

OpenAS2 Server Application

Table of Contents

1. Introduction.....	1
2. Glossary.....	1
3. Installing OpenAS2.....	1
4. Configuration.....	2
4.1. Application Configuration.....	3
4.2. Partner Configuration.....	4
4.2.1. Partner Definition.....	4
4.2.2. Partnership Definition.....	4
4.3. Certificate Configuration.....	5
4.4. Logging Configuration.....	5
5. Running OpenAS2.....	5
6. Testing OpenAS2 Transfers.....	8
7. Remote Control.....	8
8. Dynamic Variables.....	8
9. Appendix: config.xml file structure.....	9
10. Appendix: partnership.xml file structure.....	15
11. Appendix: command.xml file structure.....	17

1. Introduction

The OpenAS2 application enables you to transmit and receive AS2 messages with EDI-X12, EDIFACT, XML, or binary payloads between trading partners. This document describes how to install, configure, and use OpenAS2. In this document a partner can be either your own company or a company you will be exchanging data with using AS2.

The examples 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 it will do the conversion if necessary.

2. Glossary

EDI – Electronic Data Interchange

MDN - Message Disposition Notification

3. Installing OpenAS2

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](#).

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 call `<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. 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. Configuration

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
2. partnerships.xml – configures the partners
3. as2_certs.p12 – stores the PKCS12 certificates for all partners
4. commands.xml – stores the commands that the application will support. This file should not be modified

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.

4.1. 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
- define the location of the partnerships configuration file
- specify the listening ports

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.

4.2. 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.

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 not necessary to have 2 elements if data transfer is unidirectional.

4.2.1. 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 alias of the certificates for this partner in the keystore. The encryption and decryption of messages requires the partner's public or private key as appropriate

4.2.2. 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>` - a fully qualified URI that provides the connection string to the remote partner for sending asynchronous AS2 MDN's. 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.

The default content transfer encoding uses "8bit" if not explicitly overwritten in the configuration. Many AS2 implementations use "binary" and this 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.

4.3. Certificate Configuration

The certificate store used by default is a PKCS12 key store and stores all X.509 certificates. The key store must contain the private key of your own X.509 certificate and the public key for each of your trading partners X.509 certificates. The certificates must be stored with the matching alias as specified in the partner definition of each partner in the partnership.xml file.

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

An excellent open source visual keystore manager that will run on any OS can be found here: <http://portecle.sourceforge.net/>

4.4. Logging 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 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"

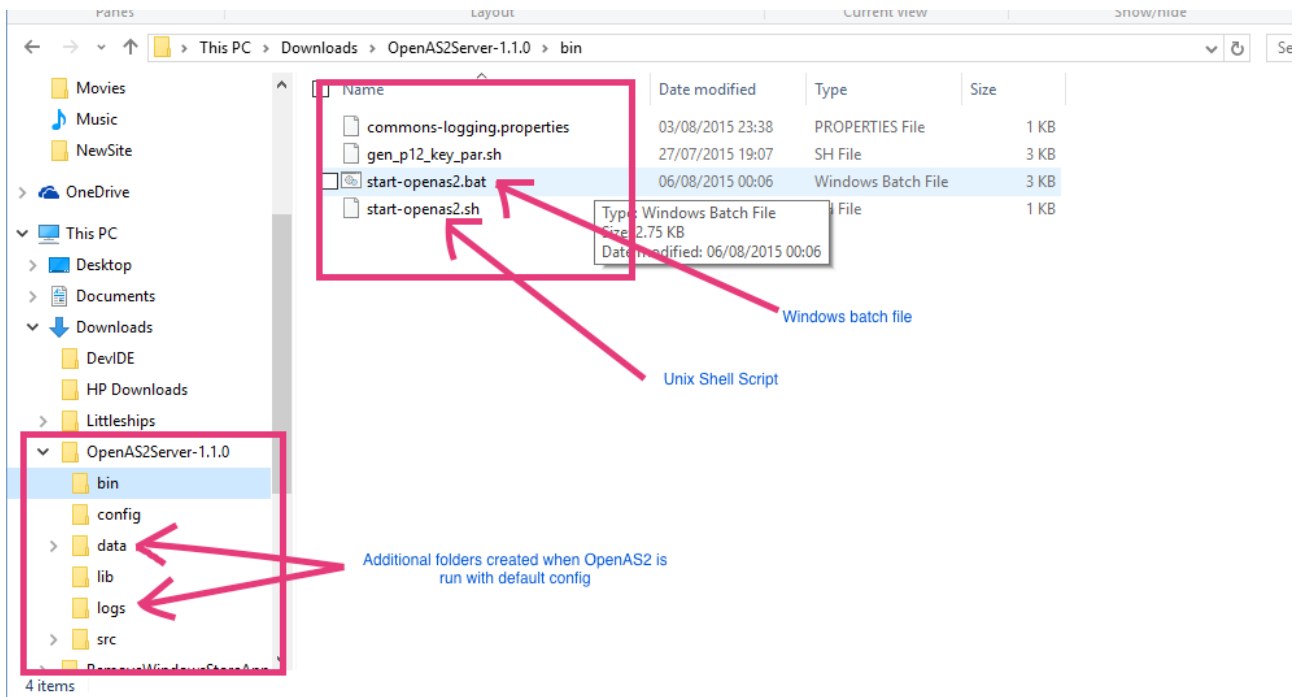
The logging levels are turned off by specifying the level you want on and all other levels higher than that level will also be turned on.

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.

5. Running 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:

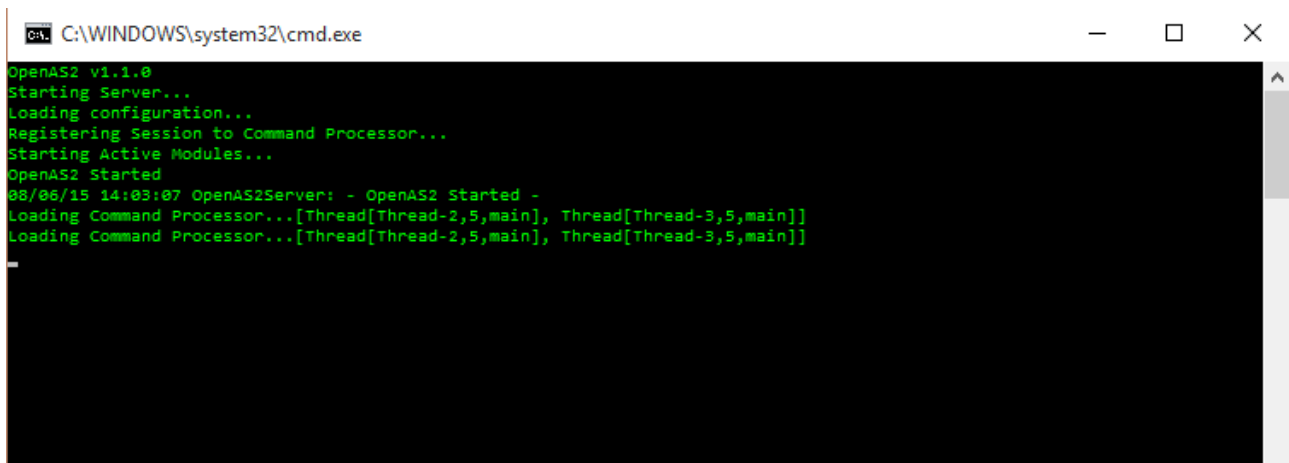
1. start-openas2.sh – for UNIX based systems
2. start-openas2.bat – for Microsoft Windows based system

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:

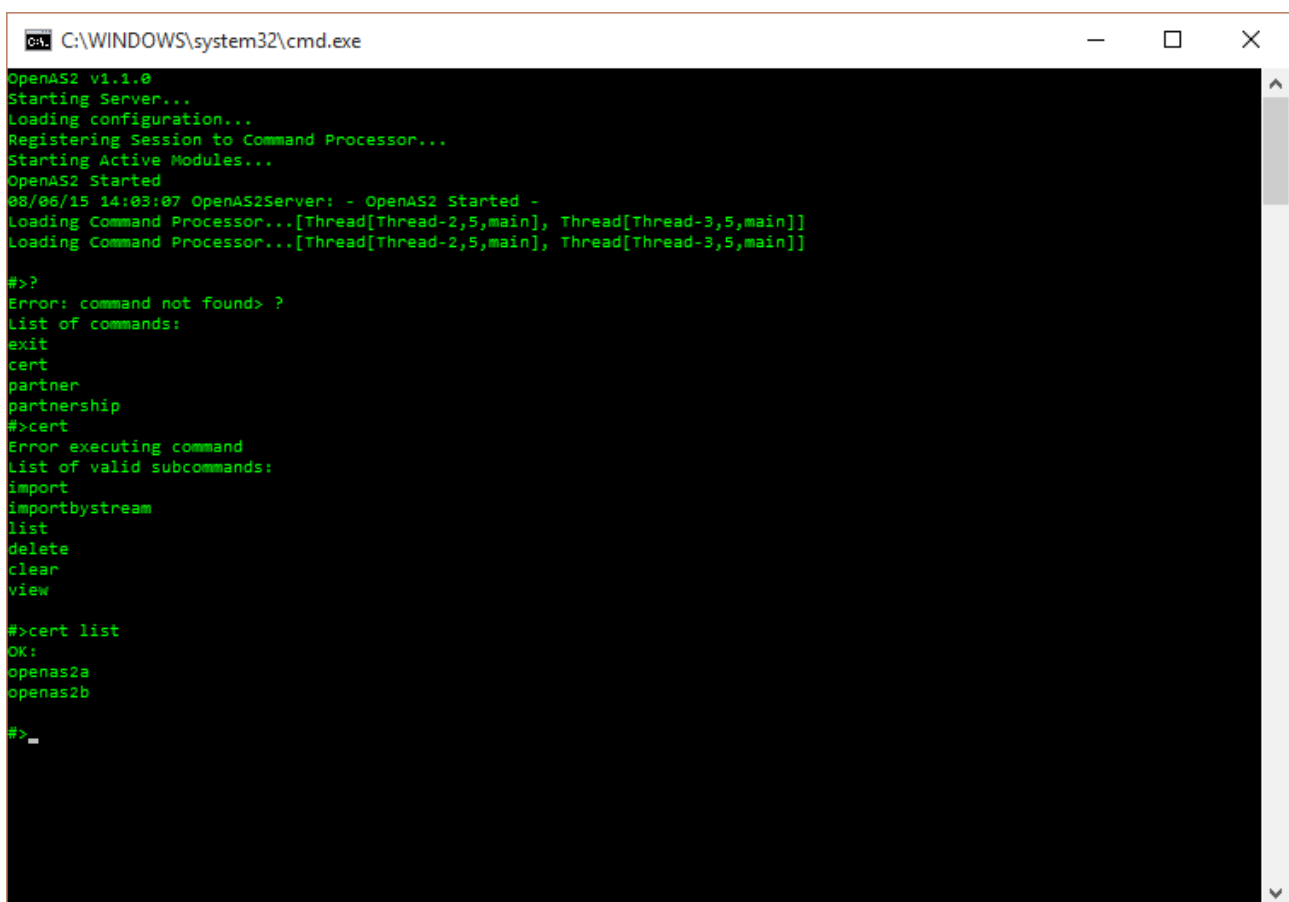
- logs – contains the norml program logging
- data – contains all the transferred files and any AS2 specific headers associated with AS2 transfers. This folder will have a number of sub folders for outbound and inbound files for different partners

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



```
C:\WINDOWS\system32\cmd.exe
OpenAS2 v1.1.0
Starting Server...
Loading configuration...
Registering Session to Command Processor...
Starting Active Modules...
OpenAS2 Started
08/06/15 14:03:07 OpenAS2Server: - OpenAS2 Started -
Loading Command Processor...[Thread[Thread-2,5,main], Thread[Thread-3,5,main]]
Loading Command Processor...[Thread[Thread-2,5,main], Thread[Thread-3,5,main]]
```

No command prompt is shown 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 sample is shown below.



```
C:\WINDOWS\system32\cmd.exe
OpenAS2 v1.1.0
Starting Server...
Loading configuration...
Registering Session to Command Processor...
Starting Active Modules...
OpenAS2 Started
08/06/15 14:03:07 OpenAS2Server: - OpenAS2 Started -
Loading Command Processor...[Thread[Thread-2,5,main], Thread[Thread-3,5,main]]
Loading Command Processor...[Thread[Thread-2,5,main], Thread[Thread-3,5,main]]

#>?
Error: command not found> ?
List of commands:
exit
cert
partner
partnership
#>cert
Error executing command
List of valid subcommands:
import
importbystream
list
delete
clear
view

#>cert list
OK:
openas2a
openas2b

#>_
```

There will be some logging as the application starts and it will then provide a command prompt (you may need to press the ENTER key to see it due to logging events).

You can enter commands after startup by typing in the console window.

6. 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). The entry in the `partnerships.xml` file for the 2nd instance for the `as2_mdn_to` and `as2_mdn_from` must match the values configured in the 1st instances `config.xml` for

7. 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.

8. Dynamic Variables

Variables can be used in configuration files for real time replacement of strings. Some variables are specific to certain processor modules. The variables used in the configuration files are as follows:

\$date.xxx\$::: for date parameters
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\$, from the incoming message, used by MessageFileModule
where xxx can be any of the following values to get

- sender
- receiver
- attributes
- headers

for example: *\$msg.sender\$* gets the sender id on the message

\$mdn.xxx\$ for message mdn parameters, used by EmailLogger and MDNFileModule
where xxx can be any of the following values to get

- msg
- sender
- receiver
- text
- attributes
- headers

for example: *\$mdn.text\$* gets the text portion of the MDN

\$exception.xxx\$, used by EmailLogger

where xxx can be any of the following values to get

- name
- message
- trace
- terminated

for example: *\$exception.trace\$* gets the trace log of the exception

9. Appendix: config.xml file structure

- 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*

interval

describes how often the file should be checked 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
- info = all info log entries

for example: *terminated*

from

describes

for example: *openas2*

to

describes

for example: *your e-mail address*

smtpserver

describes the SMTP server to process outgoing e-mail

for example: *mySillyMailerDot.com*

subject

describes the e-mail to the receiving party

for example: *\$exception.name\$: \$exception.message\$*

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 asynchronously.

Attributes

classname

describes the Java class to send asynch MDN
for example:
org.openas2.processor.sender.AsynchMDNSenderModule

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*

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 modules

for example:

org.openas2.processor.receiver.AS2ReceiverModule

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*

- 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*

10. *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
openas2a

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

- name="subject" value="From OpenAS2A to OpenAS2B"*
- 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** defines MDN server's URL
 - value**
name="as2_url" value="http://www.MyPartnerAS2Machine.com:10080"
name="as2_mdn_to" value="http://www.MyAS2Machine.com:10081"
- 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"*

11. 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