custom_mime_headers" value="X-CustomRoute: X1Z34Y ; X-CustomShape:oblong"/>
```
Note that spaces before or after the ";" and ":" separators will be excluded.

7.7.2. Dynamic Header Values From File Name

Dynamic headers require 2 attributes to configure their behaviour and there are 2 different

modes of operation for extracting the value(s) for the defined header(s) from the file name:

1. delimiter mode

2. regular expression mode

 Delimiter mode is relatively simple and does not require any special knowledge but regular expression mode may require someone with regular expression skills. Regular expression mode provides far greater flexibility for extracting the value(s) from a file name where specific character sequences or character counts are required.

Both modes use an attribute named "**custom_mime_header_names_from_filename**" to enter the list of header names but the format for the two are slightly different. The second attribute required has a different name for each of the modes, "**custom_mime_header_name_delimiters_in_filename**" for delimiter mode and "**custom_mime_header_names_regex_on_filename**" for regular expression mode.

IMPORTANT: if both delimiter mode and regular expression mode attributes are entered into a partnership then delimiter mode will be chosen irrespective.

Delimiter Mode

In delimiter mode, the values in the file name are separated by specifiying one or more delimiters and the entire file name is parsed into a list of values using the delimiter(s) defined. In order to accommodate file names that have more than just the values required for the custom headers, the list of header names are defined with a prefix that designates if the value in the list will be used as a header value or not. For an entry to be added as a header it must have the prefix "header.". Any other prefix will cause that entry to be ignored. There must be as many header names defined as there are string sequences that would result from splitting the file name string by the delimiter(s) otherwise the system will throw an error.

Below is an example of a delimiter based configuration.

```
<attribute name="custom_mime_header_names_from_filename"
        value="header.X-Header1,header.Y-Header2, junk.extraStuff"/>
 <attribute name="custom_mime_header_name_delimiters_in_filename" value="-_"/>
```
Using this configuration, given a file name **ABC-123-INVOICES.csv** there would be 2 headers added as:
X-Header1   value ABC
Y-Header2   value 123
If the file name was **ABC-123-H4FT_INVOICES.csv** the system would throw an error as there would be 4 string sequences extracted so you could fix this by appending junk.moreStuff to the "custom_mime_headers_from_filename" attribute.
Another example of delimiter mode in the partnership:
```
<attribute name="custom_mime_header_names_from_filename"
        value="header.X-Header1, other.string1,header.Y-Header2"/>
 <attribute name="custom_mime_header_name_delimiters_in_filename" value="-"/>
```
Using this configuration, given a file name **ABC-123_TEST-INVOICES.csv** there would be 2 headers added as:
X-Header1   value ABC
Y-Header2   value INVOICES

Regular Expression Mode

Regular expression based mode uses Java regular expressions and requires that the regular

expression is constructed in grouping mode where the number of groups in the regular expression exactly matches the number of header names in the "**custom_mime_header_names_from_filename**" attribute. The regular expression will be used to parse the file name to extract the values for the defined names in the attribute named "**custom_mime_header_names_regex_on_filename**". Regular expressions can become extremely complex and this document will show some simple examples but there are many sites that provide regular expression tutorials if you need a complicated soultion.

An example for a regular expression mode configuration is shown below:

```
<attribute name="custom_mime_header_names_from_filename" value="X-Header1,Y-Header2"/>
<attribute name="custom_mime_header_names_regex_on_filename" value="([^-]*)-([^.]*).csv"/>
```

Using this configuration, given a file name **ABC-123-INVOICES.csv** there would be 2 headers added as:

X-Header1   value ABC
Y-Header2   value 123-INVOICES

If the file name was **ABC-123-H4FT_INVOICES.csv** there would be 2 headers added as:

X-Header1   value ABC
Y-Header2   value 123—HFT_INVOICES

If the file name was **ABC-123-H4FT_INVOICES.txt** or **ABC_123.csv** the system would throw an error since there would be no match.

Another example for a regular expression mode configuration is shown below:

```
<attribute name="custom_mime_header_names_from_filename" value="X-Header1,Y-Header2"/>
<attribute name="custom_mime_header_names_regex_on_filename" value="([^-]*)-([^.]*).csv"/>
```

Using this configuration, given a file name **ABC-123-INVOICES.csv** there would be 2 headers added as:

X-Header1   value ABC
Y-Header2   value 123-INVOICES

7.7.3. Adding Custom Headers To HTTP

The following attribute set to a value of "true" will additionally add the headers to the HTTP headers for both static and dynamic header mechanisms:

```
<attribute name="add_custom_mime_headers_to_http" value="true"/>
```

7.8. *Setting Dynamic Attributes From File Name*

Partnership attributes can be added to the partnership definition based on parsing the file name of the document to be sent using a regular expression. Dynamic attributes require 2 partnership attributes to configure their behaviour for extracting the value(s) for the defined attribute(s) from the file name.

1. "**attribute_names_from_filename**" - when added to a partnership it must contain a list of comma separated attribute names

2. "**attribute_values_regex_on_filename**" - defines the regular expression

The extracted name/value pairs can then be referenced in config using the format:

$attributes.<attribute name>$

Regular expressions uses Java regular expressions and requires that the regular expression is constructed in grouping mode where the number of groups in the regular expression exactly matches the number of attribute names in the "**attribute_names_from_filename**" attribute. Regular expressions can become extremely complex and this document will show some simple

examples but there are many sites that provide regular expression tutorials if you need a complicated solution.

An example for a regular expression mode configuration is shown below:

```
<attribute name="attribute_names_from_filename" value="X-attribute1,Y-attribute2"/>
<attribute name="attribute_values_regex_on_filename" value="([^-]*)-([^.]*).csv"/>
```

Using this configuration, given a file name **ABC-123-INVOICES.csv** there would be 2 attributes added as:

X-attribute1   value ABC
Y-attribute2   value 123-INVOICES

If the file name was `ABC-123-H4FT_INVOICES.csv` there would be 2 attributes added as:

X-attribute1   value ABC
Y-attribute2   value 123—HFT_INVOICES

If the file name was `ABC-123-H4FT_INVOICES.txt` or `ABC_123.csv` the system would throw an error since there would be no match.

Another example for a regular expression mode configuration is shown below:

```
<attribute name="attribute_names_from_filename" value="X-attribute1,Y-attribute2"/>
<attribute name="attribute_values_regex_on_filename" value="([^-]*)-([^.]*).csv"/>
```

Using this configuration, given a file name **ABC-123-INVOICES.csv** there would be 2 attributes added as:

X-attribute1   value ABC
Y-attribute2   value 123-INVOICES

The above attributes could be referenced in config to set a more dynamic subject using something like this:

```
<attribute name="subject" value="Target product: $attributes.X-attribute1$ Sequence Count:
$attributes.Y-attribute2$"/>
```

This would produce a subject looking like this:

```
Target product: ABC Sequence Count: 123-INVOICES
```

8.   Certificate Configuration

The first step is to set up your own private certificates so that you can accept inbound messages and an associated public certificate that your partner will use to encrypt and sign the message they send to you and for any MDN responses that your partner returns as a result of sending them a message.

Once you have your own certificates set up then you can import partner public certificates for signing and encrypting messages to be sent to them.

The certificate store used by default is a PKCS12 key store and stores all X.509 certificates for all trading partners.

The "**certificates**" element in the application configuration file (described in section Application Configuration above) causes the certificates stored in the keystore pointed to by the "**filename**" attribute to be loaded by the application. The default entry in the application configuration file for this element is as below:

```
<certificates classname="org.openas2.cert.PKCS12CertificateFactory"
    filename="%home%/as2_certs.p12"
    password="testas2"
    interval="300"/>
```

The "**password**" attribute specifies the password to open the keystore.

The "**interval**" attribute specifies the number of seconds between each check for a changed certificate file. If a change of the certificate file is detected, the certificate file will be automatically reloaded.

The application supports certificate management via the command interface but the functionality is limited and you will need to use 3rd party tools for creating and manipulating certificates to a format that can be used in the OpenAS2 command interface.

An excellent open source visual keystore manager that will run on any OS and will allow importing and managing certificates in your keystore can be found here: http://portecle.sourceforge.net/

8.1. Possible Issues With Older Certificates

With the latest version of cryptographic libraries it is possible that importing older certificates will cause a failure relating to certificates with something like this in the error:

Caused by: java.lang.IllegalArgumentException: invalid info structure in RSA public key

See this discussion for more information: https://github.com/OpenAS2/OpenAs2App/issues/98

To solve this problem if you have to use an older certificate, add the following option to the startup script (in the latest scripts it is in the file but commented out):

```
-Dorg.bouncycastle.asn1.allow_unsafe_integer=true
```

8.2. My Certificates

You can have multiple certificates per trading partner (each partner is sent a unique certificate and there is a matching private key for each certificate in your keystore). Alternatively you can have a single certificate for all trading partners (all partners are sent the same certificate and there is only one private key in your keystore to match). For increased security it is highly recommended you use multiple certificates.

Your own certificate(s) will always be imported from a keystore that contains both the private key and the certificate.

NOTE: SHA1 certificates are no longer supported and are rapidly being phased out so you should use SHA256 for all partners that do support SHA256 certificates.

8.2.1. Creating Certificates And Importing Into OpenAS2 Keystore

There are many tools for creating certificates. This document focusses on using openssl but it can be done using the Java keytool application in similar way.

The key store must contain the private key of your own X.509 certificate and the signed certificates for each of your trading partners.

The certificates must be stored with the matching alias as specified in the partner definition of each partner in the partnership.xml file.

The OpenAS2 command processor (or remote OpenAS2 app) import command for a certificates

contained in a PKCS12 keystore would be in this format:

> c**ert import** *<alias> <path+filename> <keystore password>*

There is a shell file to help generating certificates: **bin/gen_p12_key_par.sh**

The following steps will create an X509 self signed certificate using OpenSSL:

> **openssl req -x509 -newkey rsa:4096 -keyout priv.key -out selfcert.crt -days 3650 -sha256**

To import the public certificate and private key you create a temporary PKCS12 key store that will be used to import the public/private key pair:

> **openssl pkcs12 -export -in selfcert.crt -inkey priv.key -out certs.p12 -name my_new_alias**

The file named **certs.p12** is now a PKCS12 keystore containing the private key and the certificate and can be imported into the OpenAS2 keystore via the command interface using a command like this:

> c**ert import my_new_alias** *certs.p12 <keystore password>*

If you are replacing your existing keystore completely then simply delete the existing keystore and copy the new one into the same folder with the same name as the old one (of course this will delete any partner certificates you may have imported to the old key store).

NOTE: It is important to use ".p12" as the extension when importing certificates from a PKCS12 keystore as the importer requires the ".p12" extension to detect that you are not importing a certificate directly but rather the certificates in a PKCS12 keystore.

8.2.2. Creating Public Certificate For Sending To Partner

Most systems will support a base64 (ASCII) encoded PEM format. If your partner needs a different format there are numerous methods to convert certificates to other formats including using openssl but that is not covered in this document.

You can create the certificate using this command against the keystore that contains the private key and certificate that you want create it from:

> **openssl pkcs12 -clcerts -nokeys -out <output file> -in <keystore file>**

This should be run against the temporary keystore when creating the private key and certificate in the section above as it will create a PEM file containng only that specific certificate.

You can run it against the keystore as configured in the OpenAS2 application but it will export all certifictes in the keystore and you will have to edit the file to extract only the one you are interested in.

8.2.3. Supporting Multiple Certificates

In the case where you need to support multiple certificates such as when one partner needs SHA1 and another needs SHA256 or when you want to set up different certificates per partner, follow these steps below.

The key to supporting multiple certificates is ensuring you use a separate ***as2_id*** and ***x509_alias*** attribute.

In the partnership.xml you would add another partner element pointing to a different certificate.

If for example you have a <partner> element definition for your company as below:

```
<partner name="MyCompany" as2_id="MyCompany_OID" x509_alias="MyCompanyCert"
email="me@MyCompany.com"/>
```

For each additional certificate you support, you then add another <partner> element. If for instance you have SHA1 already deployed and working with existing partners and you create a SHA256 certificate to support a new partner, you add a new <partner> element something like this:

```
<partner name="MyCompany256" as2_id="MyCompany2_OID"
x509_alias="MyCompanyCert256" email="me@MyCompany.com"/>
```

In your partnership definition for the partners using the SHA256 certificate you set the "**sender**" and "**receiver**" attribute as appropriate to point to the correct partner definition ("*MyCompany256*" per the example above) along with changing the SHA1 to SHA256 in the other relevant attributes as shown in the snippet below.

```
<partnership name="MyCompany256-to-MyPartner256">
  <sender name="MyCompany256"/>
  <receiver name="MyPartner256"/>
  <attribute name="protocol" value="as2"/>
  ...
</partnership>
```

Import the new certificate into the existing p12 keystore using the *alias* as defined in the *x509_alias* attribute above ("*MyCompany2Cert256*") and send the partner the matching public key for the new certificate along with the as2_id "*MyCompany256_OID*" that they will need to use so you can differentiate your target definition in the partnership file containing the SHA1 certificate from the SHA256 certificate. See the previous section for importing certificates into your existing keystore.

8.3. Partner Certificates

The certificate(s) that you will obtain from your partner(s) will need to be imported into the keystore you have created for your own certificate. The partner certificates must be imported with unique aliases so that they can be uniquely referenced from the partnership configuration in the same way described in the section above for your own certificates.

As per the section above dealing with your own certificates, you can either use the OpenAS2 command interface or a 3rd party tool to import partner public keys into the keystore.

The OpenAS2 command processor (or remote OpenAS2 app) import command for importing your partners certificate would be in this form:

c**ert import** *<alias> <path+filename>*

9.   Logging System

Loggers are configured in the config.xml file using the "loggers" element.

9.1. Log Output Targets

The logging output can be directed to to multiple destinations including:

- System console
- File system log files
- Email – log messages are emailed to a configured email address.
- Socket – log messages are written to a socket supporting remote logging
- Sentry – support for the Sentry logger that provides logging management for exceptions

By default the OpenAS2 system deploys with the console and file system loggers enabled.

All log classes can be overridden or custom logger classes can be coded and included via configuration to support custom logging applications or SaaS log management systems.

9.1.1. Console Logger

The console logger simply logs to the shell/command window that the server is started in or if not started from a shell/command window then it logs to whatever System.out is connected to. The console logger is configured using this entry in the <loggers> element:

```
<logger classname="org.openas2.logging.ConsoleLogger" />
```

9.1.2. File Logger

The file logger will log to a file on a file system. The file system can be a netwrok file share as long as it has write permissions. The file system directory and file name are configured in the "filename" attribute. The file logger is configured using this entry in the <loggers> element:

```
<logger classname="org.openas2.logging.FileLogger"
        filename="%home%/../logs/log-$date.yyyyMMdd.txt" />
```

9.1.3. Email Logger

The email logger uses the javax mail API to send ERROR level log messages – all log entries below ERROR level are ignored. Some of the basic email configuration parameters are supported via config in the config.properties file as indicated in the appendix. The rest of the mail properties as listed in the Javamail API can be set by passing them as system properties on the command line by modifying the start-openas2.sh or start-openas2.bat file as appropriate or using the ***javax.mail.properties.file*** attribute on the email logger element.

The configuration values can overwrite each other depending on the source of the configurtion value. The order of priority is as follows:
1. values set in the logger element attributes
2. entries in the file identified by ***javax.mail.properties.file***
3. entries using system properties

For example, to pass the port for connection you could add this to the command line: -Dmail.smtp.port=529

To point to a properties file containing all the relevant information you would add something like this:
```
<logger classname="org.openas2.logging.EmailLogger"
```

```
                    javax.mail.properties.file="%home%/java.mail.properties"
                    from="openas2"
                    …
```

9.1.4. Socket Logger

This logger writes to a socket with connection parameters as specified in the attribues for this logger:

```
        <logger classname="org.openas2.logging.SocketLogger"
                ipaddr="127.0.0.1"
                portid="12345" />
```

9.1.5. Sentry Logger

This logger provides the ability to use the Sentry logging system (http://www.sentry.io). The configuration is as follows:

```
        <logger classname="org.openas2.logging.SentryLogger"
                dsn="SENTRY DSN" />
```

9.2. Log Level Configuration

The logging system supports the use of either or both the ***commons-logging.properties*** file or a file named ***openas2log.properties*** to control the logging level. Properties in openas2log.properties will override commons-logging.properties entries. There is a commons-logging.properties file in the ***bin*** directory which is part of the classpath specified in the script file described in the section on running the application. The default batch script uses a CLASSPATH setting that includes the current working directory that you are starting OpenAS2 application from and all files in the lib folder (-cp .:${binDir}/../lib/*). This means that if you are invoking the batch script from a folder other than the folder the batch script is in then it will not "see" the commons-logging.properties file in the bin folder. The solution is to set the classpath to point to the bin folder (-cp ${binDir}/:$ {binDir}/../lib/*) or move the commons-logging.properties file to the folder you are invoking the script from.

The properties in the ***openas2log.properties*** file should be prefixed by "`org.openas2.logging.`"
The following are the logging levels supported by the application in order of lowest(finest) to highest:
"TRACE", "DEBUG", "INFO", "WARN", "ERROR", "FATAL"
The logging levels are turned off by specifying the level you want on and all other levels higher than that level will also be turned on.
The default level is INFO and therefore WARN, ERROR and FATAL are also turned on by default.
By adding a property level=DEBUG in the common-logging.properties file will result in DEBUG logging being enabled along with INFO, WARN, ERROR and FATAL
The same can be achieved by adding org.openas2.logging.openas2log.level=DEBUG in the openas2log.properties file.

The default deployment has logging level set to INFO.

9.3. Log Date Format Configuration

The default format for the timestamp prefixed to all log entries is the international standard

including millisecond precision as of version 2.3.1 (yyyy-MM-dd HH:mm:ss.SSS).

The format can be changed by setting the attribute named "log_date_format" in the "properties" element of the config.xml with the desired format. The format string must comply with the Java SimpleDateFormat specification (e.g for Java 8 - https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html)

10. MDN Configuration

MDN's can be sent synchronously or asynchronously. By default the system will use synchronous MDN mechanism. Per the AS2 specification, an MDN will only be sent on receipt of an AS2 message if the "**Disposition-Notification-To**" header is present in the received message with a non-empty value. Although this value is specified to be configured with an email address, it is not utilized for any purpose in the AS2 protocol other than to indicate an MDN is required so can in fact be any random string. To set the "**Disposition-Notification-To**" header in an outbound message, the "**as2_mdn_to**" attribute must be set on the partnership.

The other attribute that must be set is the "**as2_mdn_options"**. This defines the encryption algorithm and other MDN settings as specified by the AS2 protocol and the value entered for this attribute will be sent in the "**Disposition-Notification-Options**" header of the AS2 message. Generally changing the encryption algorithm to suit the trading partner should be sufficient on this attribute.

10.1. Asynchronous MDN Receiver Configuration

In order to specify an asynchronous MDN response from a partner requires setting the following attribute on the partnership element in the partnership configuration:

- **as2_receipt_option** – set to the URL of the asynchronous MDN receiver to target the asynchronous MDN receiver module configured in the config file (ie. this is the URL that the partner will send the MDN to). The value set in this attribute will be sent in the "**Receipt-Delivery-Option**" header of the AS2 message to the trading partner. For testing using the default config file that comes with the OpenAS2 installation package, set this to: http://localhost:10081

Receiving an asynchronous MDN requires the "***AS2MDNReceiverModule***" module. This module declaration requires a port parameter in addition to the class and can be entered as a child member of the processor node in the config file as shown below:

<module classname="org.openas2.processor.receiver.AS2MDNReceiverModule" port="10081" />

```
<module classname="org.openas2.processor.receiver.AS2MDNReceiverModule" port="10081"/>
```

If desired you can bind the module to a specific IP address using the "address" attribute:

```
<module classname="org.openas2.processor.receiver.ASMDNREceiverModule" address="10.0.2.1"/
port="10081"/>
```

10.2. MDN Sender Configuration

Sending an asynchronous or synchronous MDN requires the "**MDNSenderModule**" module. This module declaration does not require any parameters other than the class and can be entered as shown below as a child member of the processor node in the config file:

<module classname="org.openas2.processor.sender.MDNSenderModule" />

11. Configuring HTTPS Transport

HTTPS transport using SSL is configured separately for inbound and outbound connectivity.

11.1. Inbound Transfers

Configuration for inbound is in the config.xml file. The requirements for receiving AS2 files using HTTPS are:

- JKS keystore containing the SSL certificate
- an appropriately configured As2ReceiverModule module element

The key attributes that configure HTTPS are:

- `protocol="https"`
- `ssl_keystore="%home%/ssl_certs.jks"` – points to the JKS certificate keystore
- `ssl_keystore_password="<passwordforkeystorefile"`
- `ssl_protocol="TLS"`

See the appendix for details on the attributes.

11.2. Outbound Transfers

The partnership definition for the connection URL will also have to be set to the appropriate host name.

The key attributes that configure HTTPS are:

- `as2_url`
- `as2_mdn_to (only if MDN is required)`

If asynchronous MDN is in use and requires HTTPS then a **As2MDNReceiverModule** module needs to be configured in the same way as for the As2ReceiverModule class above.

If the target system being connected to uses self signed certificates, the following system property will have to be passed to the application in the java command line with a comma separated list (no spaces before or after comma) of the "Common Name" (CN) in the self signed certificate that will be returned by the target system:

`-Dorg.openas2.cert.TrustSelfSignedCN=<Common.Name1>,<Common.Name2>,…`

12. Running OpenAS2

OpenAS2 can be started from the command line in any operating system that supports Java or can be configured to run as a daemon using the appropriate mechanisms for the operating system.

The default deployment for OpenAS2 has a console logger enabled which means that all logging will be visible in the command line window that OpenAS2 is started from. The server can also be configured from the command line once the application is running by simply typing in commands once it has started. Because the logging will appear in the window it may make command entry difficult if there are active transfers at the time you try to enter commands and it may be desirable to switch off the console logger if you have no need for it.

12.1.   Starting OpenAS2

The default install of the application is as in the figure below from a windows PC.
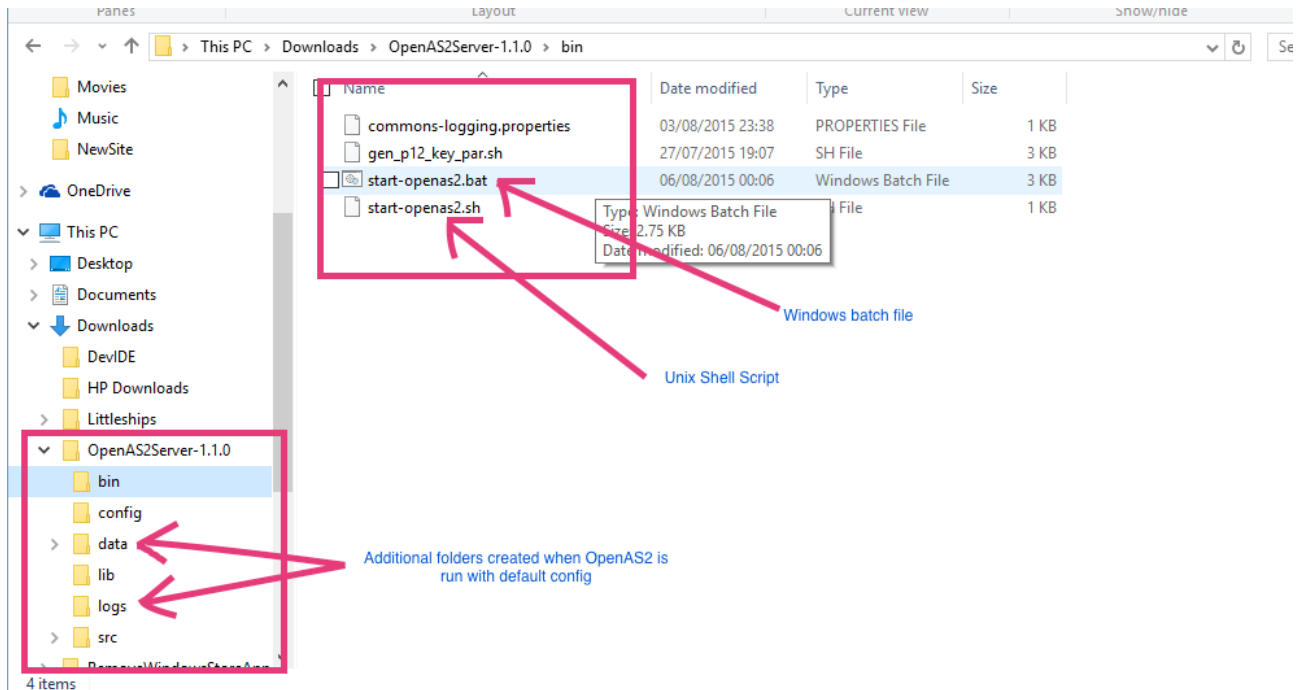
There are 2 executable script files in the *bin* folder of the AS2 application root as indicated in the screenshot above:
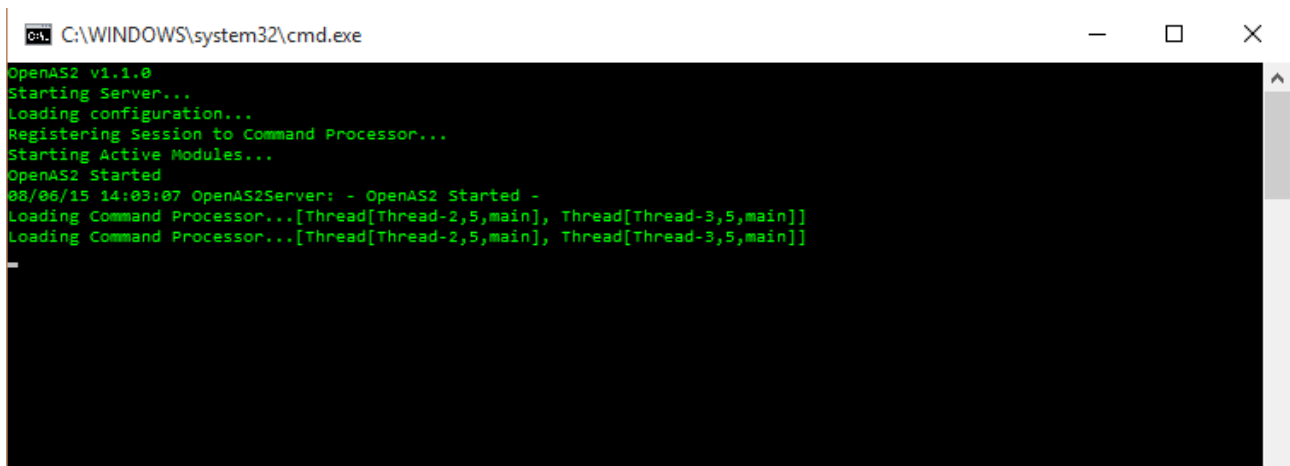
1.   start-openas2.sh – for UNIX based systems

2.   start-openas2.bat – for Microsoft Windows based system

It is not necessary to modify these files for the default install to work. If you choose to put the config.xml file in a different location than the default then you will need to edit the appropriate script file and set the path to the config.xml file appropriately.

Simply execute the script file and an AS2 server will start up. It will create the following folders along with sub folders when it starts assuming no change to the default config:

- logs – contains the normal program logging
- data – contains all the transferred files and any AS2 specific headers associated with AS2 transfers. This folder will have a number of sub folders for outbound and inbound files for different partners

In Microsoft Windows you should be able to double click the start-openas2.bat file and a command window will open as below.

For Unix based systems such as Linux and OSX, open a terminal window and change directory to the "bin" folder of the install. The start_openas2.sh file should have execute permissions in which case simply type the name and press enter. If no execute permissions are set, either set the execute permission as needed or use "bash" to run the script:

/opt/OpenAS2:>bash opensas2.sh
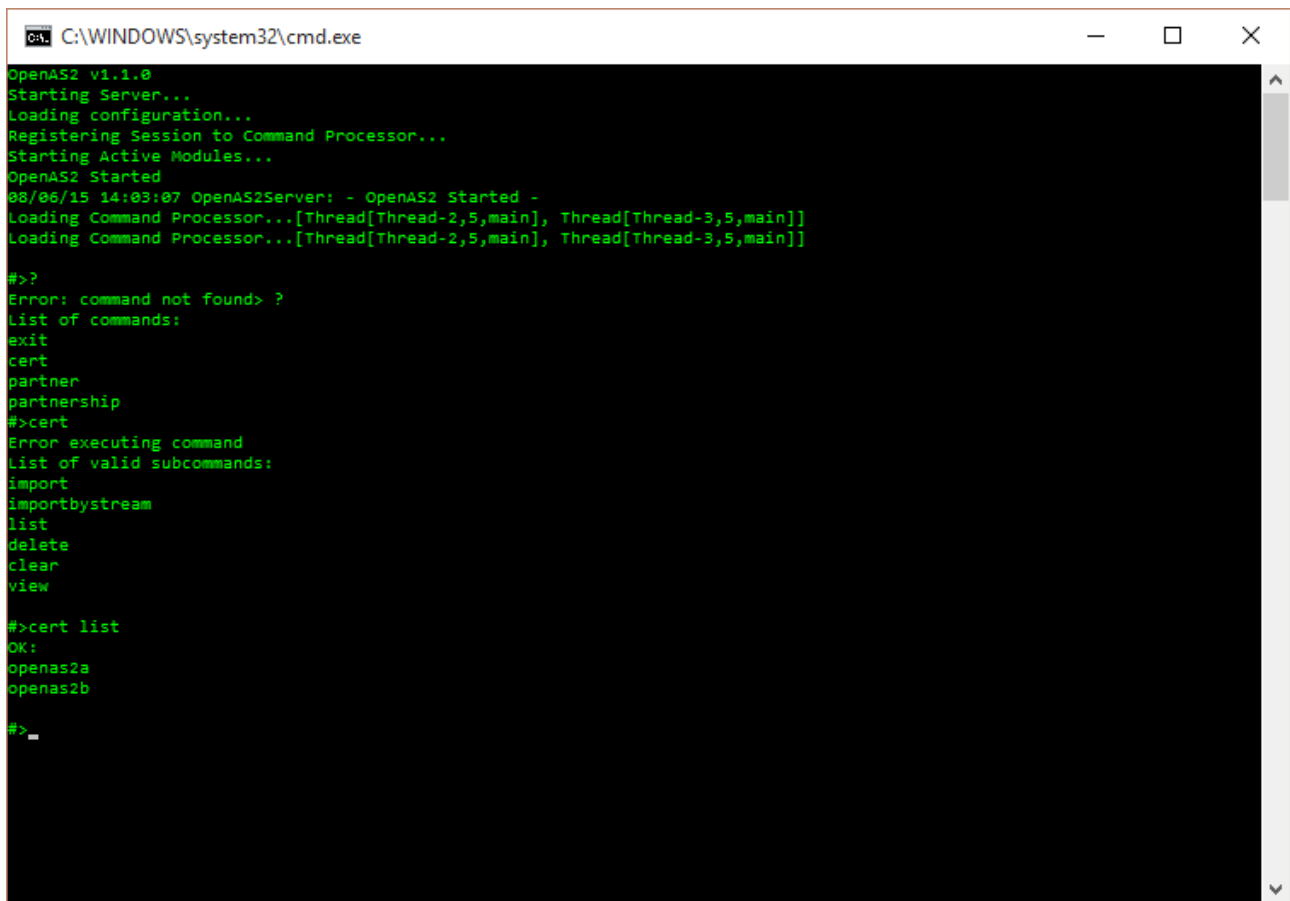
The output in a Unix based system will be identical to that in a Windows based system.

12.2.   Command Entry

After startup of the OpenAS2 application, no command prompt is shown in the command line window initially but you can enter a command or just press <ENTER> to get a visible prompt. Typing ? Will show possible commands. Each command will list sub commands they require if you try to enter them without the appropriate parameters.

 A screenshot showing command entry is shown below.

Command Entry

12.3. Automated Launching As UNIX Daemon

Although the application will launch as a daemon without any change to the default config, it is recommended that the following configuration changes are made to reduce unnecessary processing by modules that are redundant in this mode and filling the system logs with unwanted logging:

1. Remove the console logger – remove the element in the <loggers> element as shown below
   <logger classname="org.openas2.logging.ConsoleLogger"/>

2. Remove the stream command processor in the <commandProcessors> element as shown below
   <commandProcessor classname="org.openas2.cmd.processor.StreamCommandProcessor"/>

12.3.1. INIT.D Service

A sample "openas2.d" is provided in the bin directory of the install package. It provides support for starting and stopping the OpenAS2 application as a daemon using the init.d mechanism. Use the appropriate tool for the NIX operating system you are using to install the script in the /etc/init.d folder and create the soft links to launch the OpenAS2 application when the system starts.

First modify the **openas2.d** file to reflect the path where you have installed OpenAS2 then follow one of the options below.

On Redhat based systems as root:

```
$ cp <srcDir>/bin/openas2.d /etc/init.d/
$ chkconfig --add openas2.d
$ chkconfig --level 2345 openas2.d on
```
On Debian/Ubuntu based systems as root:

```
$ cp <srcDir>/bin/openas2.d /etc/init.d/
$ chmod 750 /etc/init.d/openas2.d
$ update-rc openas2.d defaults
```

12.3.2.   SYSTEMD Service

A sample file **openas2.service** is provided in the bin folder of the install package.

First modify the *openas2.d* file to reflect the path where you have installed OpenAS2 then follow the steps below.

```
$ cp <srcDir>/bin/openas2.service /etc/systemd/system/
$ systemctl daemon-reload
$ systemctl enable openas2.service
```

```
Test that it works using the below commands:
```

```
$ systemctl enable openas2.service
$ systemctl start openas2.service
$ systemctl stop openas2.service
```

12.4.   Windows Service Install

The deployment package contains a version of the Apache Commons Daemon for Windows to support running OpenAS2 as a Windows service. The default name of the service is *OpenAS2Server*.

There is a batch script in *<installDir>/bin* folder named *install_winsvc.bat* to simplify the install process.

NOTE: By default it names the service *OpenAS2Server* and assumes a **64-bit JVM**.

The following steps will install OpenAS2 as a windows service:

1. Edit the *install_winsvc.bat* file and make any changes to the defaults (eg. Change startup to manual, change JVM parameters and set specific JVM instead of default JVM etc). See the Apache Commons Daemon project for more information on parameter settings for the install command.

2. Run the *install_winsvc.bat* file.

 The Apache Commons Daemon files are located in *<installDir>/bin/commons-daemon* folder. The prunmgr.exe has been renamed to the name of the windows service as specified in the *install_winsvc.bat* file (OpenAS2Server.exe). To check the installation or fine tune some of the service settings run OpenAS2Server.exe and adjust as needed.

To uninstall the service use this command from the *installDir>/bin* folder:

commons-daemon\amd64\prunsrv.exe //DS/OpenAS2Server

For 32-bit JVM remove the "**amd64**" from the path in install_winsvc.bat file and the uninstall command.

13.   Testing OpenAS2 Transfers

The default configuration of the OpenAS2 configuration is set up for two partners named "OpenAS2A" and "OpenAS2B". The system will effectively send messages to itself between the 2 configured partners. You can simply start the OpenAS2 server without any changes and then copy a file into the appropriate outbox as defined by the relevant module using the org.openas2.processor.receiver.AS2DirectoryPollingModule classes "**outboxdir**" attribute to send the file to the desired partner.

The default configuration provides for 2 partners OPENAS2A and OPENAS2B and will create outbox folders *<installDir>/data/toOpenAS2A* and *<installDir>/data/toOpenAS2B* for explicitly targeting a partner for any file dropped in one of those folders.

If you wish to run 2 OpenAS2 servers on the same machine then the ports on the 2nd instance of OpenAS2 as configured in the config.xml must be different to those configured on the first instance (see Application Configuration above). If using asynchronous MDN, the URL entry for the attribute "*as2_receipt_option*" in the partnerships.xml file for the 2nd instance must match the values configured in the 1st instances config.xml for hist name and port and vice-versa.

13.1.   Using HTTPS Transport

To test on a local machine using the supplied sample self signed SSL certificate (config/ssl_certs.jks) you should create a localhost DNS entry. The sample certificate was generated for "www.openas2.localhost".

This site will help in how to set up a local DNS: http://www.selfsignedcertificate.com/development_tips.php

The As2ReceiverModule module element should be configured correctly. The key attributes that will work with the supplied sample certificate are already in the sample config file and should just be uncommented:

- `protocol="https"`
- `ssl_keystore="%home%/ssl_certs.jks"`
- `ssl_keystore_password="testas2"`
- `ssl_protocol="TLS"`

The partnership definition for the connection URL will also have to be set to the appropriate host name and use "https" instead of "http":

```
<attribute name="as2_url" value="https://www.openas2.localhost:10080"/>
```

If asynchronous MDN is used then the as2_receipt_option attribute must be configured for SSL as well:

```
<attribute name="as2_receipt_option" value="https://www.openas2.localhost.com:10081"/>
```

The following system property will have to be passed to the application in the java command line:

```
-Dorg.openas2.cert.TrustSelfSignedCN=www.openas2.localhost
```

If you experience problems with SSL, try adding this to the startup command in the script file: -*Djavax.net.debug=SSL*

14. Troubleshooting OpenAS2

This section provides some help in identifying issues with AS2 transfers or configuration and execution of the OpenAS2 application. Experience has shown that not all systems properly implement the AS2 specification or have an interpretation of the specification that is different to the OpenAS2 default implementation. To accommodate these differences, the OpenAS2 application has some configuration parameters to change the default behaviour on a per partnership basis that may help to accommodate the implementation anomalies for various other AS2 systems.

As a first step that may shortcut you to a solution, check the compatibility settings for certain AS2 vendor software in this section: Partner AS2 Compatibility Settings

The sub-sections in this troubleshooting part of the document deal withg specific issues that may be quickly identified via logging (often requiring turning on DEBUG level logging or even more logging using TRACE level to provide some detail to the issue).

Some of the quick and easy things to try that have been known to fix a specific partnership that is failing when others are working are changes to the following partnership attributes:

1. content_transfer_encoding – see here below: Binary Encoding

2. no_chunked_max_size – see here below: Content Length Versus Chunked

3. remove compression by removing the "`compression_type`" attribute from the partnership

4. turn off CMS algorithm protection – see here below" CMS Algorithm Protection

5. manage restricted HTTP headers – see here below: SSL Certificate Exceptions

6. content_type – the system default is set in the config.xml file add this to a more specific value than the system default perhaps specifying the character such . For example: application/edifact; charset=iso-8859-1

7. Add an attribute named "rename_digest_to_old_name " to a value of "true" - this uses the old style of naming the digest algorithm

8. Change the "sign" attribute is lower-case so for example use "sha1" instead of "SHA1" for the value.

14.1.   Canonicalization For MIC Algorithm

Some systems (including OpenAS2 prior to V1.3.7) do not canonicalize the MimeBodyPart as specified in the RFC when content transfer encoding is not "binary" (the OpenAS2 default is "binary" but can be set to other values using the "***content_transfer_encoding***" attribute on the prtnership). This manifests as errors that cause signature authentication failure that may specifically mention a mismatched MIC. To cater for this set the following attribute on the partnership:

`<attribute name="`**`prevent_canonicalization_for_mic`**`" value="`**`true`**`"/>`

14.2. Binary Encoding

If using a content transfer encoding algorithm other than "binary" results in authentication failures, try setting the attribute on the partnership:

`<attribute name="`**`content_transfer_encoding`**`" value="`**`binary`**`"/>`

14.3. HTTP Restricted Headers

Depending on the version of Java you are running, the HTTP class handling sending AS2 messages over HTTP that is part of the core Java distribution will automatically remove any restricted HTTP headers (see here for a discussion: http://stackoverflow.com/questions/11147330/httpurlconnection-wont-let-me-set-via-header).

In terms of OpenAS2 it specifically affects sending the "Content-Transfer-Encoding" header in the HTTP headers (see section 19.4.5 here: https://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html). This should not be a problem for modern AS2 implementations that OpenAS2 communicates with but there are reports that some systems respond with an HTTP 400 error code and reject the message if the "Content-Transfer-Encoding" header is not present in the HTTP headers (it is present in the mime body part headers of the AS2 message).

To solve this, uncomment the line in the startup script file containing this entry

```
-Dsun.net.http.allowRestrictedHeaders=true
To ensure that other partners do not receive the "content-Transfer-
Encoding" header you have 2 options:
   1. Set the following property in the config.xml:
      set_content_transfer_encoding_http_header="false"
      Set this property in the partnership that requires sending the header:
      name="set_content_transfer_encoding_http_header" value="true"
      Ensure that the partnership has the "content_transfer_encoding" header set to
      either "binary" or "8bit"
   2. If the config file property "set_content_transfer_encoding_http_header" is not
      set it defaults to "true" so set this property in the partnerships that do NOT
      require sending the header:
      name="set_content_transfer_encoding_http_header" value="false"
      Ensure that the partnerships have the "content_transfer_encoding" header set to
      either "binary" or "8bit"
```

14.4. CMS Algorithm Protection

Some AS2 systems do not support RFC6211.

The partner system will most likely not provide detailed information that this OID is the issue unless you request detailed logging from the partner but will manifest as authentication failures of some sort. Currently known systems that do not support this are IBM Sterling Integrator.

To disable the OID from being sent, add this attribute to the partnership (from a security point of view to include it wherever possible as it plugs a security issue in CMS signed messages):

<attribute name="remove_cms_algorithm_protection_attrib" value="true"/>

14.5.   Content Length Versus Chunked

OpenAS2 will send all messages using the "chunked" mechanism wherby the actual size of the payload is not pre-determined and sent as a header"Transfer-Encoding=chunked". Some systems cannot handle the chiunked mechanism (it was standardised in HTTP 1.1) and require the "Content-Length" header is used instead. To make OpenAS2 use the "Content-Length" header method, set the following attribute on the partnership that needs it:

<attribute name="no_chunked_max_size" value="104857600"/>

The value for the "no_chunked_max_size" attribute specifies the maximum size of the file it will attempt to send in bytes so with the above value will not be able to send files larger than 100MB If possible avoid using the ***no_chunked_max_size attribute*** for partners because it causes the OpenAS2 server to add an additional step to calculate the length of the payload by converting the payload to a byte array which has a small performance cost for small files but can become significant in high volume transfers of very large files.

14.6.   SSL Certificate Exceptions

Sometimes a partner uses a certificate that has intermediate providers not registered in your Java security keystore. Generally this will be manifested by an exception something like this:

> *javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target*
>
> > *at sun.security.ssl.Alerts.getSSLException(Alerts.java:192)*
> >
> > *at sun.security.ssl.SSLSocketImpl.fatal(SSLSocketImpl.java:1917)*
> >
> > *at sun.security.ssl.Handshaker.fatalSE(Handshaker.java:301)*
> >
> > *at sun.security.ssl.Handshaker.fatalSE(Handshaker.java:295)*
> >
> > *at sun.security.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1369)*

In this case you will need to set up a local trusted certificate provider keystore containing the root or chained (intermediate) certificates that are missing.

Steps:

1. Run the class embedded in the OpenAS2 library jar:

   java -cp <pathToOpenAS2LibFolder>/openas2-server.jar CheckCertificate <host name>[:port] <localKeystoreFile> [passphrase]

   ```
   "<host name>[:port]" should be the same as what you have in the
   partnerships "as2_url" attribute EXCLUDING the "https://"

   "<localKeystoreFile>" is the name you want to give to your local keystore
   (e.g jssechaincerts)
   "[passphrase]" is the password for the keystore - it will default to
   ```

```
"changeit" if you do not provide one`
```

NOTE: If there is no existing keystore you want to add it to then leave out the password otherwise it will throw an error. You can use the keytool utility that comes with java to change the keystore password if you wish but since it does not contain any private keys there is little point in changing the password but if you do then you will have to pass the new password in to the OpenAS2 app using the ***javax.net.ssl.trustStorePassword*** property.

If the class only receives a single certificate as response from the remote host it generally indicates that the root certificate is not trusted and will need installing into a keystore for use by the OpenAS2 application. The output from the class should make it clear it was unable to successfully complete an SSL handshake and it will import the certificate (root or chain as necessary) into the keystore.

2. Add the local cert store to the OpenAS2 startup by adding this to the startup command in the relevant batch file you are using to start OpenAS2:
    ```
    -Djavax.net.ssl.trustStore=<pathToKeystore>/<localKeystoreFile>
    ```

```
NOTE: If you ran the CheckCertificate mechanism a second time but
point it at the keystore it created the first time round it should
successfuly complete the handshake and there will be no messages
to say it is missing a certifcate.
```

For example, run it once like this:

java -cp openas2-server.jar CheckCertificate as2.xyz.com:98765 jssechaincerts

Then run it like this:

java -Djavax.net.ssl.trustStore=jssechaincerts -cp openas2-server.jar CheckCertificate as2.xyz.com:98765 jssechaincerts2

The second instantiation uses the keystore from the first instantiations output and it should not create a new certificate in keystore "jssechaincerts2"

14.7.   Java Versions Prior To 1.7

Java versions below 1.7 are no longer supported.

14.8.   Mime Body Part Logging

Sometimes it may be necessary to see what is actually in the mime body parts received from a partner. OpenAS2 provides a mechanism to enable logging of either received message mime body parts or receieved MDN mime body parts. These are enabled using OpenAS2 startup variables in the startup script in combination with TRACE level logging. Both the DOS and Unix scripts provide these variables but are commented out near the top of the batch file and you can simply uncomment and start the application.

IMPORTANT: this could produce large log files so use sparingly and disable as soon as possible.

The startup variables are:
logRxdMsgMimeBodyParts=true
logRxdMdnMimeBodyParts=true

14.9. TLSv1.2

It appears that although Java7 does support TLSv1.2 it is not enabled by default (refer here: https://blogs.oracle.com/java-platform-group/entry/diagnosing_tls_ssl_and_https)

If you need to use the protocol, add the following to the top of the batch shell script that starts OpenAS2:

Windows: ***set EXTRA_PARMS=%EXTRA_PARMS% -Dhttps.protocols=TLSv1.2***

Linux/Unix/OSX: ***EXTRA_PARMS=$EXTRA_PARMS -Dhttps.protocols=TLSv1.2***

14.10. HTTP Read Timeout Errors

The system is configured to wait a maximum amount of time for a response to any message it sends to your partner and if no response is received it will abort, throw an error and attemot to put the message into the resend queue. The default time out is 60seconds.

If you are transferring very large files to your partner and/or your partner system takes a long time to respond to the sent AS2 message your wil receive read timeout errors.

To fix this, set the "readtimeout" attribute on the AS2SenderModule to a large value if the time taken for the receiving system to respond to the sent file takes longer than 60 seconds. This attribute is set in milliseconds and the default is 60000.

So you would need something like this in the config.xml for a 2 minute timeout:

```
<module classname="org.openas2.processor.sender.AS2SenderModule" retries="3" readtimeout="120000">
</module>
```

14.11. Out Of Memory And File Size Issues

See the section on tuning java (4.3Tuning Java) for solutions to this issue.

14.12. File System Issues

If there are strange issues with files that cannot be found it is often the result of illegal characters in the file name that prevents the creation of a file from dynamic variables. Currently the system as of version 2.3.1 will remove specific characters from any generated file name.

The characters removed by default are: <>:\"|?*

To change the default, set the following property in the config.xml "properties" element escaping XML reserved characters as appropriate: **reservedFilenameCharacters**

```
e.g reservedFilenameCharacters="%lt;&gt;:&quot;|?*"
```

14.13. Header Folding

By default, the OpenAS2 app automatically removes header folding in HTTP headers to comply with the IETF specification for HTTP 1.1 (https://tools.ietf.org/html/draft-ietf-httpbis-p1-messaging-13#section-3.2). It is possible to disable removal of header folding using a property name "**remove_http_header_folding**" set to a value of "**false**" in the <properties> element of the configuration file. An example is shown below:

```
<properties
    log_date_format="yyyy-MM-dd HH:mm:ss.SSS"
    sql_timestamp_format="yyyy-MM-dd HH:mm:ss.SSS"
    as2_message_id_format="&lt;OPENAS2-$date.ddMMyyyyHHmmssZ$-
$rand.UUID$@$msg.sender.as2_id$_$msg.receiver.as2_id$>"
    remove_http_header_folding="false"
/>
```

15. Partner AS2 Compatibility Settings

The below table provides configuration settings for other AS2 systems that are known to work based on user feedback.

PLEASE FEEL FREE TO PROVIDE SETTINGS FOR ANY SYSTEMS THAT REQUIRE A CHANGE FROM THE DEFAULT PROVIDED WITH THE OPENAS2 INSTALL PACKAGE TO COMMUNICATE WITH OTHER AS2 SYSTEMS.

Where the field is left blank, the setting is unknown and the default that comes with OpenAS2 will probably work.

AS2 System	Allow Restricted Headers (startup script property: sun.net.http.allowRestrictedHeaders)	Prevent Canonicalization For MIC (partner attribute: *prevent_cononicalization _for_mic*)	Remove CMS Algorithm Protection (partner attribute: *remove_cms_algorithm_ protection_attrib*)	Other
IBM Sterling	false		true	
IBM Datapower	false		true	
Mendelson	false	true		
Seeburger (older versions)				Add partner attribute: `<attribute name="rename_digest_to_old_name" value="true"/>`
Oracle Integration B2B	false	false	false	
Amazon	false	true	false	

16. Remote Control

By default the OpenAS2 server application will start up a command processor as a socket listener allowing remote connection to the OpenAS2 server to execute commands. The OpenAS2 remote application is part of the application package but is not necessary to use it if you have no remote access requirement and should be disabled in the config.xml file if not using it by removing or commenting out the **<commandProcessor>** element with classname value **org.openas2.cmd.processor.SocketCommandProcessor**

You can set the port that the command processor listens on using the **portId** parameter.

> **<commandProcessor**
> **classname="org.openas2.cmd.processor.SocketCommandProcessor" portId="14321"**
> **userid="userID" password="pWd"/>**

The remote control application will need to connect to the specified port with the specified user ID and password.

The connection uses an anonymous secure socket cipher and may require changing this if your Java implementation does not support the default cipher which is TLS_DH_anon_WITH_AES_256_CBC_SHA for the latest release. This cipher is not available in older Java versions and it may be necessary to switch to SSL_DH_anon_WITH_RC4_128_MD5

To switch cipher you will need to start the OpenAS2 server and the remote command client passing the cipher name as a system property using the -D switch that can be added to the batch script that starts the application. The property must be named "`CmdProcessorSocketCipher`".

e.g java -D`CmdProcessorSocketCipher=SSL_DH_anon_WITH_RC4_128_MD5` …

17. Dynamic Variables

Variables can be used in configuration files for run time replacement of strings. In the case of headers and attributes the reference can be used to change its value. Some variables are specific to certain processor modules and not supported for all situations where dynamic variables can be used. The variables used in the configuration files are as follows:

> *$date.xxx$* - create date strings in a defined format
>> where **xxx** is any valid character formatting string defined in
>> java.text.SimpleDateFormat
>> for example: *$date.YYYY$* gets the four digit year

> *$msg.xxx.yyy$* - accesses various information contained in the AS2 message.
>> Typically used by file modules to configure the name of files used to persist the message payload.
>> The "**xxx**" part can be any one of the following:
>> - *sender* – accesses the "sender" element of the partnership in use for the current message. "**yyy**" can be any attribute name within the "sender" element
>>   eg. $msg.sender.as2_id$ - retrieves the AS2 ID of the sender of the message
>> - *receiver* - accesses the "receiver" element of the partnership in use for the current message. "**yyy**" can be any attribute name within the "receiver"

element
eg. $msg.receiver.as2_id$ - retrieves the AS2 ID of the receiver of the message
- *attributes* – accesses any attributes on the message. The attribute name is used in place of "*yyy*"
  e.g $msg.attributes.filename$ accesses the name of the file contained in the AS2 message
- *headers* - accesses any headers on the message. The header name is used in place of "*yyy*"
  e.g $msg.headers.content-type$ accesses the content type header for the AS2 message
- *content-disposition* - used to access any content-disposition attribute in the received message content disposition where the attribute identifier is used in place of "*yyy*"
  e.g $msg.content-disposition.fielname$ accesses the name of the file received from the partner

$mdn.zzz$ for message mdn parameters, used by EmailLogger and MDNFileModule
where **zzz** can be any of the following values:
- **msg** – requires "zzz" to be in the form "xxx.yyy" and can access data points as defined for $msg.xxx.yyy$ format dynamic variables above
- **sender** – gets the as2_id of the sender
- **receiver** – gets the as2_id of the receiver
- text - gets the text portion of the MDN
- **attributes** – requires "zzz" to be in the form "xxx.yyy" and can access data points as defined for $msg.xxx.yyy$ format dynamic variables above
- **headers** – requires "zzz" to be in the form "xxx.yyy" and can access data points as defined for $msg.xxx.yyy$ format dynamic variables above

eg.: *$mdn.text$* gets the text portion of the MDN

$rand.zzz$ can be used on most strings to produce random strings.
Produces a random UUID or a 0 padded random number of a defined number of digits where zzz can be any string of any number of characters
- if "zzz" is "UUID" or "uuid" (e.g $rand.UUID$) then it produces a random UUID
- for any other string of characters other than UUID, the number of characters in the string determines the number of digits in the random number that is generated and will be zero padded
  e.g $rand.1234$ - creates a 4 digit random number between 0000 and 9999
  $rand.ax1fg4c5$ - creates an 8 digit random number between 00000000 and 99999999

$exception.xxx$ -used by EmailLogger
where **xxx** can be any of the following
- name - retrieves name of the exception
- message – retrieves the exception message
- trace – retrieves the trace log for the exception

eg.: *$exception.trace$* gets the trace log of the exception

$component.xxx$ -used in module configuration
where **xxx** can be any of the attribute names specified above the attribute in the same

module element. Can be used to simplify setting hard coded strings into a concatenated string used by the server module. See the DB tracking module definition for an example

18. Appendix: config.xml file structure

- Node: **properties**
  > Various properties can be defined here and accessible globally within the OpenAS2 application. See the standard config.xml for existing properties that are supported. Developers who create custom modules can pass config to those modules via the properties and access them using Properties.getProperty(<propertyName>)
- Node: **openas2**
  - Node: **certificates**

    Attributes

    classname
    > describes the Java class to process the certificate file.
    > for example: *org.openas2.cert.PKCS12CertificateFactory*
    filename
    > defines the file name containing the certificates
    > for example: *%home%/certs.p12*
    password
    > opens the file using this password
    > for example: *test*
    > > *NOTE: this can be overriden using a java system property when starting the application: -Dorg.openas2.cert.Password=<somePassword>*
    interval
    > describes how often the file should be check up for updates. Specified in seconds.
    > for example: *300*
  - Node: **partnerships**
    Describes the OpenAS2 classes to handle the trading partner identifications.

    Attributes

    classname
    > describes the Java class to process the partnerships file
    > for example: *org.openas2.partner.XMLPartnershipFactory*
    defines the file name containing the partnerships definitions
    > describes
    > for example: *%home%/partnerships.xml*
  - Node: **loggers**
    Describes the OpenAS2 logging classes to use. **You must include -Dorg.apache.commons.logging.Log=org.openas2.logging.Log in your startup call or as a property in the commons-logging.properties file. See http://commons.apache.org/logging/guide.html#commons-logging-api.jar for**

more information.

Do not use this node when using other logging packages (e.g. log4j) with the OpenAS2 package.

- Node: **logger** (for E-mail logging)
  Optional, if not specified no E-mail logging is performed.

    Attributes

    classname
    > describes the Java class to process E-mail logging
    > for example: *org.openas2.logging.EmailLogger*

    show (Optional)
    > describes what level of logging to handle
    >> Possible values
    > - all = all exceptions (terminated or not) and info
    > - terminated = all terminated exceptions **Default value**
    > - exceptions = all non-terminated exceptions

    > for example: *terminated*

    from
    > defines the source email address
    > for example: logger@*openas2.org*

    from_display
    > defines the displayed text of the source email address
    > for example: O*penas2*

    to
    > defines the recipient email address
    > for example: *your@e-mailaddress.com*

    smtpserver
    > describes the SMTP server to process outgoing e-mail
    > for example: *mySillyMailerDot.com*

    smtpport
    > defines the SMTP server port to connect to
    > for example: 587

    smtpauth
    > defines whether authentication is required for the SMTP server
    > (Possible values: true, false)
    > for example: true

    smtpuser
    > defines user name if authentication is required for the SMTP
    > server

    smtppwd
    > defines user password if authentication is required for the SMTP
    > server

    subject
    > describes the e-mail to the receiving party
    > for example: *$exception.name$: $exception.message$ (only
    > relevant for specific exceptions type)*

bodytemplate

> defines the file that contains the body of the message
> for example: *%home%/emailtemplate.txt*

- Node: **logger** (for file logging)

  Optional, if not specified no file logging is performed.

  Attributes

  classname
  > describes the Java class to log messages
  > for example: *org.openas2.logging.FileLogger*

  filename
  > defines the name of the output log file.
  > for example: *%home%/log-$date.MMddyyyy$.txt*

  show (Optional)
  > describes what level of logging to handle
  > > Possible values
  > - all = all exceptions (terminated or not) and info **Default value**
  > - terminated = all terminated exceptions
  > - exceptions = all non-terminated exceptions
  > - info = all info log entries
  >
  > for example: *terminated*

- Node: **logger** (for Console logging, writes to System.out)

  Optional, if not specified no console logging is performed.

  Attributes

  classname
  > describes the Java class to log messages
  > for example: *org.openas2.logging.ConsoleLogger*

  show (Optional)
  > describes what level of logging to handle
  > > Possible values
  > - all = all exceptions (terminated or not) and info **Default value**
  > - terminated = all terminated exceptions
  > - exceptions = all non-terminated exceptions
  > - info = all info log entries
  >
  > for example: *info*

- Node: **commands**

  Describes the OpenAS2 command classes to use

  Attributes

  classname
  > describes the Java class to process the command file
  > for more information see Command File
  > for example: *org.openas2.app.XMLCommandRegistry*

filename
> defines the name of the file command all possible commands
> for example: *%home%/commands.xml*

- Node: **processor**

Describes the OpenAS2 class to handle the message processors.

Attributes

classname
> describes the default Java class to handle outgoing message
> for example: *org.openas2.processor.DefaultProcessor*

- Node: **module**

Module that sends out AS2 messages.

Attributes

classname
> describes the Java class to send outgoing Messages
> for example: *org.openas2.processor.sender.AS2SenderModule*

retry
> defines the number of attempts for sending a message,default is -1 aka infinite.
> for example *retries="3"* will stop sending the message after 3 failures.

connecttimeout
> defines the millisecond count before a connection times out. default value is 30000 or 30 seconds.
> for example *connecttimeout="60000"* will time out after 60 seconds.

readtimeout
> defines the millisecond count before a read times out. default value is 30000 or 30 seconds.
> for example *readtimeout="60000"* will time out after 60 seconds.

- Node: **module**

Module that sends out AS2 MDNs.

Attributes

classname
> describes the Java class to send synchronous or asynchronous MDN
> for example: *org.openas2.processor.sender.MDNSenderModule*

retry
> defines the number of attempts for sending a message, default value is -1 (infinite.)
> for example *retries="3"* will stop sending the message after 3 failures.

connecttimeout
> defines the millisecond count before a connection times out. default value is 30000 or 30 seconds.

for example *connecttimeout="60000"* will time out after 60 seconds.

readtimeout
> defines the millisecond count before a read times out. default value is 30000 or 30 seconds.
> for example *readtimeout="60000"* will time out after 60 seconds.

- Node: **module**

  The following will describe a module to process outgoing message placed in a generic directory. The module determines the receiver and send from the file name placed in the directory (see [format](#) attribute). This module will look for files in specified directory and file names to send to the default message processor.

  Attributes

  classname
  > describes the Java class to process files to be sent to the AS2SenderModule for its delivery process.
  > for example:
  > *org.openas2.processor.receiver.AS2DirectoryPollingModule*

  outboxdir
  > defines the directory where files are to be found.
  > for example: *%home%/toAny*

  fileextensionfilter
  > defines the extension of the file name if file filtering is required. The system will prefix the text entered in this attribute with a period and only files matching that extension will be picked up by the polling module
  > for example: txt - *this will only find files like test.txt but not mytxt*

  errordir
  > defines directory where files containing errors are redirected to.
  > for example: *%home%/toAny/error*

  interval
  > describes how often the directory is to be checked for work. Specified in seconds. Default is 30 seconds.
  > for example: *5*

  delimiters
  > defines the characters used to parse the incoming file name. Characters are separate the tokens: sender, receiver and file id.
  > for example: -.

  format
  > describes the file name by the tokens sender, receiver and file id. May be in any order. Sender id and receiver id are as defined in the partnership.xml file.
  > for example: *sender.as2_id, receiver.as2_id, attributes.fileid* or *attributes.mimetype, attributes.mimesubtype, sender.name, receiver.name*

  mimetype
  > describes the outgoing message mime message type.

for example: *application/EDI-X12*

- Node: **module**

  Attributes

  classname
  : describes the Java class to process files for a particular trading partner that are sent to the AS2SenderModule for its delivery process.
    for example: *org.openas2.processor.receiver.AS2DirectoryPollingModule*
  outboxdir
  : defines the directory where outgoing message are defined.
    for example: *%home%/toOpenAS2A/*
  errordir
  : defines the directory where erroneous messages are left.
    for example: *%home%/toOpenAS2A/error*
  interval
  : describes how often the incoming directory is searched. Defined in seconds, default is 30 seconds.
    for example: *5*
  defaults
  : describes the AS2 sender and receiver ids as defined in the partnership.xml file.
    for example: *defaults="sender.as2_id=OpenAS2A_OID, receiver.as2_id=OpenAS2B_OID"*
  protocol
  : describes the AS2 protocol, which is AS2.
    for example: *as2*
  mimetype
  : describes the outgoing message mime message type.
    for example: *application/EDI-X12*

- Node: **module**

  Attributes

  classname
  : describes the Java class to process incoming MDNs
    for example: *org.openas2.processor.storage.MDNFileModule*
  filename
  : describes
    for example:
    %home%/mdn/$date.yyyy$/$date.MM$/$mdn.msg.sender.as2_id $-$mdn.msg.receiver.as2_id$-$mdn.msg.headers.message-id$
  protocol
  : describes
    for example: *as2*
  tempdir
  : describes
    for example: *%home%/temp*

- Node: **module** Defines the module to handle messages.

Attributes

classname
> describes the Java class to process and store incoming messages
> for example:
> *org.openas2.processor.storage.MessageFileModule*

filename
> describes the location and formatted filename of the stored
> MDNs.
> for example: *%home%/inbox/$msg.sender.as2_id$-*
> *$msg.receiver.as2_id$-$msg.headers.message-id$*

protocol
> describes the AS2 protocol
> for example: *as2*

tempdir (Optional)
> defines temporary directory used to store MDNs during message
> processing.
> for example: *%home%/temp*

- Node: **module**

Attributes

classname
> describes the Java class to process handle incoming transfers
> for example:
> *org.openas2.processor.receiver.AS2ReceiverModule*

address
> an optional attribute that defines the host name or ip address to
> bind the listener to on the server. It defaults to localhost
> (127.0.0.,1)
> for example: 192.168.1.3

port
> defines the port the server listens on.
> for example: *10080*

errordir
> defines directory where invalid incoming messages are stored.
> for example: *%home%/inbox/error*

errorformat
> defines the format of filenames for invalid incoming messages.
> for example: *sender.as2_id, receiver.as2_id, headers.message-id*

protocol
> optional and defaults to "http" if not present
> set to *"https"* for SSL transport protocol

ssl_protocol
> optional and defaults to "TLS" if not present
> set to preferred SSL transport protocol
> for example: SSLv3

ssl_keystore
> The name of the file including path containing SSL certificate
> only required for "protocol" attribute set to "https"
> for example: *%home%/ssl_certs.jks*

ssl_keystore_password

The password to open the SSL keystore
only required for "protocol" attribute set to "https"
for example: *mySecretPassword*
NOTE: this can be overriden using a java system property
when starting the application: -
Dorg.openas2.sslPassword=<somePassword>

- Node: **module**

    Attributes

    classname
    describes the Java class to send asynchronous MDN response
    for example:
    org.openas2.processor.receiver.AS2MDNReceiverModule
    address
    an optional attribute that defines the host name or ip address to
    bind the listener to on the server. It defaults to localhost
    (127.0.0.,1)
    for example: 192.168.1.3
    port
    defines the port the server listens on.
    for example: *10080*
    protocol
    optional and defaults to "http" if not present
    set to "*https*" for SSL transport protocol
    ssl_protocol
    optional and defaults to "TLS" if not present
    set to preferred SSL transport protocol
    for example: SSLv3
    ssl_keystore
    The name of the file including path containing SSL certificate
    only required for "protocol" attribute set to "https"
    for example: *%home%/ssl_certs.jks*
    ssl_keystore_password
    The password to open the SSL keystore
    only required for "protocol" attribute set to "https"
    for example: *mySecretPassword*
    NOTE: this can be overriden using a java system property
    when starting the application: -
    Dorg.openas2.sslPassword=<somePassword>

- Node: **module**

    Attributes

    classname
    describes the Java class to rehandle messages
    for example:
    org.openas2.processor.resender.DirectoryResenderModule
    resenddir
    defines the directory to find message to resend

for example: *%home%/resend*

errordir

defines the director to store resend messages that are in error.

for example: *%home%/resend/error*

resenddelay

defines the wait time between resends. Defined in seconds.

Default is 60.

for example: *600*

19. *Appendix: partnership.xml file structure*

This file describes your company and your trading partners. This file requires modification to work with your application

- Node: **partnerships**

  The root node.

  - Node: **partner**

    partner definition

    Attributes

    name

    partner name as defined in OpenAS2 configuration file.
    OpenAS2A

    as2_id

    partner name as defined in partnership node
    OpenAS2A

    x509_alias

    Alias as defined in certificate file
    o*penas2a*

    email

    E-mail address of partner
    as2a@MySillyMailerServer.com

  - Node: **partnership**

    defines partner relationships between sender and receiver

    - Node: **partnership**

      Attributes

      name

      Unique name of partnership relation. See filename parsing above.
      OpenAS2A-OpenAS2B

    - Node: **sender**

      Attributes

      name

      Unique name of Sender
      OpenAS2A

    - Node: **receiver**

      Attributes

name
> Unique name of receiver
> *OpenAS2B*

*The following is a list of nodes that use the node name of **attribute**. The subnodes of **attribute** use a name/value node naming pair structure.*

- Node: **attribute**
  name is **protocol** defines the protocol to use with this partner.
  value is **as2**
  name="protocol" value="as2"
- Node: **attribute**
  name is **subject** defines text used in E-mail subject line
  value – can use references to message parameters as in example. If this attribute is not present in the partnership when sending an MDN, the subject will use the text in the received message much like responding to an email does so putting a subject attribute in a partnership where it is the receiver effectively overrides the received subject
  name="subject" value="File $attributes.filename$ sent from $sender.name$ to $receiver.name$"
- Node: **attribute**
  name is **as2_url** defines partners AS2 server's URL
  value
  name="as2_url" value="http://www.MyPartnerAS2Machine.com:10080"/>
- Node: **attribute**
  name is **as2_mdn_to** when set this specifies that an MDN response is required and defines value of the ""`Disposition-Notification-To"` header in the AS2 message sent to the partner. It is normally an email address but can be any string that is meaningful
  value
  name="as2_mdn_to" value="datamanager@mypartner.com"
- Node: **attribute**
  name is **as2_receipt_option** defines asynchronous MDN server's URL
  value
  name="as2_receipt_option" value="http://www.MyAS2Machine.com:10081"
- Node: **attribute**
  name is **as2_mdn_options** defines MDN option values for E-mail header
  value
  name="as2_mdn_options" value="signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, sha1"
- Node: **attribute**
  name is **encrypt** defines encrypting algorithm name for E-mail header
  value
  name="encrypt" value="3des"
- *Node: **attribute (optional)***

> *name* is **content_transfer_encoding** *defines what the header field should display*
> *value* *8bit (default), binary, ...*
> *name="content_transfer_encoding" value="binary"*

- *Node:* **attribute (optional)**
  name is **compression_type** *if defined it determines what the type of compression to use. Leave this attribute out if no compression is required*
  value *ZLIB (default) – no other supported options*
  name="compression_type" value="ZLIB"

- *Node:* **attribute (optional)**
  name is **compression_mode** *if defined it determines when compression occurs. If this attribute is not specified then compression occurs before signing.*
  value *– "compress-after-signing"*
  name="compression_mode" value="compress-after-signing"

20. Appendix: command.xml file structure

List of commands available to the OpenAS2 server Application.

- Node: **commands** the root node
  - Node: **multicommand**

    attribute

    name
      value "cert|part", certificate commands or partnership commands
    description
      value is some useful text
  - Node: **command**

      attribute

      classname
        value is a OpenAS2 classname that will process a command

21. Appendix: Updating database structure

The table structure for message tracking is stored in an XML file structure that uses the Apache DDLUtils project structure.

OpenAS2 includes a jar file that uses the Apache DDLUtils project to generate DDL statements to create or update the database to match the XML definition of the database and can also be used to automatically update the target database. Configuration for the database update functionality comprises 3 files located in the <installDir>/resources/DB folder:

1. openas2-schema.xml – the XML definition of the table used to capture AS2 message states

2. jdbc.properties.h2 – defines the connection parameters for the databases

3. build.xml – the Ant build file that invokes

Run this command in the dorectory containing the Ant build file to generate a DDL file containing statements to update the DB to match the XML definition:

    ant -Djdbc.properties.file=jdbc.properties.h2

To directly update the database without generating a DDL file use this command:

    ant -Djdbc.properties.file=jdbc.properties.h2 writeSchemaToDb

22. Appendix: Creating database DDL for external databases

The deployment package for OpenAS2 contains resources to generate DDL statements for the database table used to log AS2 message state. The tool requires Ant to be installed (https://ant.apache.org/).

It supports the following database platforms:

- axion
- cloudscape
- db2
- derby
- firebird
- h2
- hsqldb
- interbase
- maxdb
- mckoi
- mssql
- mysql
- oracle
- postgresql
- sapdb
- sybase

To use a different database system than H2, follow these steps:

1. Create a database with the desired name , user and password in the target database system

2. Change the property named "platform" in the build.xml file to the required database platform

3. Set up a jdbc.properties file with the appropriate settings using the jdbc.properties.h2 file as a template.

4. Create the database table. To generate DDL statements to an SQL file that can then be used to apply to the database use this command:

ant -Djdbc.properties.file=jdbc.properties

To directly update the database without generating a DDL file use this command:

ant -Djdbc.properties.file=jdbc.properties writeSchemaToDb

5. Change the appropriate parameters in the config.xml file for the database tracking module.

23. Appendix: Upgrading

Each release contains a RELEASE-NOTES.txt file. This file contains a section specifically about upgrading to use new functionality if there was any config related new functionality in that release. There will be no upgrade notes for the particular release if it was just a bugfix or minor enhancement where there is no config to be done. You should add any configuration related elements to the appropriate XML file(s) if you wish to use new functionality that requires configuration settings.

The basic upgrade path is as follows:

1. partnerships.xml : Review the release notes for any new attributes that are supported in the partnerships.xml and add if there is a perceived advantage/enhancement (in general the partnerships.xml should not need any modification if it already works as all enhancements to the partnership configuration have ensured they do not change the default behavior from prior versions)

2. config.xml: Review and merge any new configuration into your existing config.xml. Generally it should be fairly obvious where there are "missing" items in your existing file compared to the version you are upgrading to. For performance purposes, make sure you do not add unwanted modules and perhaps a good idea to review any modules you have not used but have enabled such as the remote command processor, socket logging etc.

3. Review the startup script (*start_openas2.sh* or *start_openas2.bat*) and merge any enhancements you may have made in your deployed version into the new version and replace the old version if necessary. Specifically ensure you set the classpath appropriately to cater for upgraded or new libraries.

4. Delete all files in your "lib" folder and copy all the files from the release package "lib" folder into your deployed folder.

NOTE: The alternative route is to unzip the new release into a new folder on your server and follow steps 1, 2 and 3 above except merging the changes you made originally into the new deployment. This route may be the quicker route if you have not customized the configuration files

extensively and will ensure you do not miss newly added configuration.

24. Appendix: Clustering and Load Balancing

There are 4 key modules that must be considered in designing for a clustered/load balanced setup:
1. Directory polling module (or equivalent module responsible for passing a file into the sender module)
2. Resender module - handles the case where the remote partner either is not available when a connection is attempted or the exchange fails at some point in the AS2 message exchange process.
3. Asynchronous MDN receiver module - requires access to the pending information about the message that was sent causing this MDN to be received (the pending information is stored by the AS2 sender module on the file system)

There is a different complexity involved depending on whether you use synchronous or asynchronous MDN for any sent messages in a load balanced scenario. Synchronous is much simpler since in the case of an asynchronous MDN there is a separate network connection made back to your OpenAS2 service to the one made to send the message and therefore there is no guarantee which node will receive the MDN unless you have dedicated host names for each node in a cluster just to ensure the MDN is returned to the node that sent the message (the sender includes the MDN response host name as part of the sent AS2 message to the recipient that will send the MDN back). Since the decision to request a synchronous or asynchronous response rests with the sender and the recipient is told which mechanism to use in the headers of the sent message it is possible to simply ensure that sent messages only use the synchronous mechanism. However, you might find that some partners require asynchronous MDN so the design relying on using only synchronous MDN may not cater for all your partners you send messages to.

Currently, OpenAS2 stores the sent message on the file system pending an MDN response so that the resend mechanism works if there is an MDN response indicating a failure or no MDN response is received. If the sent message pending information is stored on a shared file system then the node that receives the MDN can either mark the message as processed and delete the stored pending information file or reconstruct the original message if a resend is needed passing the message to the resender queue and it should work ok.

For the outbound side, you will need to have a mechanism to ensure only one node picks up the message to be sent. Since the current OpenAS2 code base only provides for a directory polling module, some of your possible options are:
1. have a dedicated directory per OpenAS2 instance and a controller deciding which instance to send the file to (means the controller itself has some level of complexity in its design and must have some way of knowing if the file was ever actually sent - could use the state logging in the database for this)
2. have a shared network file system and enhance the directory poller to provide a mechanism to ensure only one of the nodes picks up the file
3. Create a new web service module that provides a means for a network connection for the file to be passed in - the web service would be load balanced to ensure high availability. The asynchronous MDN response means there is no guarantee which node will receive the MDN and therefore any

web service would need some means of verifying the MDN was received.

There is a similar problem for the resender module in as much as it works somewhat like the directory polling module where it scans a directory looking for messages to resend. Some possible solutions are:
1. Have a resender module selection mechanism that allows only one node in a cluster to be the active resender and a mechanism to ensure a new node will take over in the case of the active node failing
2. Implement a resender queue module that provides a mechanism for resender modules on all active nodes to request any pending message to be resent
3. Offload the resend decision to the web service that passes in the original file so there is no resender module in OpenAS2. Instead the failure to successfully send is fed back to the originating service

For the inbound side there should be no real significant hurdles since the whole process is synchronous even for asynchronous MDN. As long as you use a shared network file system for storing received messages or each node writes the received file to a file system that will be picked up by the appropriate consumer of that file.

25. Appendix: Maven Artifacts

If you are using Maven for your project and building custom modules to enhance the OpenAS2 application or you are including OpenAS2 into a larger project and using it as a library then you can use the maven dependency in your om.xml as follows:

```xml
<dependency>
        <groupId>net.sf.openas2</groupId>
        <artifactId>openas2-server</artifactId>
        <version>2.4.0</version>
</dependency>
```