Hermes Documentation

Release 2.2

CECID

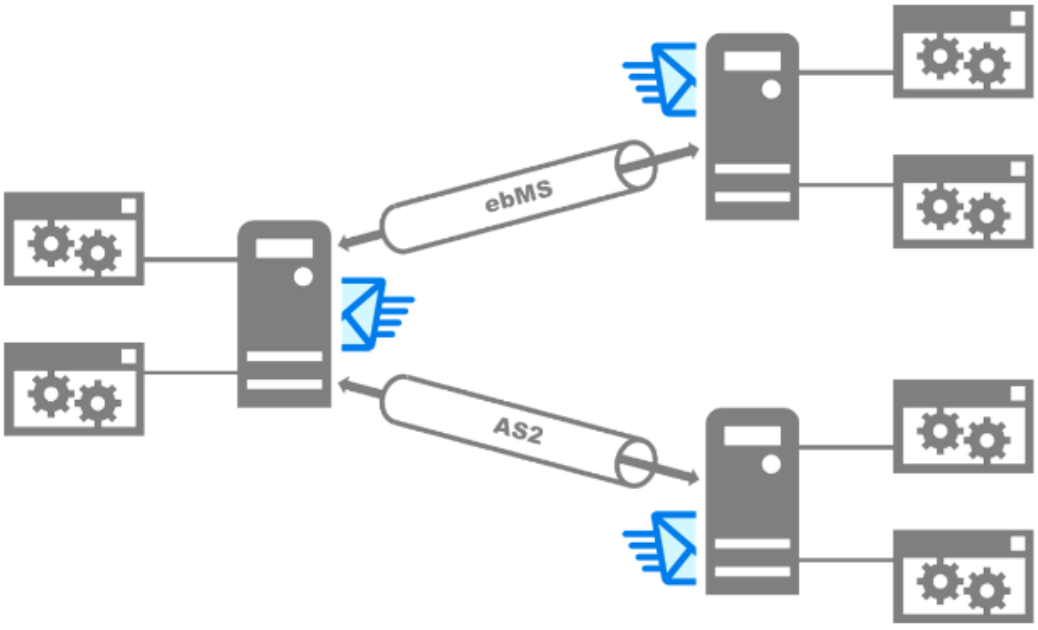
Nov 21, 2017

Contents

CHAPTER 1

Proven Solution to Automate B2B Transactions

Hermes Business Messaging Gateway is a proven open-source solution for enterprises to automate business transactions with business partners through secure and reliable exchange of electronic documents (e.g., purchase orders). Hermes is secure; it allows you to encrypt and digitally sign the documents for transmission. Hermes is reliable; the sender can automatically retransmit a message when it is dropped in the network while the receiver can guarantee every message is delivered once and only once, and in the right order.



CHAPTER 2

EDI over the Internet

Electronic Data Interchange (EDI) was developed as the de facto standard for organizations to exchange business data. EDI is running on private networks and based on a cryptic protocol, which makes implementation complicated, expensive, and flexible. These disadvantages have limited the EDI usage to very large organizations only. Hermes is designed to use the Internet, Public Key Infrastructure (PKI), and XML technologies to replace the EDI as a more affordable and extensible solution. Hermes supports mainstream business-to-business (B2B) transport protocols, such as ebXML Message Service 2.0 (ebMS 2.0) and Applicability Statement 2 (AS2). (The ebMS 3.0 / AS4 support is currently under development.).

CHAPTER 3

Unified and Extensible B2B Messaging Framework

Hermes unifies different transport protocols into a single B2B messaging framework. Based on this framework, you can easily develop an enterprise application to exchange business data with trading partners via different transport protocols. Designed to be extensible, Hermes provides an open Simple Plug-in Architecture (SPA) for developers to implement new messaging protocols as plug-ins. Most importantly, Hermes is open-source; you can freely extend Hermes and integrate it with other systems to meet your own business needs.

Quickstart

Install Hermes with Docker

1. Install the Docker Engine.
2. Run the Docker container for Hermes database (MySQL).

```
docker run --name hermes_db -e MYSQL_ROOT_PASSWORD=corvus -d cecid/hermes_db:2.2
```

3. Run the Docker container for Hermes application server (Tomcat).

```
docker run --name hermes_app --link hermes_db:db -p 8080:8080 -d cecid/hermes_
↪app:2.2
```

4. Log in to the Hermes administration console at `http://localhost:8080/corvus/admin/home` (username:`corvus`, password:`corvus`) to check if Hermes is up and running.

Note:

1. When it is the first time to run a container, the Docker image will be downloaded from the Docker Hub. The Docker images are large. The sizes of `cecid/hermes_db` and `cecid/hermes_app` are about 400MB and 1.4GB.
2. You may need the administrator or root privilage to execute `docker run`.

Loopback Messaging with Sample Clients

Preparation

Linux / Unix:

1. Install Java 8 or above, or OpenJDK 8 or above.
2. Please check Java 8 has been set up properly.

```
java -version
```

   If the above command fails to run, please try to set environment variable `JAVA_HOME` to the directory where Java is installed. On Ubuntu, you may use the following command to locate the Java home directory, e.g., `/usr/lib/jvm/java-8-openjdk-amd64`.

```
update-java-alternatives -l
```

3. Download and extract the `Hermes sample clients` to a working directory *<WorkDir>*

```
cd <WorkDir>
curl -O http://hermes.cecid.org/en/latest/_downloads/Hermes_client_sample.zip
unzip Hermes_client_sample.zip
sudo chmod -R 755 sample
```

4. Change the current directory to *<WorkDir>*`/sample`.

Windows:

1. Install Java 8 or above.
2. Please check Java 8 has been set up properly.

```
java -version
```

   If the above command fails to run, please try to set environment variable `JAVA_HOME` to the directory where Java is installed.

1. Download and extract the `Hermes simple clients` to a working directory *<WorkDir>*`/sample`.
2. Change the current directory to *<WorkDir>*`/sample`.

Create Loopback Partnership

Linux / Unix:

```
./ebms-partnership.sh
```

Windows:

```
ebms-partnership.bat
```

You will see the following message.

```
-------------------------------------------------
      EBMS Partnership Maintainance Tool
-------------------------------------------------
Initialize logger ..
Importing EBMS partnership parameters ...
```

```
Importing EBMS administrative sending parameters ...
Initialize EBMS HTTP data service client...
log4j:WARN No appenders could be found for logger    (org.apache.commons.httpclient.
↪HttpClient).
log4j:WARN Please initialize the log4j system properly.
Sending    EBMS HTTP partnership maintenance request ...

                    Sending Done:
-----------------------------------------------------
The result status : Operation executed successfully.
Please view log for details ..
```

Send Loopback Message

Linux / Unix:

```
./ebms-send.sh
```

Windows:

```
ebms-send.bat
```

This program sends a request attached with the payload named `testpayload` under the directory *<WorkDir>*/`sample/config/ebms-send` to local Hermes server. You will see the following message.

```
-----------------------------------------------------
          EbMS sender web service client
-----------------------------------------------------
Initialize Logger ...
Importing  ebMS sending parameters ... ./config/ebms-send/ebms-request.xml
Importing  ebMS partnership parameters ... ./config/ebms-partnership.xml
Initialize ebMS web service client...
Adding     payload in the ebMS message...
Sending    ebMS sending request ...

                    Sending Done:
-----------------------------------------------------
New message id: 20170204-090520-45900@172.17.0.3
```

Query Message History

Linux / Unix:

```
./ebms-history.sh
```

Windows:

```
ebms-history.bat
```

This program lists all sent and received messages. You will see the following message.

```
-----------------------------------------------------
         EbMS Message History Queryer
-----------------------------------------------------
Initialize Logger ...
```

```
Importing  ebMS config parameters ... ./config/ebms-history/ebms-request.xml
Initialize ebMS messsage history queryer ...
Sending ebMS message history query request ...

                    Sending Done:
-----------------------------------------------------
-----------------------------------------------------
         EbMS Message Query Result
-----------------------------------------------------
0    | Message id : 20170204-090520-45900@172.17.0.3 | MessageBox: outbox
1    | Message id : 20170204-090520-45900@172.17.0.3 | MessageBox: inbox
-----------------------------------------------------

Select message (0 - 1), -1 to exit: 0
```

Enter `0` to check the sent message and the following message will be displayed:

```
Sending    EBMS-status sending request ...

                 Sending Done:
-----------------------------------------------------
Query Message ID          : 20170204-090520-45900@172.17.0.3
Query Message Status      : DL
Query Message Status Desc : Message was sent.
ACK   Message ID          : null
ACK   Message Status      : null
ACK   Message Status Desc : null

-----------------------------------------------------

Please view log for details ..
```

Download Payload of Received Message

Linux / Unix:

```
./ebms-history.sh
```

Windows:

```
ebms-history.bat
```

You will see the following message.

```
-----------------------------------------------------
         EbMS Message History Queryer
-----------------------------------------------------
Initialize Logger ...
Importing  ebMS config parameters ... ./config/ebms-history/ebms-request.xml
Initialize ebMS messsage history queryer ...
Sending ebMS message history query request ...

                    Sending Done:
-----------------------------------------------------
-----------------------------------------------------
         EbMS Message Query Result
```

```
--------------------------------------------------
0    | Message id : 20170204-090520-45900@172.17.0.3 | MessageBox: outbox
1    | Message id : 20170204-090520-45900@172.17.0.3 | MessageBox: inbox
--------------------------------------------------

Select message (0 - 1), -1 to exit: 1
Currrent Dir: /home/cecid/WorkDir/sample
Please provide the folder to store the payload(s):
Initialize ebMS receiving web service client...
Sending    ebMS receiving request ... for 20170204-090520-45900@172.17.0.3
--------------------------------------------------

Please view log for details ..
```

1. Enter `1` to select the received message and you will be asked to the folder to store the payloads.
2. Press enter to save the payload in the current folder. A file named `ebms.<timestamp>@127.0.1.1.Payload.0` will be downloaded, where *<timestamp>* indicates the time **ebms-send** was executed.
3. Open the payload file and you will see the following content:

```
This is an sample message.

                  :#+,
                 +'++
               ,++'+
              ++'+#`
            ;+''++              `           `           `
           #++''+;`          `++++` ``,:;::   `,::::  ++  ;+'++;
         ;++''+++          '++++#` `;:;;;;  `;;;;;;  ++  ;'+++++:
        +'''''++:``;;;:   +#`      ,;;      `::,      ++  ;+   `+#
       +'''''''' ;:;;;: ;+:       ;;``      ;;`       ++  ;+`   ++`
      ,+'+''''++ .:;:;;; ;+.      `;;,.,,, `;;`       ++  ;+    ;'.
      ''+''''''+`,;;;::;`'+......`;::;;;;`.;;``..`.`+#``;+``..:+:`..`
      `++'''+'++ `;;;;;; ''`       :;.```` `:;        ++  ;+`   :'.
       ;'''+'''+` ;:::;. :':`      ;; `      ;;        ++  ;+    +'`
        :+'+'+''+  .,,` ` ++`      ::,````   ::,````   ++  ;+   .+#
      `   #'+'''+`         ''++++` `;;;;;;  `;;;;;;  ++  ;'+++++,`
           ,++'''#          .;;''`    .:::,`   `,:::,  ''  :''''';
             +'+''; `                            `
             `,+''',
                '''+.
                 `+++`
                  :+:

This is an sample message.
```

Installing Hermes

Introduction

The Hermes installer is packaged in a self-extracted Java archive (JAR). Please download the source code from GitHub and build the Hermes Installer by Maven. Otherwise, you may download the JAR file from `hermes2_installer.jar`. Upon proper invocation, you will see an installation wizard, either in graphical or text format. Following the steps will install the following components:

- Hermes core
- Hermes plugins (AS2 / AS2 Plus / ebMS)
- Database tables of Hermes plugins for one of the following database:
  - Postgres 9.2 or later
  - Oracle 11gR2 or later
  - MySQL 5.5 or later with InnoDB storage engine supported
- Web service usage sample

Prerequisite

1. Java SE Development Kit 8
2. Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files
   (a) Download the JCE Unlimited Strength Jurisdiction Policy Files for JDK 8 from

   http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

   (b) Unzip the downloaded file

   ```
   unzip jce_policy-8.zip
   ```

   (c) Replace the two jar files `local_policy.jar` and `US_export_policy.jar` in the directory `/usr/lib/jvm/java-8-oracle/jre/lib/security` with the corresponding jar file unzipped in the previous step.

   ```
   cd UnlimitedJCEPolicyJDK8
   sudo cp local_policy.jar /usr/lib/jvm/java-8-oracle/jre/lib/security/local_
   ↪policy.jar
   sudo cp US_export_policy.jar /usr/lib/jvm/java-8-oracle/jre/lib/security/US_
   ↪export_policy.jar
   ```

3. Tomcat 8.5 or above with port `8080`
   (a) Edit `/etc/systemd/system/tomcat.service`. Change `Environment=JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64/jre` to `Environment=JAVA_HOME=/usr/lib/jvm/java-8-oracle/jre`

   (b) Restart Tomcat

Note: To access the admin page, you will need to have a Tomcat user with an admin role. One way to do this is to define the user in `tomcat-users.xml`. Please refer to the Realm Configuration section in the Tomcat documentation for more details.

Sample of `tomcat-user.xml`:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="admin"/>
    <role rolename="api"/>
    <user username="corvus" password="corvus" roles="tomcat,admin,api"/>
</tomcat-users>
```

4. One of the following databases installed on any server:
    - PostgreSQL 9.2 or later. *<POSTGRES_HOME>* is referring to the home directory of PostgreSQL in the remaining parts of the document.
    - MySQL 5.5 or later. *<MYSQL_HOME>* is referring to the home directory of MySQL in the remaining parts of the document.
    - Oracle 11gR2 or later. *<ORACLE_HOME>* is referring to the home directory of Oracle in the remaining parts of the document.

Installation

Step 1 – Environment setup

Install all the prerequisite items. The rest of this guide assumes that they are all running on the same machine.

Step 2 – Database Configuration

Postgres

1. Create a database user with username `corvus` and password `corvus`.
    (a) Open a command prompt
    (b) Go to *<POSTGRES_HOME>*`/bin`
    (c) Type `createuser -A -d -P -U` *<POSTGRES_ADMIN>* `corvus` where *<POSTGRES_ADMIN>* represents the name of an administrator/super-user in the PostgreSQL database. This value is `postgres` if not specified. This may require a super user or Postgres owner to execute in Linux.
    (d) Enter the password `corvus`
    (e) Enter the password again for confirmation
    (f) Enter the PostgreSQL administrator password for creating a new user role.
2. Create two databases named `as2` and `ebms` with the `corvus` user.
    (a) Open a command prompt
    (b) Go to *<POSTGRES_HOME>*`/bin`
    (c) Type `createdb -U corvus -W as2`
    (d) Enter the password `corvus`
    (e) Repeat steps 2.3 - 2.4 for the `ebms` database.

MySQL

1. Create two databases named `as2` and `ebms` with username `corvus` and password `corvus`.
    (a) Open a command prompt
    (b) Go to *<MYSQL_HOME>*`/bin`
    (c) Type `mysql -u` *<MYSQL_ADMIN>* `-p` where *<MYSQL_ADMIN>* represents the name of an administrator/super-user in the MySQL database. This is `root` by default. This may require super user or MySQL owner to execute in Linux.
    (d) Enter the command below to create the `as2` database. Note that specifying collate to `latin1_general_cs` is essential.

```
create database as2 collate=latin1_general_cs;
```

(e) Enter the command below to create and assign access privileges to user `corvus`.

```
grant all on as2.* to 'corvus'@'localhost' identified by 'corvus';
```

(f) Repeat steps 1.4 – 1.5 for the `ebms` database.

Oracle

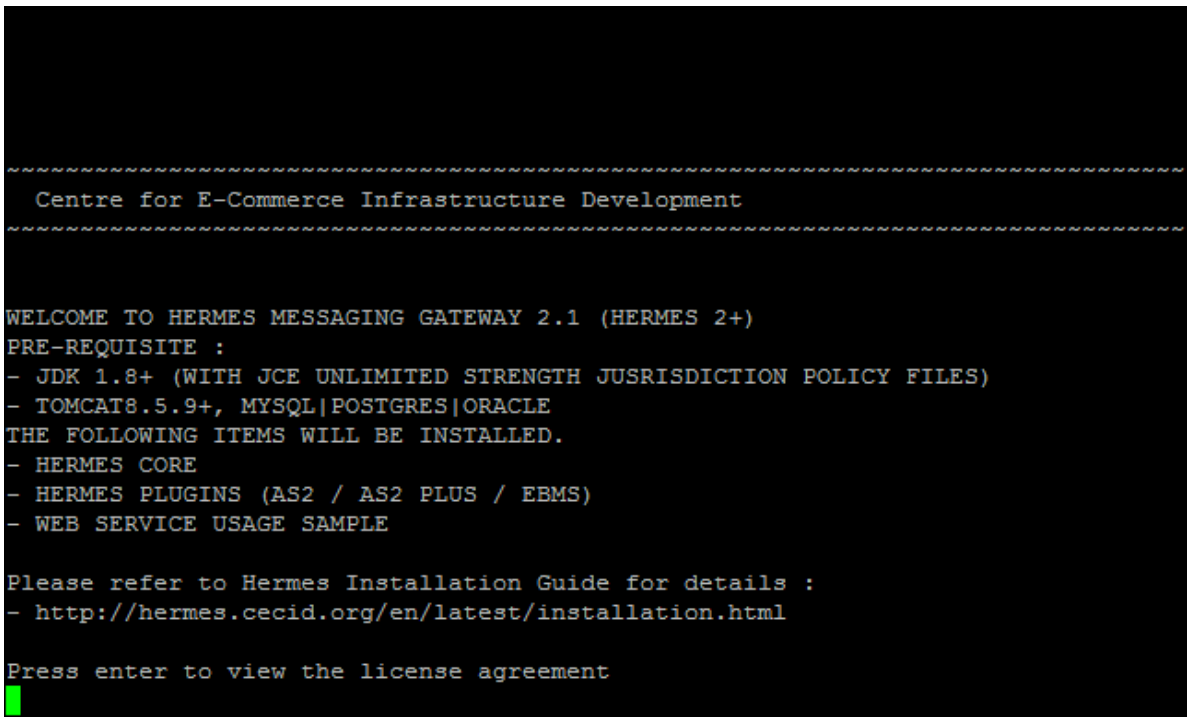
Oracle database creation involves a number of steps and custom parameters for different requirements for the database server. We propose the following reference as a guideline for creating an Oracle database for Hermes:

https://docs.oracle.com/cd/E11882_01/server.112/e10897/install.htm#ADMQS0232

Step 3 – Hermes Deployment

1. Execute the installer

   - For Unix/Linux, open **`terminal`** and type `sudo java -jar hermes2_installer.jar`.



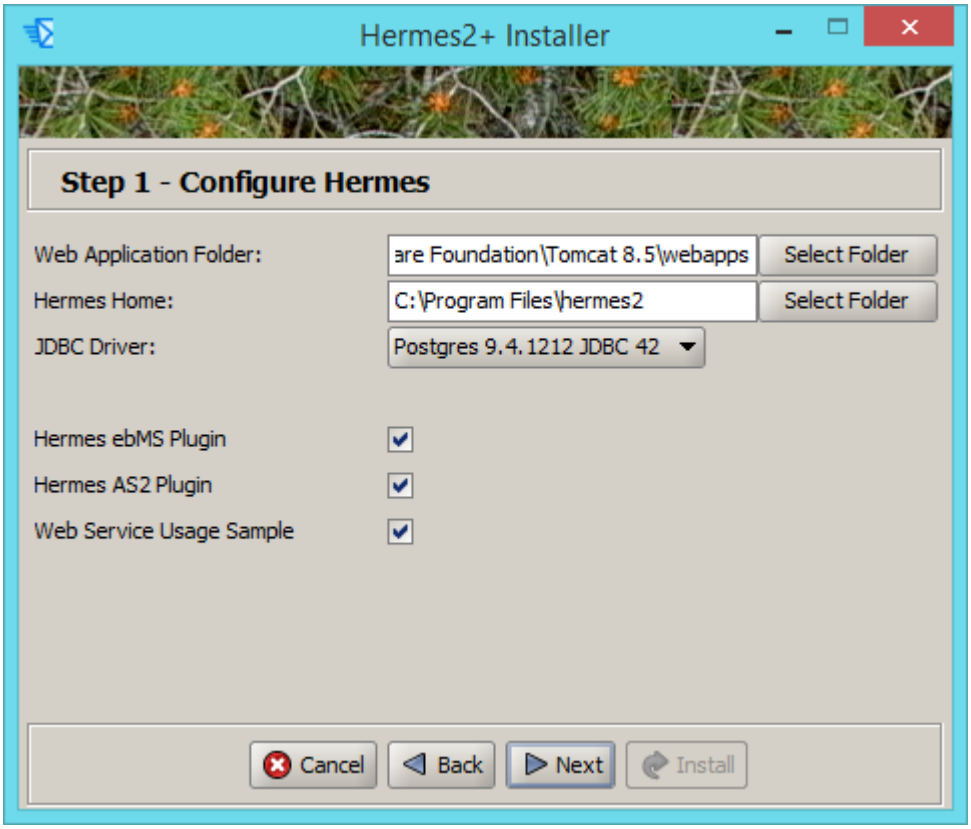
   Press *Enter* until you get to Screen in "2. Step 1 - Configure Hermes Core".

   - For Windows, open a command prompt as an Administrator and type `java -jar hermes2_installer.jar` or if **`java`** is not set in your environment path, specify the full path.

Click *Next* until you get to Screen in "2. Step 1 - Configure Hermes Core".

2. Step 1 - Configure Hermes Core

```
/home/cecid/webapps

Hermes Home:    [default:/home/hermes2]
/home/cecid/hermes2

JDBC Driver:
  view available options
  1) Postgres 9.4.1212 JDBC 42 [default]
  2) Oracle (Download manually)
  3) MySQL (Download manually)
  Enter a number



Install the following component?
Hermes ebMS Plugin    [default:true]


Hermes AS2 Plugin    [default:true]


Install the following component?
Web Service Usage Sample    [default:true]
```



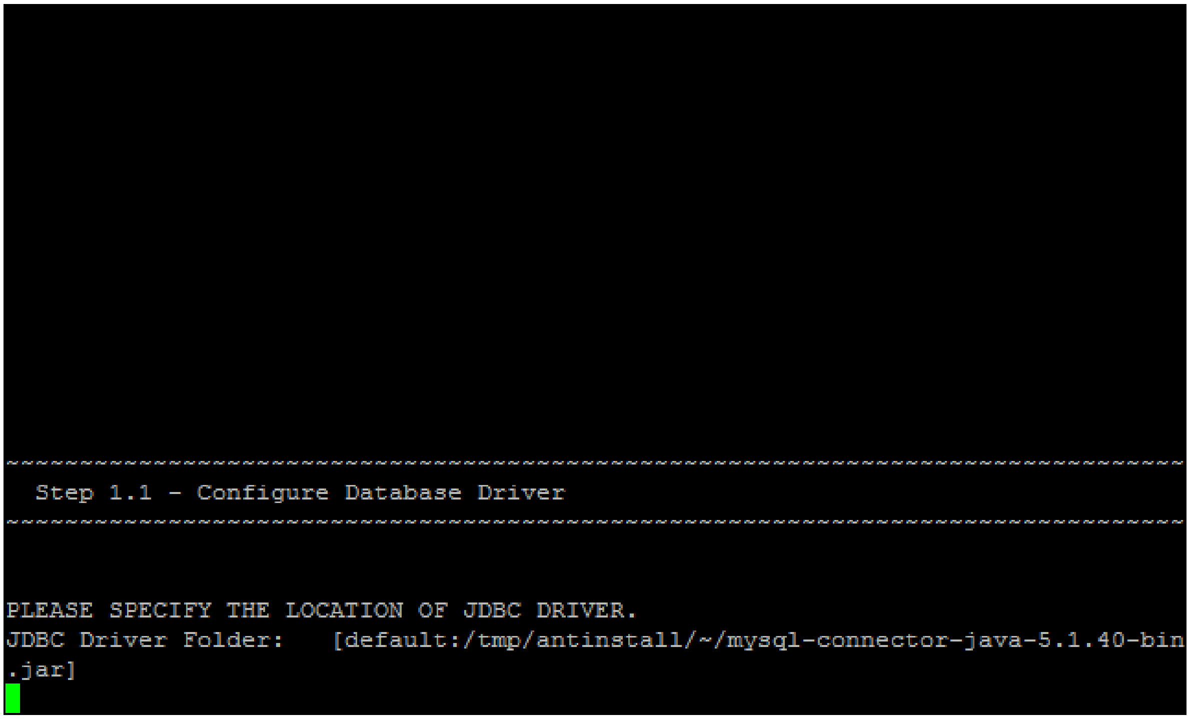
Description of the settings:

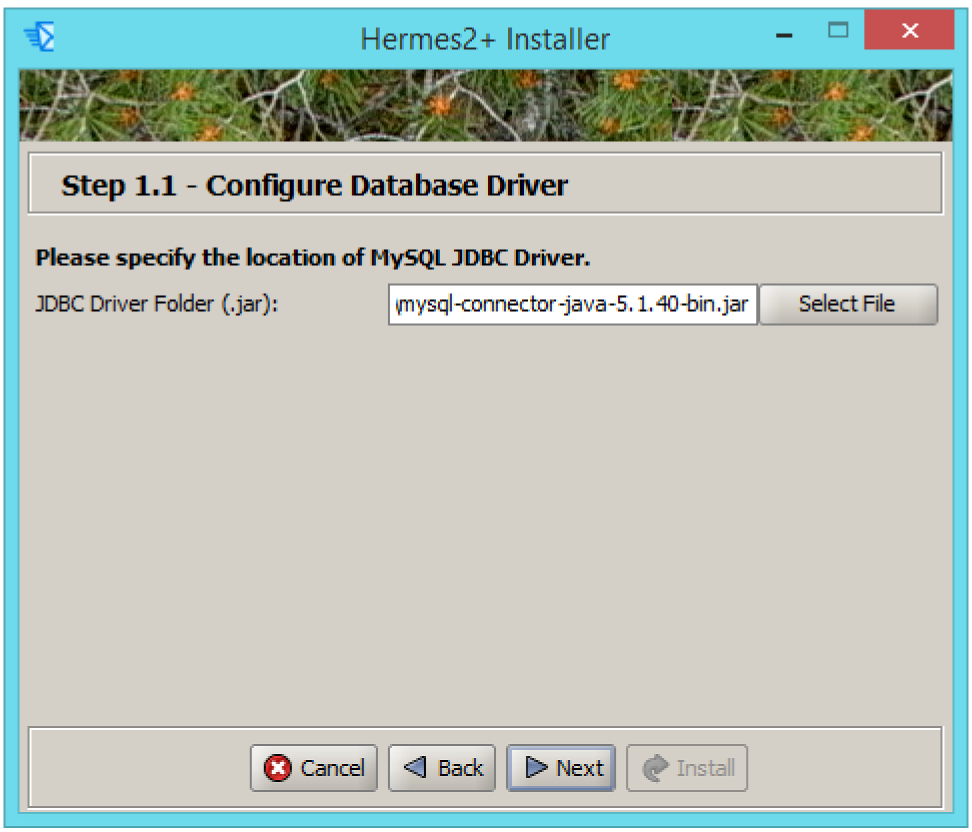
Web Application Folder	Folder to place the web application (e.g `webapps`) in Tomcat.
Hermes Home	Location to place the Hermes core library and some related files.
JDBC Driver	Specify which database vendor to connect to. One of the following 3 database vendors can be selected: • Postgres • Oracle • MySQL
Hermes ebMS Plugin	Optional. Install the ebMS component.
Hermes AS2 Plugin	Optional. Install the AS2 component.
Web Service Usage Sample	Optional. Install the sample program of web service client.

Click *Next* and press *Yes* if the installer prompts you to create a new directory.

3. Step 1.1 - Configure Database Driver

   Oracle and MySQL drivers need to be downloaded manually. Once this is done, specify the location of the driver:



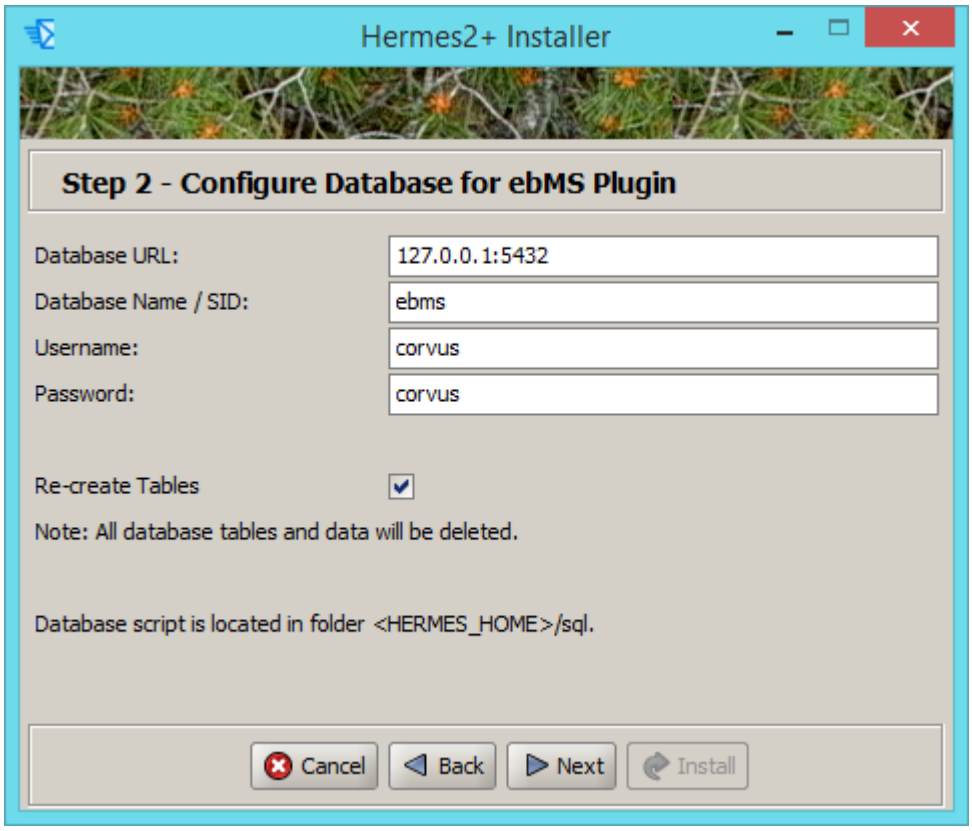


Description of the settings:

JDBC Driver Folder (.jar)	Directory of the downloaded JDBC driver.

4. Step 2 - Configure Database for ebMS Plugin (Optional)

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Step 2 - Configure Database for ebMS Plugin
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Database URL:    [default:127.0.0.1:5432]


Database Name / SID:    [default:ebms]


Username:    [default:corvus]


Password:    [default:]



Install the following component?
Re-create Tables    [default:false]
```

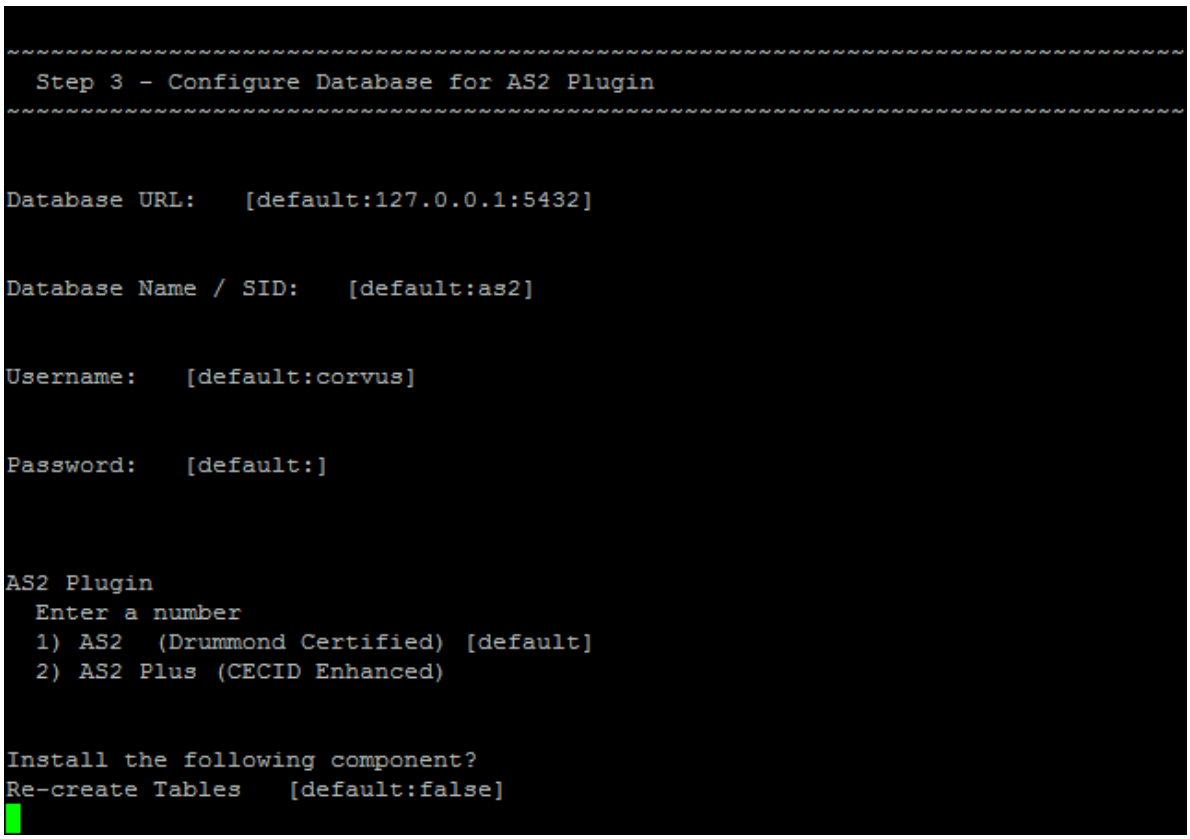


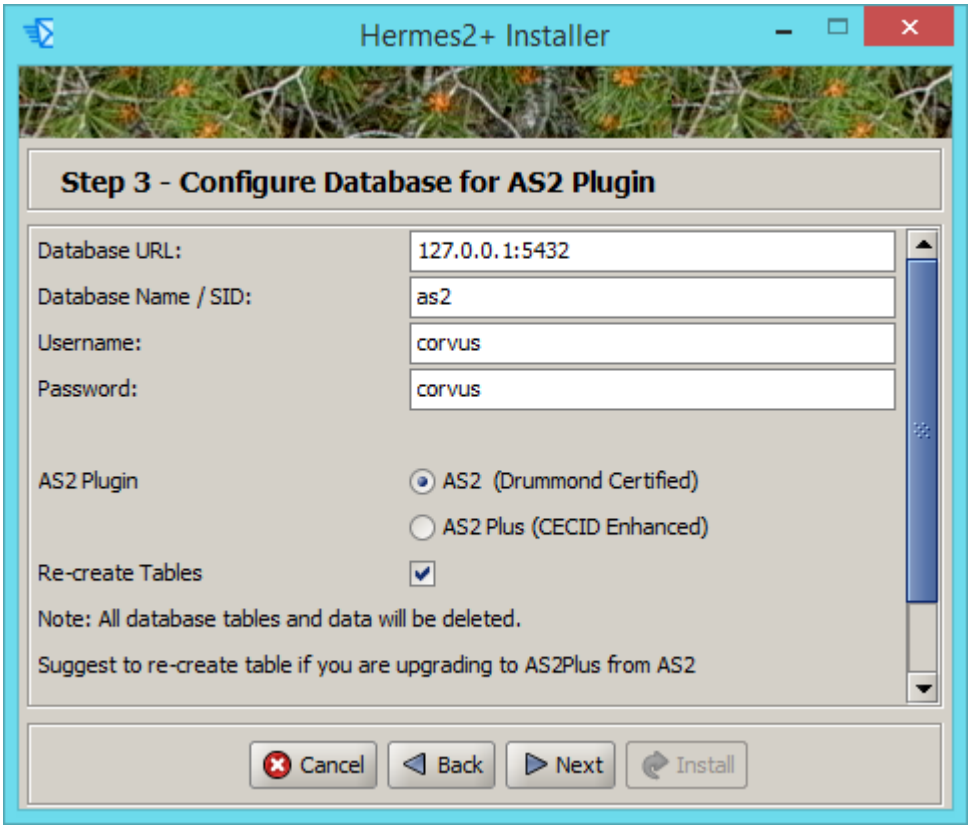
Description of the settings:

Database URL	The URL address of the database server. Port number may be attached to the address with the format *<HOST_ADDRESS>:<PORT>* where *<HOST_ADDRESS>* is the address of the database server and *<PORT>* is the port number of the database server address.
Database Name/SID	For Postgres and MySQL, please specify the name of the database. For Oracle, please specify the Oracle System ID (SID).
Username	Username to access the database.
Password	Password to access the database.
Re-create Tables	Optional. Re-create all the tables in the specified database. **Important Notes:** • If this is your **first time** installing Hermes, please check this option. • If you choose to re-create the tables, all of the existing data in the specified database will be removed during installation. Please backup all the data in the selected database before choosing to re-create the tables.

If you followed the prerequisite installation procedures above, you can just leave it as the default. Click *Next* when you have finished the configuration.

5. Step 3 - Configure Database for AS2 Plugin (Optional)





Description of the settings:

Database URL	The URL address of the database server. Port number may be attached to the address with the format *<HOST_ADDRESS>*:*<PORT>* where *<HOST_ADDRESS>* is the address of the database server and *<PORT>* is the port number of the database server address.
Database Name/SID	For Postgres and MySQL, please specify the name of the database. For Oracle, please specify the Oracle System ID (SID).
Username	Username to access the database.
Password	Password to access the database.
AS2 Plugin	**AS2:** Original AS2 plugin certified by Drummond Group Inc. **AS2 Plus:** Built based on AS2 plugin with new/enhanced features.
Re-create Tables	Optional. Re-create all the tables in the specified database. **Important Notes:** • If this is your **first time** installing Hermes, please check this option. • If you are switching from AS2 to AS2 Plus or vice versa, we highly recommend you check this option. • If you choose to re-create the tables, all of the existing data in the specified database will be removed during installation. Please backup all the data in the selected database before choosing to re-create the tables.

If you followed the prerequisite installation procedures above, you can just leave it as the default. Click *Next* when you have finished the configuration.

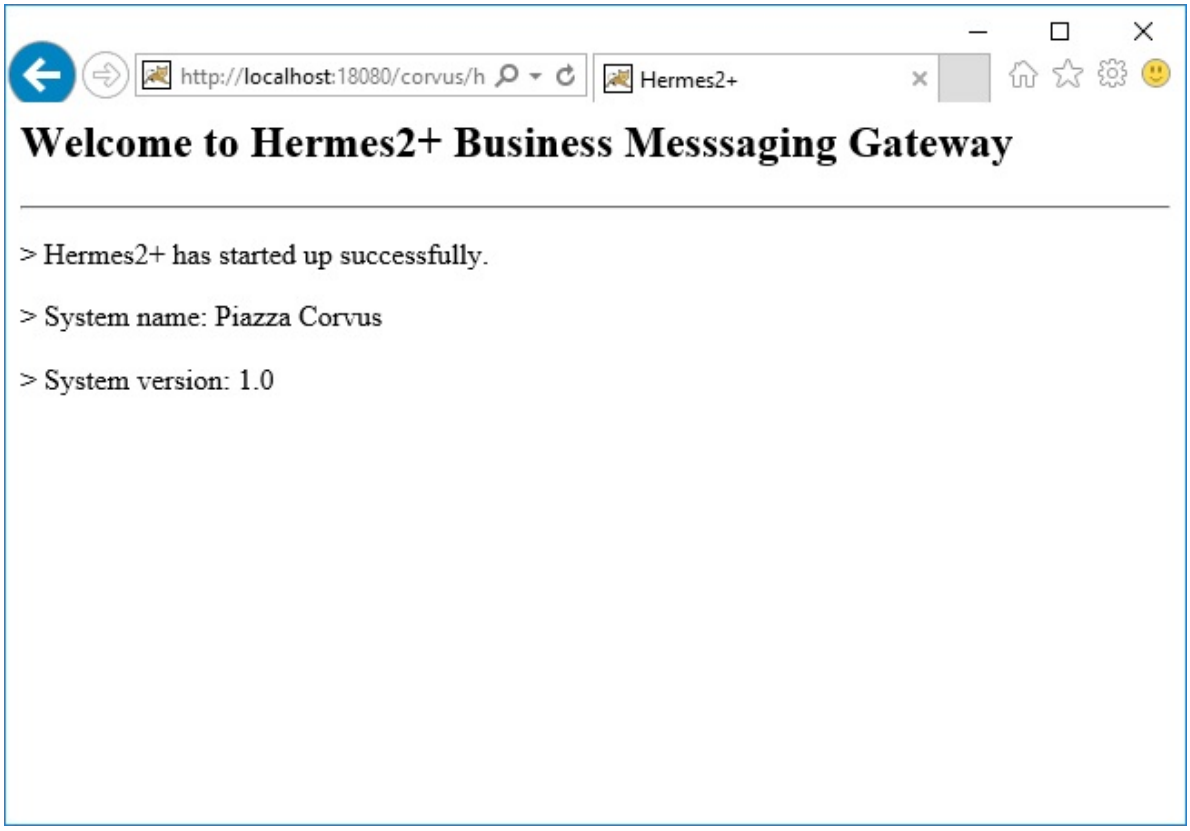
6. Click on *Install* and you're done!

Step 4 – Start Hermes2

1. Checklist:
   - Java JDK 8 or above with Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7.
   - Apache Tomcat 8.5 or above Servlet/JSP Container.
   - Database server is running with ebMS and/or AS2 database instances and the tables are created.
   - If you are running Unix/Linux, make sure that at least read permissions are set to the core directory and read/write for the AS2 repository directory in *<HERMES2_HOME>*.
   - Start Tomcat.
2. To verify that Hermes is running, access the following URL from a web browser:

   http://localhost:8080/corvus/home

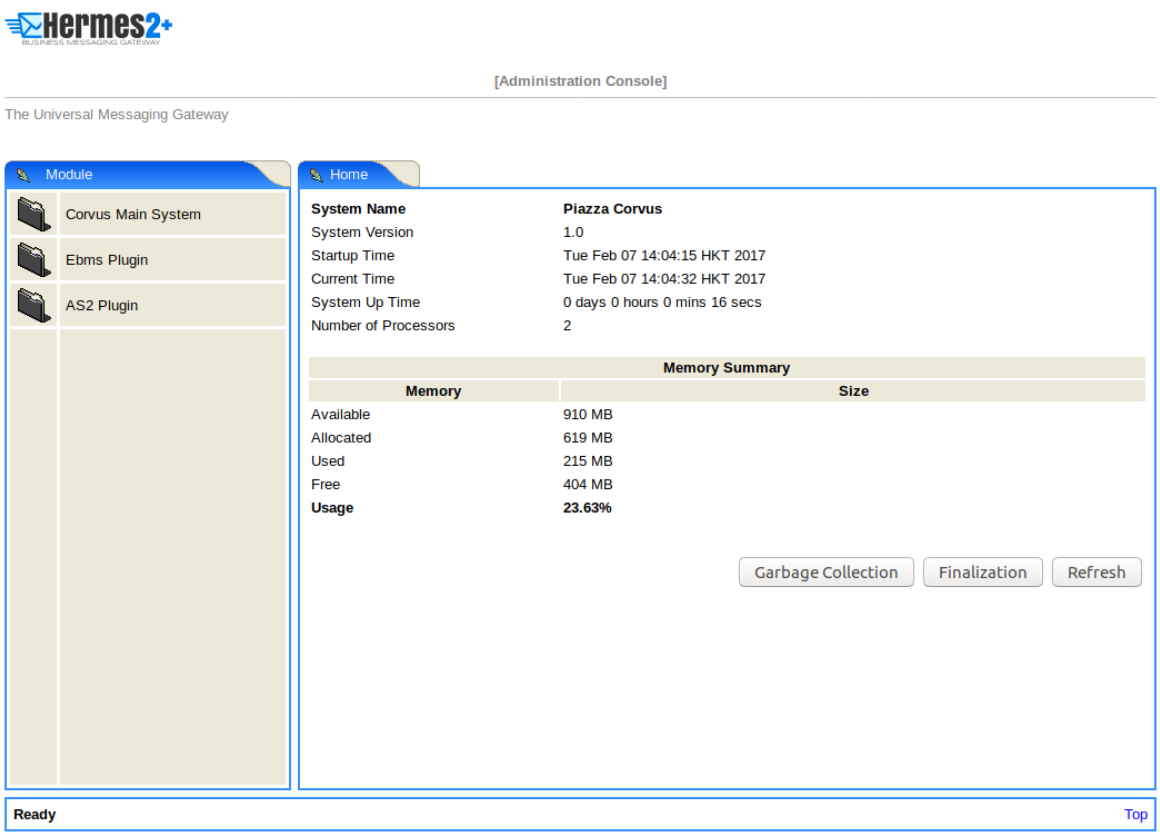
   The welcome page should be displayed as below:



3. To access the admin page, go to the following URL. The login user and password are the same as the Tomcat user with admin privileges specified in Point 3 of *Prerequisite*.

   http://localhost:8080/corvus/admin/home

4. Once you have gained access to the admin page, you should see the Hermes Administration Console page:



That's it! Hermes should now be up and running. You can test your setup by running our web service usage sample in next section.

Partnership Maintenance and Web Service Usage Sample

A tool kit called **`Web Service Usage Sample`** was installed under the *<HERMES2_HOME>*`/sample` folder. It contains tools to test the installed Hermes.

Directory Organization

Directory/File	Description
`config/*`	Contains the configuration file for the sample programs. The folders inside this directory contain related files for specific sample programs.
`config/ebms-partnership.xml` `config/as2-partnership.xml`	These two files contain partnership settings for ebMS and AS2 that are used by the sample programs.
`logs/*`	A set of logs that contain the output from each sample program.
`lib/*`	The library files required for the sample programs.
`*.bat/*.sh`	The scripts for executing the sample programs.

Preparation

Windows environment

1. Set environment variable `JAVA_HOME` to the directory where Java is located.

> **Note:** To run the sample program, Administrator privilege is required.

UNIX environment

1. Set environment variable `JAVA_HOME` to the directory where Java is located.
2. Change the owner and the group of *<HERMES2_HOME>* and *<TOMCAT_HOME>*`/webapps/corvus` with the following commands:

```
sudo chown -R tomcat:<OWNER_GROUP> <HERMES2_HOME>
sudo chown -R tomcat:<OWNER_GROUP> <TOMCAT_HOME>/webapps/corvus
```

3. Change the permissions of all files in *<HERMES2_HOME>* and *<TOMCAT_HOME>*`/webapps/corvus` to `775` with the following command:

```
sudo chmod -R 775 <HERMES2_HOME>
sudo chmod -R 775 <TOMCAT_HOME>/webapps/corvus
```

Partnership Maintenance

Users need to define a **partnership**, which contains the messaging details between sender and recipient. It is required to identify the sender and the recipient when transporting messages.

A web service sample program is provided to manage partnerships (add, update or delete). The partnership configuration for the AS2/ebMS loopback test is placed in *<HERMES2_HOME>*`/sample/config/`*<as2/ebms>*`-partnership.xml`.

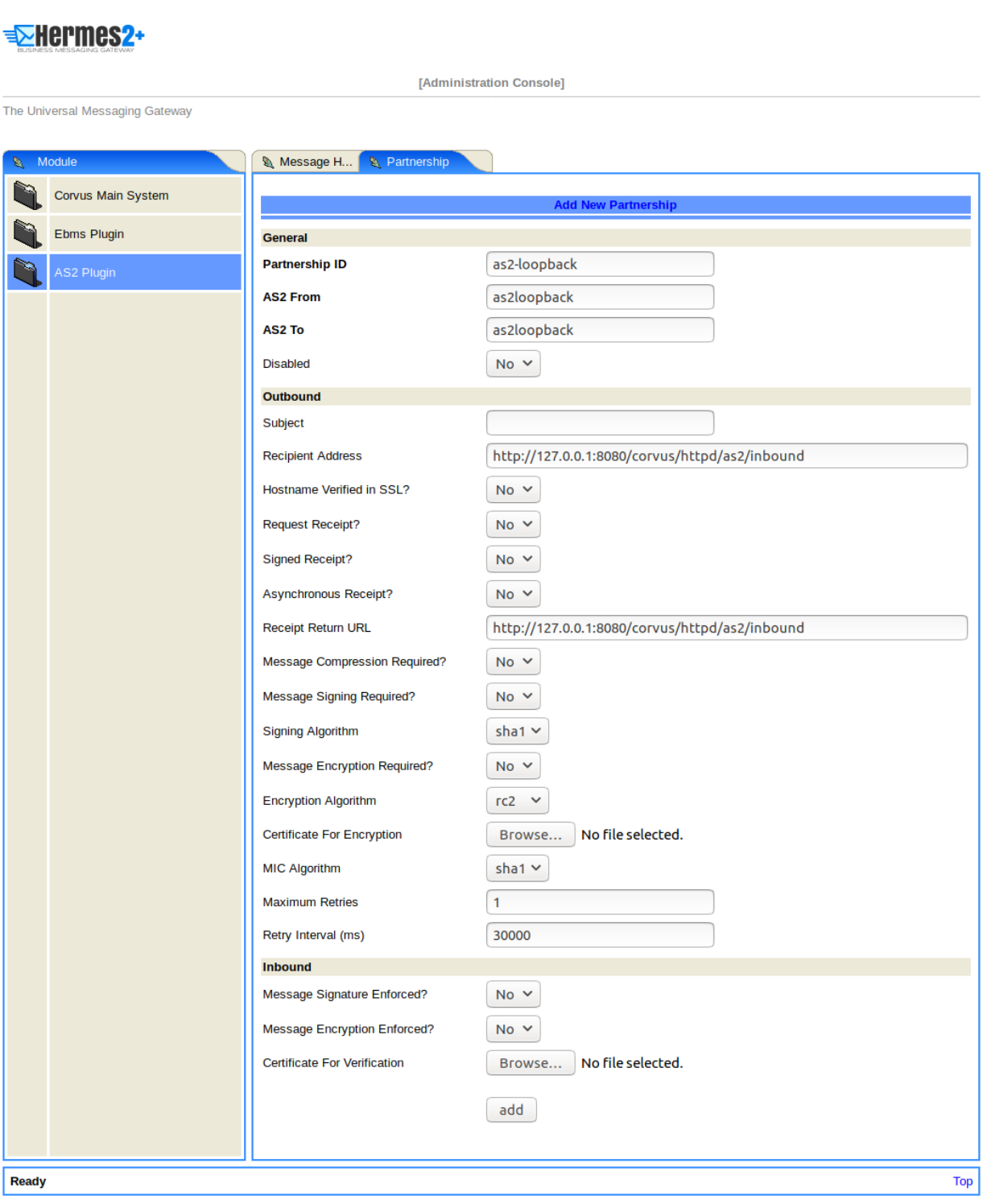
Program	Purpose
`as2-partnership` / **`ebms-partnership`**	Maintains a specified AS2/ebMS partnership in Hermes.

Creating an AS2 Partnership

To create the partnership required to perform the AS2 messaging loopback test using *AS2 Web Service Usage Sample*, you need to execute the script **`as2-partnership`**.

Or:

Access http://localhost:8080/corvus/admin/as2/partnership to configure the partnership manually. Below is a simple loopback configuration sample:


Hermes2+
BUSINESS MESSAGING GATEWAY
[Administration Console]
The Universal Messaging Gateway
Module
Message H...
Partnership
Corvus Main System
Ebms Plugin
AS2 Plugin
Add New Partnership
General
Partnership ID
as2-loopback
AS2 From
as2loopback
AS2 To
as2loopback
Disabled
No
Outbound
Subject
Recipient Address
http://127.0.0.1:8080/corvus/httpd/as2/inbound
Hostname Verified in SSL?
No
Request Receipt?
No
Signed Receipt?
No
Asynchronous Receipt?
No
Receipt Return URL
http://127.0.0.1:8080/corvus/httpd/as2/inbound
Message Compression Required?
No
Message Signing Required?
No
Signing Algorithm
sha1
Message Encryption Required?
No
Encryption Algorithm
rc2
Certificate For Encryption
Browse...
No file selected.
MIC Algorithm
sha1
Maximum Retries
1
Retry Interval (ms)
30000
Inbound
Message Signature Enforced?
No
Message Encryption Enforced?
No
Certificate For Verification
Browse...
No file selected.
add
Ready
Top

Partnership ID	`as2-loopback`
AS2 From	`as2loopback`
AS2 To	`as2loopback`
Disabled	*No*
Subject	none
Recipient Address	http://127.0.0.1:8080/corvus/httpd/as2/inbound
Hostname Verified in SSL?	*No*
Request Receipt?	*No*
Signed Receipt?	*No*
Asynchronous Receipt?	*No*
Receipt Return URL	http://127.0.0.1:8080/corvus/httpd/as2/inbound
Message Compression Required?	*No*
Message Signing Required?	*No*
Signing Algorithm	*sha1*
Message Encryption Required?	*No*
Encryption Algorithm	*rc2*
Certificate For Encryption	none
MIC Algorithm	*sha1*
Maximum Retries	`1`
Retry Interval (ms)	`30000`
Message Signature Enforced?	*No*
Message Encryption Enforced?	*No*
Certificate For Verification	none

Creating an AS2 Plus Partnership

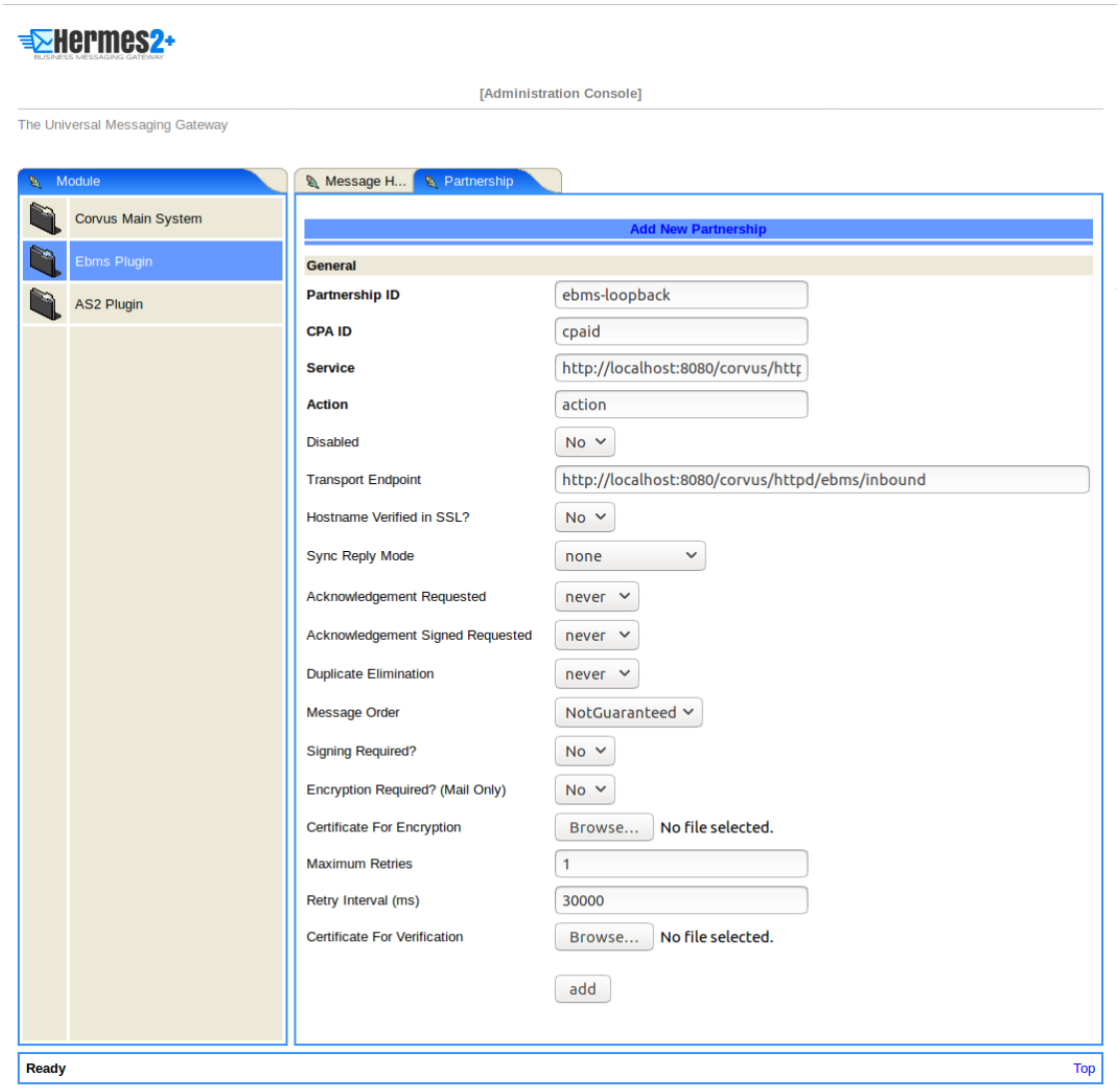
Please follow the same procedure listed in *Creating an AS2 Partnership* .

Creating an ebMS Partnership

To create the partnership required to perform the ebMS messaging loopback test using *ebMS Web Service Usage Sample*, you need to execute the script **`ebms-partnership`**.

Or:

Access http://localhost:8080/corvus/admin/ebms/partnership to configure the partnership manually. Below is a simple loop-back configuration sample:



Partnership ID	`ebms-loopback`
CPA ID	`cpaid`
Service	http://localhost:8080/corvus/httpd/ebms/inbound
Action	`action`
Disabled	*No*
Transport Endpoint	http://localhost:8080/corvus/httpd/ebms/inbound
Hostname Verified in SSL?	*No*
Sync Reply Mode	*none*
Acknowledgement Requested	*never*
Acknowledgement Signed Requested	*never*
Duplicate Elimination	*never*
Message Order	*NotGuaranteed*
Signing Required?	*No*
Encryption Required? (Mail Only)	*No*
Certificate For Encryption	none
Maximum Retries	`1`
Retry Interval (ms)	`30000`
Certificate For Verification	none

Web Service Usage Sample Flow

In order to validate the installation of Hermes, a web service usage sample program is provided. It can be executed by running the following scripts in a command prompt.

Program	Purpose
as2-send / **ebms-send**	Send an AS2/ebMS message to the installed Hermes.
as2-history / **ebms-history**	Show the message history of Hermes. This program will list the inbox and outbox messages in the data storage of Hermes. The user can view the details of the inbox and outbox. For inbox messages, the user can also download the payload in the repository of Hermes, if available.

In order to test whether Hermes has been installed successfully or not, we suggest running the sample programs in the following steps:

1. Send a message to the local Hermes by running **ebms-send/as2-send**.
2. Check the status of the sent message by running **ebms-history/as2-history** and select the message from the outbox.
3. Check the received message by running **ebms-history/as2-history** and select the message from the inbox to download the payload.

AS2 Web Service Usage Sample

Before executing the following AS2 web service usage sample, the partnership from *Creating an AS2 Partnership* must be created.

1. Send a message to the local Hermes using the script **as2-send**.

   This program creates and sends a request attached with the payload named `testpayload` under the directory *<HERMES2_HOME>*`/sample/config/as2-send` to Hermes.

   Upon successful execution, you should be able to see the similar output shown as follow:

```
------------------------------------------------
          AS2 Message Sender
------------------------------------------------
Initialize Logger ...
Importing  AS2 sending parameters ... ./config/as2-send/as2-request.xml
Importing  AS2 partnership parameters ... ./config/as2-partnership.xml
Initialize AS2 message sender...
Adding payload in the AS2 message...
Sending AS2 sending request ...

                    Sending Done:
------------------------------------------------
New message id: 20170207-155843-19800@127.0.1.1

Please view log for details ..
```

2. Check the sent message using the script **as2-history**.

   This program retrieves the list of sent/received message from Hermes.

```
--------------------------------------------------
        AS2 Message History Web Service Client
--------------------------------------------------
Initialize Logger ...
Importing  AS2 config parameters ... ./config/as2-history/as2-request.xml
Initialize AS2 messsage history queryer ...
Sending AS2 message history query request ...

                    Sending Done:
--------------------------------------------------
        AS2 Message that are matched
--------------------------------------------------
No. of message: 2
0       | Message id : 20170207-155843-19800@127.0.1.1  MessageBox: outbox
1       | Message id : 20170207-155843-19800@127.0.1.1  MessageBox: inbox
--------------------------------------------------

Select message (0 - 1), -1 to exit:
```

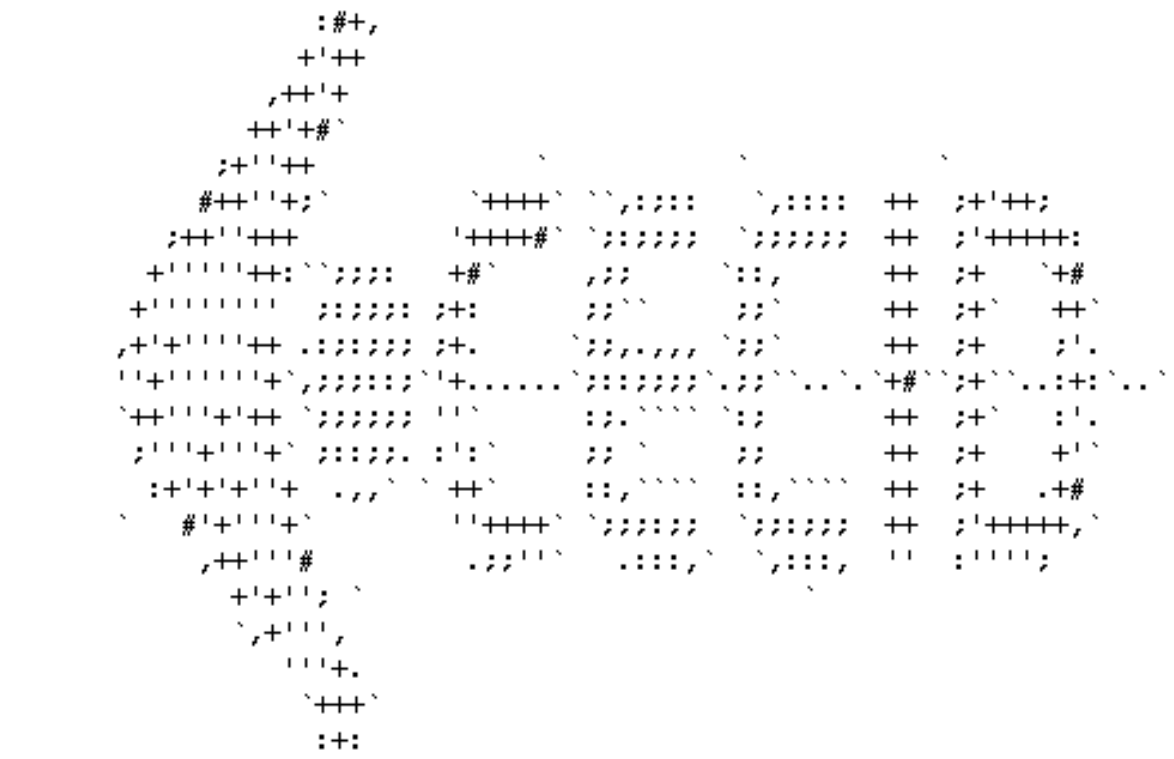
Enter `0` to check the sent message. A display similar to the following will appear:

```
Query Message ID          : 20170207-155843-19800@127.0.1.1
Query Message Status      : DL
Query Message Status Desc : null
ACK   Message ID          : null
ACK   Message Status      : null
ACK   Message Status Desc : null
--------------------------------------------------

Please view log for details ..
```

3. Check the received message and download the payload.

   From the select message screen of **`as2-history`**, enter 1 to select the inbox message and it will display `Please provide the folder to store the payload(s):`. Press enter to save the payload in the current folder. A file named `as2.<timestamp>@127.0.1.1.Payload.0` will be downloaded, where *`<timestamp>`* is the time **`as2-send`** was executed. Open that file and you will see the follow content:

```
This is an sample message.
```



```
This is an sample message.
```

ebMS Web Service Usage Sample

Before executing the following ebMS web service usage sample, the partnership from *Creating an ebMS Partnership* must be created.

1. Send a message to the local Hermes server using the script **`ebms-send`**.

   This program creates and sends a request attached with the payload named `testpayload` under the directory *<HERMES2_HOME>*`/sample/config/ebms-send` to Hermes.

   Upon successful execution, an output similar to the following will be displayed:

```
-----------------------------------------------------
         EbMS sender web service client
-----------------------------------------------------
Initialize Logger ...
Importing  ebMS sending parameters ... ./config/ebms-send/ebms-request.xml
Importing  ebMS partnership parameters ... ./config/ebms-partnership.xml
Initialize ebMS web service client...
Adding     payload in the ebMS message...
Sending    ebMS sending request ...

                  Sending Done:
-----------------------------------------------------
New message id: 20170207-160023-86402@127.0.1.1

Please view log for details ..
```

2. Check the sent message using the script **ebms-history**.

   This program retrieves the list of sent/received message from Hermes.

```
-----------------------------------------------------
         EbMS Message History Queryer
-----------------------------------------------------
Initialize Logger ...
Importing  ebMS config parameters ... ./config/ebms-history/ebms-request.xml
Initialize ebMS messsage history queryer ...
Sending ebMS message history query request ...

                    Sending Done:
-----------------------------------------------------
-----------------------------------------------------
         EbMS Message Query Result
-----------------------------------------------------
0       | Message id : 20170207-160023-86402@127.0.1.1 | MessageBox: outbox
1       | Message id : 20170207-160023-86402@127.0.1.1 | MessageBox: inbox
-----------------------------------------------------

Select message (0 - 1), -1 to exit:
```

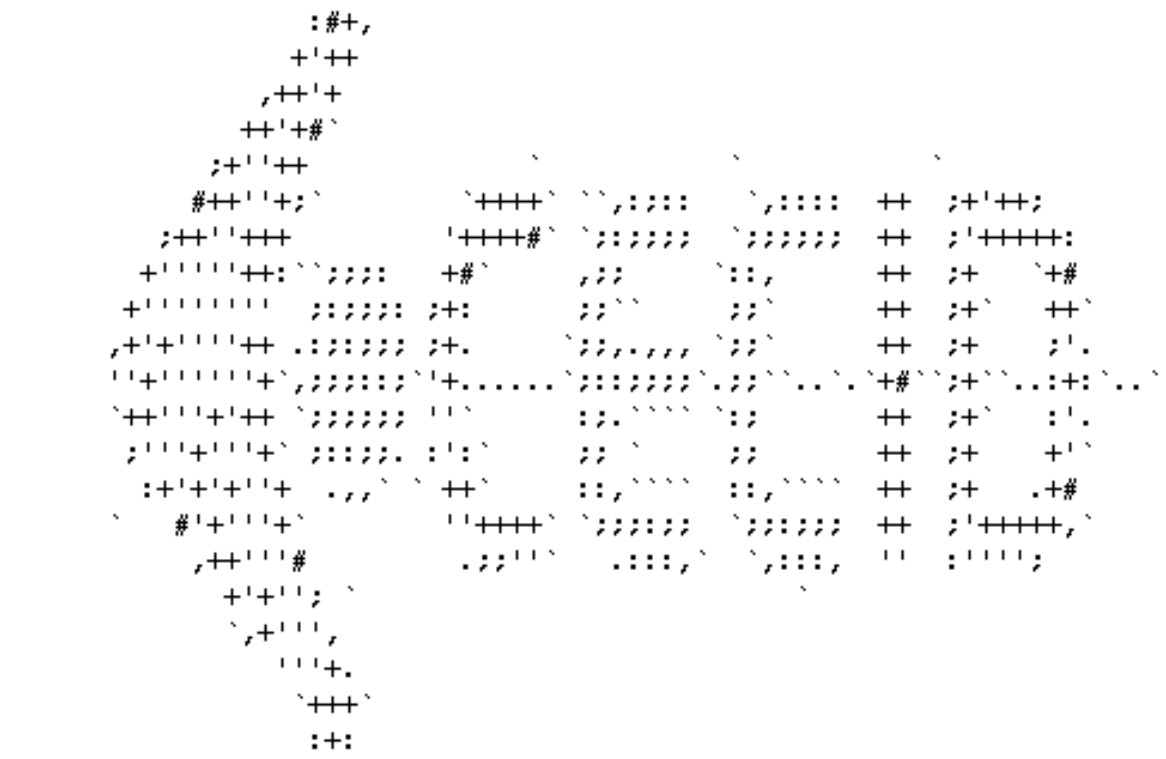
   Enter `0` to check the sent message and a screen similar to the following will be displayed:

```
Sending    EBMS-status sending request ...

                    Sending Done:
-----------------------------------------------------
Query Message ID          : 20170207-160023-86402@127.0.1.1
Query Message Status      : DL
Query Message Status Desc : Message was sent.
ACK   Message ID          : null
ACK   Message Status      : null
ACK   Message Status Desc : null

-----------------------------------------------------

Please view log for details ..
```

3. Check the received message and download the payload.

   From the select message screen of **ebms-history**, enter `1` to select the inbox message and it will display `Please provide the folder to store the payload(s):`. Press enter to save the payload in the current folder. A file named `ebms.<timestamp>@127.0.1.1.Payload.0` will be downloaded, where *<timestamp>* is the time **ebms-send** was executed. Open that file and you will see the following content:

```
This is an sample message.
```



```
This is an sample message.
```

Configuration for Sending Secure Message

To send signed message through HTTPS, we have to configure a trust-store, keystore and certificate separately in Hermes and Tomcat. For details, please refer to the section *Send Messages Through HTTPS*.

FAQ

Hermes Deployment

Q1. The `corvus.log` shows:

```
hk.hku.cecid.piazza.commons.spa.PluginException: Error in processing␣
↪activation by handler:
hk.hku.cecid.ebms.spa.EbmsProcessor which is caused by java.io.IOException:␣
↪exception decrypting data - java.lang.SecurityException: Unsupported␣
↪keysize or algorithm parameters
```

A1. Please ensure the Java 2 SDK files have been replaced by the JCE files.

Q2. Some log files show the following error:

```
hk.hku.cecid.piazza.commons.dao.DAOException: Unable to begin transaction.
```

A2. Ensure PostgreSQL/MySQL/Oracle was installed properly and check the following files:

For AS2:

<HERMES2_HOME>/plugins/hk.hku.cecid.edi.as2/conf/hk/hku/cecid/edi/as2/conf/as2.module.core.xml. There is a tag in this file named parameter with the attribute name=url. Check the value attribute to see if it references the correct server address. The format of the value attribute is the same as the JDBC connection string.

For ebMS:

<HERMES2_HOME>/plugins/hk.hku.cecid.ebms/conf/hk/hku/cecid/ebms/spa/conf/ebms.module.xml. There is a tag in this file named parameter with the attribute name=url. Check the value attribute to see if it references the correct server address. The format of the value attribute is the same as the JDBC connection string.

Web Service Usage Sample

Q1. The following exception is thrown:

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: xxx␣
↪(Unsupported major.minor version 49.0)
```

A1. It is very likely you are using an incompatible Java version. The web service usage sample requires J2SE 5.0 or above to run properly. In the command prompt, enter java -version to check the Java version.

Q2. The following error occurs:

```
Sending ebMS/AS2 sending request ...
java.net.ConnectException: Connection refused: connect
```

A2. Check that the Application Container (Tomcat) has been started.

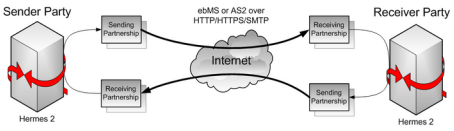
The First Step

Introduction

One of the most common difficulties in using Hermes for new developers/users is managing a partnership and programming the web service requester before they are able to deliver messages to their partner. This article is purposed to assist the user to create and maintain partnerships in Hermes. For more information on the installation and administration console of Hermes, please refer to *Installing Hermes*.

What is a partnership?

A partnership in Hermes is defined as a channel to your business partner. A partnership is a **simplex** (one-way) communication channel to your business partner. In a typical two-way business document exchange, the Hermes in each party should have **TWO** partnerships; one for sending, another for receiving. After the partnerships have been defined, the application can reference the partnerships to send or receive business messages.



Existing partnership types

Currently, Hermes has two types of partnerships, which are ebMS and AS2. They represent the communication agreement with your partner to use ebMS protocol and AS2 protocol respectively.

ebMS 2.0 partnership

What is an ebMS 2.0 partnership?

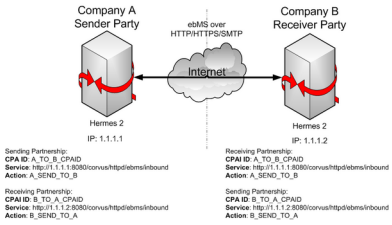
For an ebms 2.0 partnership, **CPA (Collaboration Protocol Agreement) ID**, **Service** and **Action** are used to uniquely identify each partnership. The values of these parameters will appear in the headers of outgoing ebMS messages. The purpose of these parameters for a sending partnership and a receiving partnership are different:

- In a sending partnership, these values are taken as the values in the ebMS headers of outgoing messages.
- In a receiving partnership, these values are taken as filtering criteria. If Hermes receives an ebMS message whose **CPA ID**, **Service** and **Action** values do not have a matching partnership, a negative acknowledgement will be sent back to the sender and no application can retrieve the message on the receiving side.

In Hermes, the **Service** parameter for sending and receiving partnerships has to be HTTP URL (Universal Resource Locator). The **Service** has a different meaning in each partnership:

- For a sending artnership, the **Service** acts as the endpoint URL of the sending Hermes for receiving acknowledgement.
- For a receiving partnership, since the **Service** is a filtering criterion for incoming messages, it should be the endpoint URL of the sending Hermes.

Let us look at a typical example below, in which two Hermes communicate using ebMS protocol:



In the above scenario, the Hermes in company A has an IP 1.1.1.1. Its **Service** value for receiving incoming acknowledgements is http://1.1.1.1:8080/corvus/httpd/ebms/inbound, where we suppose the application server is running on port `8080`. Similarly, the **Service** value for company B is http://1.1.1.2:8080/corvus/httpd/ebms/inbound.

Since the receiving partnership's **CPA ID**, **Service** and **Action** filter the incoming messages, the values in the receiving partnership of the receiving Hermes should always be the same as the values in the sending partnership of the sending Hermes.

How to create your first ebMS 2.0 partnership

1. Open a Web browser.
2. Go to Hermes Administration Console at `http://<LOCALHOST>:<PORT>/corvus/admin/home`.
3. Enter the administrator username and password for Hermes.
4. Click *Ebms Plugin* in the module tab located on the left hand side.
5. Click the *Partnership* tab on the central horizontal tabbed bar to display the partnership creation page.

6. To create the simplest ebMS 2.0 partnership, you are only required to fill in the following seven fields in the form:
   (a) Partnership ID
   (b) CPA ID
   (c) Service
   (d) Action
   (e) Transport URL
   (f) Retries
   (g) Retry Interval
7. Fill in **Partnership ID**, **CPA ID**, **Service** and **Action** as shown below: Partnership ID is the unique identifier of the partnership in the sending Hermes while the three remaining fields (CPA ID, Service and Action) form a composite identifier between sending and receiving systems.

Partnership ID	`MyFirstPartnership`
CPA ID	`MyFirstCPAID`
Service	http://localhost:8080/corvus/httpd/ebms/inbound
Action	`MyFirstAction`

8. Fill in the **Transport Endpoint** (the receiver's URL) as shown below:

   The Transport Endpoint URL should be formatted as `http://`*`<RECEIVER_HOST>`*`:`*`<PORT>`*`/corvus/httpd/ebms/inbound`

   where `corvus/httpd/ebms/inbound` is the context path for accepting and receiving incoming ebXML messages if the receiving system is also using Hermes.

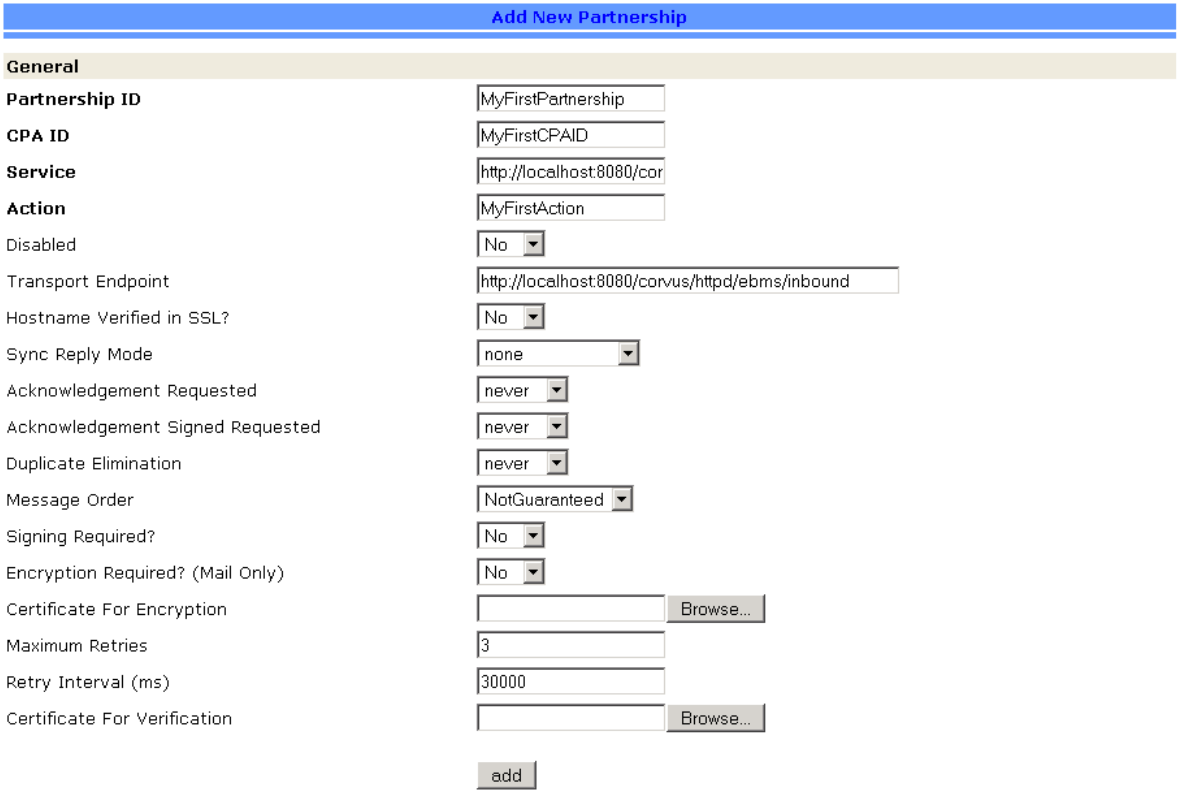
   Since the receiving host below is the same as the sending host (i.e. the ebMS message loops back to the sender), only **ONE** partnership is required for sending and receiving.

Transport Endpoint	http://localhost:8080/corvus/httpd/ebms/inbound

9. Fill in the number of retries allowed if the message fails to be delivered and the retry interval as shown below:

Retries	`3`
Retry Interval (ms)	`30000`

10. Now you have completed all required fields and you should have the same input as the figure shown here.



11. Click the *add* button at the bottom of the page.
12. A dialog box will prompt you to confirm adding the partnership. Click *Ok*.
13. The message *Partnership Added Successfully* will be shown on the status bar (the bottom of the page).
14. Congratulations! You have successfully created your first ebMS 2.0 partnership.

How to update an ebMS 2.0 partnership

1. Do **steps 1-5** in *How to create your first ebMS 2.0 partnership* or all steps if you have not registered a partnership in Hermes.
2. You should able to see a drop-down list under the header *Registered Partnership*.
3. Click the *Change* button.
4. Now you should able to see a module called *Selected Partnership* that contains the information of selected partnership from the previous step like here.

5. Change the desired parameters/fields and click *Update* when you are done.
6. A dialog box will prompt you to confirm the updates. Click *Ok*.
7. The message *Partnership Updated Successfully* will be shown on the status bar (the bottom of the page).

How to delete an ebMS 2.0 partnership

1. Do the **steps 1-4** in *How to update an ebMS 2.0 partnership*.
2. Click the *Delete* button
3. The message *Partnership deleted successfully* will be shown on the status bar (the bottom of the page).

AS2 partnership

What is an AS2 partnership?

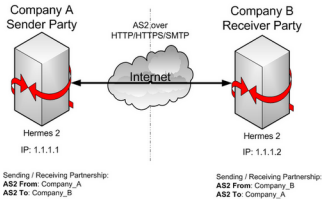
For an AS2 partnership, the **AS2 From** and **AS2 To** fields in a partnership are used to uniquely identify it. The values of these parameters will appear in AS2 message headers. The purpose of these parameters for a sending partnership and a receiving partnership are different:

- In a sending partnership, these values are taken as the values in the headers of outgoing messages.
- In a receiving partnership, these values are taken as filtering criteria. If Hermes receives an AS2 message whose **AS2 From** and **AS2 To** values do not have a matching partnership, the incoming message is rejected/ignored and no application can retrieve the message on the receiving side.

Note: The values of **AS2 From** and **AS2 To** in the incoming AS2 message are interchanged before finding the matching partnership (i.e. the filtering criteria **[AS2 From, AS2 To] = [Incoming AS2 To, Incoming AS2 From]** in the AS2 message).

Although the values of **AS2 From** and **AS2 To** have no constraints, it is highly recommended that they be company specific, such as Data Universal Numbering System (DUNS) numbers, or simply identification strings agreed upon between trading partners.

Let us look at a typical example below, in which two Hermes communicate using AS2 protocol:



How to create your first AS2 partnership

1. Open a Web browser.
2. Go to Hermes Administration Console at `http://<LOCALHOST>:<PORT>/corvus/admin/home`.
3. Enter the administrator user name and password for Hermes.
4. Click the *AS2 Plugin* in the module tab located on the left hand side.
5. Click the *Partnership* tab on the central horizontal tabbed bar to display the partnership creation page.
6. To create the simplest AS2 partnership, you are only required to fill in the following seven fields in the form:
   (a) Partnership ID
   (b) AS2 From
   (c) AS2 To
   (d) Subject
   (e) Transport URL
   (f) Retries
   (g) Retry Interval
7. Fill in **Partnership ID**, **AS2 From** and **AS2 To** as shown below: Partnership ID is the unique identifier of the partnership in the sender Hermes while the two remaining fields (AS2_From, AS2_To) form a composite identifier between sending and receiving systems.

Partnership ID	`MyFirstPartnership`
AS2 From	`FromMyMachine`
AS2 To	`ToMyMachine`

8. Fill in the **Subject** and **Transport Endpoint** (the receiver's URL) as shown below:

   The Transport Endpoint URL should be formatted as `http://<RECEIVER_HOST>:<PORT>/corvus/httpd/as2/inbound`

   where `corvus/httpd/as2/inbound` is the context path for accepting and receiving incoming AS2 messages if the receiving system is also using Hermes.

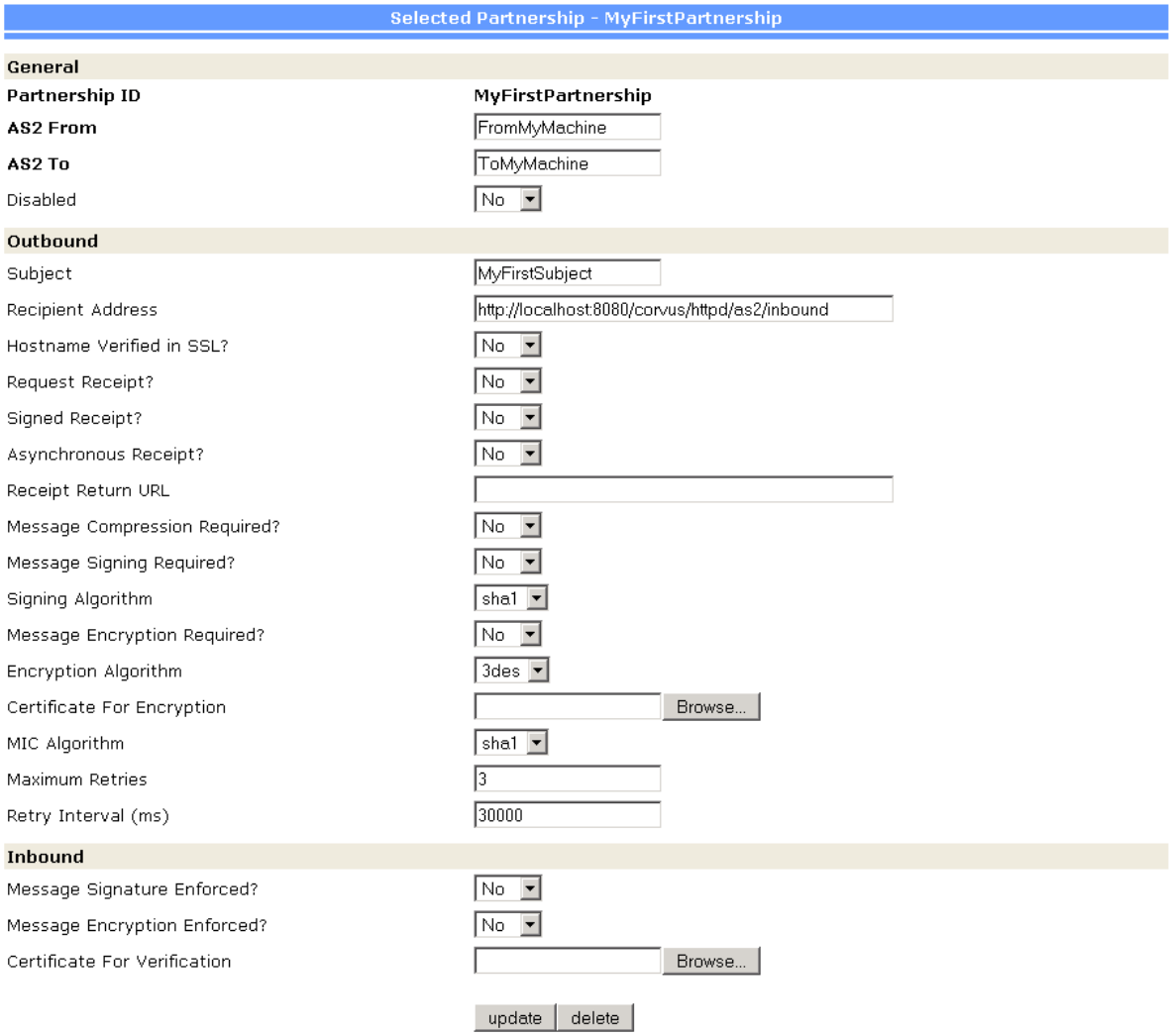
Since the receiving host below is the same as the sending host (i.e. the AS2 message loops back to the sender), only **ONE** partnership is required for sending and receiving.

Subject	MyFirstSubject
Transport Endpoint	http://localhost:8080/corvus/httpd/as2/inbound

9. Fill in the number of retries allowed if the message fails to be delivered and the retry interval as shown below:

Retries	3
Retry Interval (ms)	30000

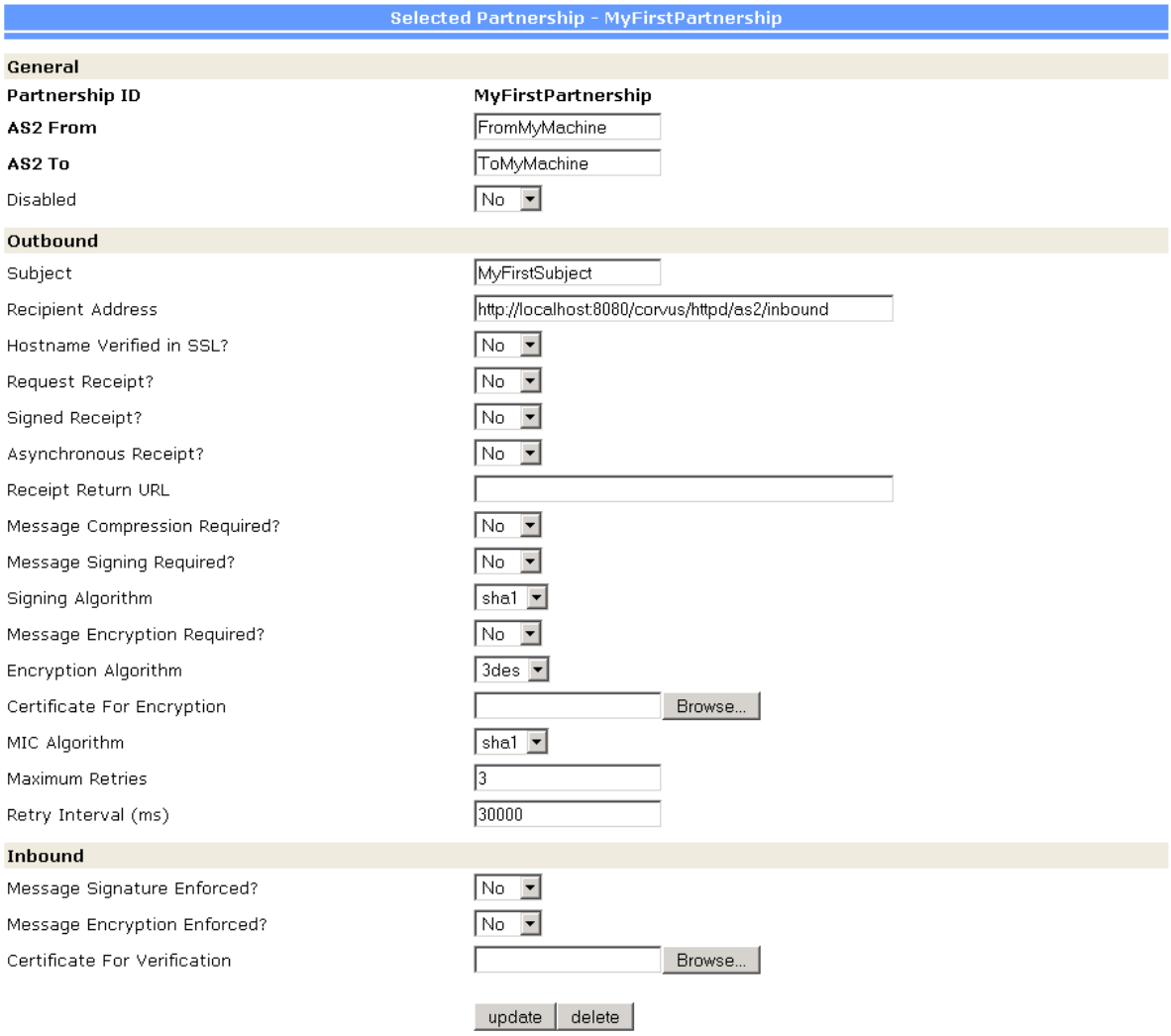
10. Now you have completed to all required fields and you should have the same input as the figure shown here.



11. Click the *add* button at the bottom of the page.
12. A dialog box will prompt you to confirm adding the partnership. Click *Ok*.
13. The message *Partnership Added Successfully* will be shown on the status bar (the bottom of the page).
14. Congratulations! You have successfully created your first AS2 partnership.

How to update an AS2 partnership

1. Do the **steps 1-5** in *How to create your first AS2 partnership* or all steps if you have not registered a partnership in Hermes.
2. You should able to see a drop-down list under the header *Registered Partnership*.
3. Click the *Change* button.
4. Now you should able to see a module called *Selected Partnership* that contains the information of selected partnership from previous step like here.



5. Change the desired parameters/fields and click *Update* when you are done.
6. A dialog box will prompt you to confirm the update. Click *Ok*.
7. The message *Partnership Updated Successfully* will be shown on the status bar (the bottom of the page).

How to delete AS2 partnership

1. Do the **steps 1-4** in *How to update an AS2 partnership*.
2. Click the *Delete* button.

3. The message *Partnership deleted successfully* will be shown on the status bar (the bottom of the page).

Conclusion

The main benefit of partnerships is that it provides abstraction on technical parameters. The abstraction is beneficial because:

1. The application does not need to change if your business partner changes the parameters, since all technical parameters are contained within the partnership.
2. The application only needs to submit payloads. It does not contain any code that is specific to the communication protocol between messaging gateways.
3. The application does not need to handle the raw and cryptic ebMS / AS2 messages. Therefore, developers only need to focus on business logic and integration with the backend systems.

See also

- *Setting Up ebMS 2.0 Partnerships*
- *Setting Up AS2 Partnerships*
- OASIS ebMS 2.0 Specification
- AS2 Specification

Setting Up ebMS 2.0 Partnerships

Partnership ID

Description	The unique identifier of an ebMS 2.0 partnership in local Hermes. The value of this field has no restriction but it is **RECOMMENDED** to be unique between sender and recipient. This field is **mandatory** and its maximum length is 255.

CPA ID

Description	The Collaboration Protocol Agreement (CPA) ID is a string that identifies the parameters governing the exchange of messages between the parties. The recipient of a message must be able to resolve the CPA ID to an individual set of parameters, taking into account the sender of the message. Simply put, it is an arbitary and **mandatory** string that can be used to identify both sender and recipient. See [OASIS ebXML Messaging Service Spec v2.0] for details.

Service

Description	In Hermes, this **mandatory** parameter is used for mapping a partnership between **sender** and **recipient**.

Action

Description	This identifies a process within a *Service* that processes the ebMS message. Action must be unique within the Service in which it is defined. The value of the Action element is specified by the designer of the Service. This field is **mandatory** and its maximum length is 255 characters.

Disabled

Description	This boolean option indicates whether the partnership is disabled or not. Disabled partnership do not deliver/receive any outgoing/incoming messages.
Options	[`true` = disabled], [`false` = enabled]

Transport Endpoint

Description	The endpoint URL of the receiving messaging gateway. If the receiving messaging gateway is Hermes and HTTP/HTTPS is the transport protocol, the endpoint URL is formatted as `http://<RECIPIENT_HOST>:<PORT>/corvus/httpd/ebms/inbound`. The recipient should set this field to `http://<SENDER_HOST>:<PORT>/corvus/httpd/ebms/inbound` in order to send acknowledgements upon request. Otherwise, if the recipient host is an SMTP gateway, the endpoint URL is formatted as `mailto:<EMAIL ADDRESS>`. This field is **mandatory** and the format must be an **HTTP/HTTPS URL** or **EMAIL ADDRESS**.

Hostname Verified in SSL?

Description	This boolean flag indicates whether HTTP SSL/TLS protocol is used to verify the recipient hostname. This is relevant only if **HTTPS** transport protocol is used in *Transport Endpoint*
Options	[`true` = hostname verified using SSL , `false` = no verification using SSL]

Sync Reply Mode

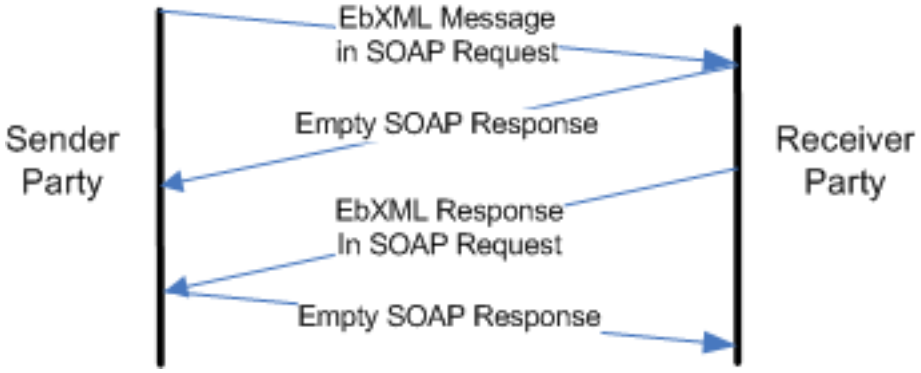
Description	Indicates whether the receiver should reply to incoming ebMS messages using the same HTTP/HTTPS connection that the sender is using for delivery. This parameter is only applicable to send partnerships using HTTP/HTTPS transport protocol.
Options	[`mshSignalsOnly` = synchronous reply], [`none` = asynchronous reply]

Synchronous reply

ebMS message acknowledgement is included in the HTTP/SOAP response.

Asynchronous reply

ebMS message acknowledgement will be delivered through another HTTP/SOAP connection from the recipient to the sender.



Acknowledgement Requested

Description	Indicates whether the sender has requested the recipient to reply with an ebMS acknowledgement. An acknowledgement is a type of ebMS message which has an `<acknowledgement>` element. For interoperability of this feature, both sender and recipient must enable it. Otherwise, the recipient will return a negative acknowledgement. How the acknowledgement is sent depends on the value of *Sync Reply Mode*. If it is enabled, the acknowledgement will be sent immediately using the same HTTP connection as the received message. Otherwise, if the recipient is using Hermes, the acknowledgement will be placed in an outgoing queue until it is delivered to the sender. It is **RECOMMENDED** to set this parameter to **always** for reliable messaging.
Options	[`always` = acknowledgement requested], [`none` = acknowledgement is not requested]

Acknowledgement Signed Requested

Description	Indicates whether the recipient must sign the ebMS acknowledgement digitally using their private key before delivering it to the sender. For interoperability of this feature, both sender and recipient must enable it. Otherwise, the recipient will return a negative acknowledgement. The format of the private key should be in PKCS12 and the created signatures should conform to W3C XML Signatures Specification [XMLDsig]. The send partnership must set *Acknowledgement Requested* to `always` for this feature to run properly. The recipient is required to provide a Certificate for Verification so the signature in the acknowledgement can be verified.
Dependencies	[*Acknowledgement Requested* = `always`], [*Certificate For Verification* **REQUIRED**]
Options	[`true` = acknowledgement must be digitally signed], [`false` = acknolwedgment must not be digitally signed]

Duplicate Elimination

Description	Indicates whether the recipient will ignore duplicate messages. For interoperability of this feature, both sender and recipient must enable it. Otherwise, the recipient will return a negative acknowledgement.
Options	[`always` = ignores duplicate messages], [`never` = receives duplicate messages]

Message Order

Description	Indicates whether the recipient must receive ebMS messages in the same sequence that they were sent. For interoperability of this feature, both sender and recipient must enable it. Otherwise, the recipient will return a negative acknowledgement.
Dependencies	[*Sync Reply Mode* = `none`], [*Acknowledgement Requested* = `always`], [*Duplicate Elimination* = `always`]
Options	[`Guaranteed` = recipient receives ebMS messages in sending order], [`NotGuaranteed` = recipient receives ebMS message with best effort behavior]

Signing Required?

Description	Indicates whether the sender must sign ebMS messages digitally using their private key. For interoperability of this feature, both sender and recipient must enable this. Otherwise, the recipient will return a negative acknowledgement. The format of the private key should be in PKCS12 and the created signature should conform to W3C XML Signatures Specification [XMLDsig]. For details of signing message, please refer to *Sign and Verify Message*.
Options	[`true` = outgoing ebMS messages must be digitally signed], [`false` = outgoing ebMS messages are not required to be digitally signed]

Encryption Required? (Mail Only)

Description	Indicates whether the sender must encrypt ebMS messages using the recipient's public certificate defined in Certificate For Encryption. This is only applicable when using **SMTP** protocol for Transport Endpoint. The encryption method is based on S/MIME standard.
Dependencies	[*Transport Endpoint* = using SMTP protocol], [*Sync Reply Mode* = `none`], [*Certificate For Encryption* **REQUIRED**]
Options	[`true` = outgoing ebMS messages must be encrypted], [`false` = outgoing ebMS messages are not required to be encrypted]

Certificate For Encryption

Description	The certificate file for encrypting outgoing ebMS messages using SMTP protocol by using the public key generated by the recipient. The recipient should use the keystore in the ebMS plugin to export the public certificate for the sender. ebMS default keystore location: *<HERMES2_HOME>*/`plugins/corvus-ebms/security` The certificate must be in X.509 format. See *Encryption Required? (Mail Only)* for details.

Maximum Retries

Description	The maximum number of retries allowed for the sender to attempt delivering an ebMS message. Hermes tries to deliver the ebMS message under the features of reliable messaging until exceeding the maximum number of retries. There will be a time interval between each attempt, which is defined in *Retry Interval (ms)*. It is **recommended** that the value of this field be between `1-10`.
Dependencies	[*Acknowledgement Requested* = `always`]

Retry Interval (ms)

Description	The time interval (milleseconds) between each consecutive attempt to deliver an ebMS message. It is **recommended** that the value of this field be between `30000-300000`.
Dependencies	[*Acknowledgement Requested* = `always`]

Certificate For Verification

Description	The certificate (`.cer`) file for verifying incoming digitally signed ebMS message by using the public key generated by sender. The sender should use the keystore in the ebMS plugin to export the public certificate for the recipient. ebMS default keystore location: *<HERMES2_HOME>*/`plugins/corvus-ebms/security` The keystore must be in PKCS12 format. See *Signing Required?* for details.

Setting Up AS2 Partnerships

Partnership ID

Description	The unique identifier of an AS2 partnership in Hermes. The value of this field has no restrictions but it is **RECOMMENDED** to be unique between sender and recipient. This field is **mandatory** and its maximum length is 255 characters.

AS2 From

Description	Identifier of the sending party in a data exchange. The values may be company specific, such as Data Universal Numbering System (DUNS) numbers, or they may simply be identification strings agreed upon between trading partners. [AS2 RFC4130 6.2] This parameter is used as the `AS2-From` property in AS2 message headers in this partnership. This field is **mandatory** and it is **RECOMMENDED** that the length of this value be less than 255 characters. See note below.

AS2 To

Description	Identifier of the receiving party in a data exchange. The values may be company specific, such as Data Universal Numbering System (DUNS) numbers, or they may simply be identification strings agreed upon between trading partners. [AS2 RFC4130 6.2] This parameter is used as the `AS2-To` property in AS2 message headers in this partnership. This field is **mandatory** and it is **RECOMMENDED** that the length of this value be less than 255 characters. See note below.

Note: *AS2 From* and *AS2 To* form a pair that identify the **send** and **recieve** partnerships (i.e. they form a composite key that identifies the parties involved in the data exchange).

The values of [`AS2 From`, `AS2 To`] are reversed in the **receive** partnership with respect to the **send** partnership. For example:

Send: [`CompanyA`, `CompanyB`] –> **Receive**: [`CompanyB`, `CompanyA`].

Disabled

Description	This boolean option indicates whether the partnership is disabled or not. Disabled partnerships do not deliver/receive any outgoing/incoming messages.
Options	[`true` = disabled], [`false` = enabled]

Subject

Description	The subject of the partnership. This parameter is used as the `Subject` property in AS2 message headers in this partnership. This field is only applicable to **send** partnerships.

Recipient Address

Description	The endpoint URL of the receiving messaging gateway. If the receiving messaging gateway is Hermes, the endpoint URL is `http://<RECIPIENT_HOST>:<PORT>/corvus/httpd/as2/inbound`. This field is **mandatory** and it must be an **HTTP/HTTPS URL**.

Hostname Verified in SSL?

Description	This boolean flag indicates whether HTTP SSL/TLS protocol is used to verify the recipient hostname. This is relevant only if **HTTPS** transport protocol is used in *Recipient Address*. This field is only applicable to **send** partnerships.
Options	[`true` = hostname verified using SSL], [`false` = no verification using SSL]

Request Receipt

Description	Indicates whether the sender has requested the recipient to reply with an AS2 receipt (acknowledgement). How the receipt is sent depends on the value of *Asynchronous Receipt*. If it is disabled, the receipt will be sent immediately using the same HTTP connection as the received message. Otherwise, if the recipient is using Hermes, the receipt will be placed in an outgoing queue until it is delivered to the sender. It is **RECOMMENDED** to set this parameter to `true` for reliable messaging. This field is only applicable to **send** partnerships.
Options	[`true` = receipt requested], [`false` = receipt is not requested]

Signed Receipt

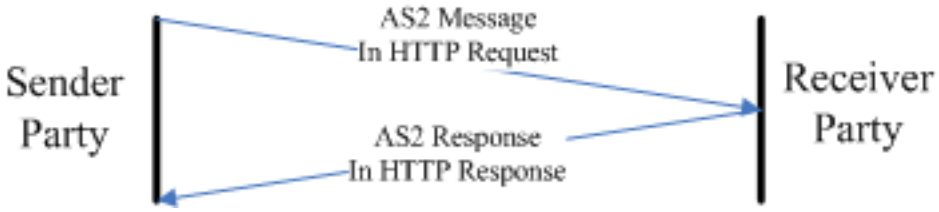
Description	Indicates whether the sender has requested the recipient to digitally sign the AS2 receipt with their private key before delivering it. The format of the private key should be in PKCS12 and the created signatures should conform to IETF S/MIME. The send partnership must enable Request Receipt for this feature to function properly. The recipient is required to provide a Certificate for Verification so the source of the receipt can be verified. This field is only applicable to **send** partnerships.
Dependencies	[*Request Receipt* = true] , [*Certificate for Verification* **REQUIRED**]
Options	[`true` = receipt must be digitally signed], [`false` = receipt must not be digitally signed]

Asynchronous Receipt

Description	Indicates whether the recipient should reply to incoming AS2 messages using the same HTTP/HTTPS connection that the sender is using for delivery. This field is only applicable to **send** partnerships.
Dependencies	[*Request Receipt* = `true`]
Options	[`true` = asynchronous reply], [`false` = synchronous reply]

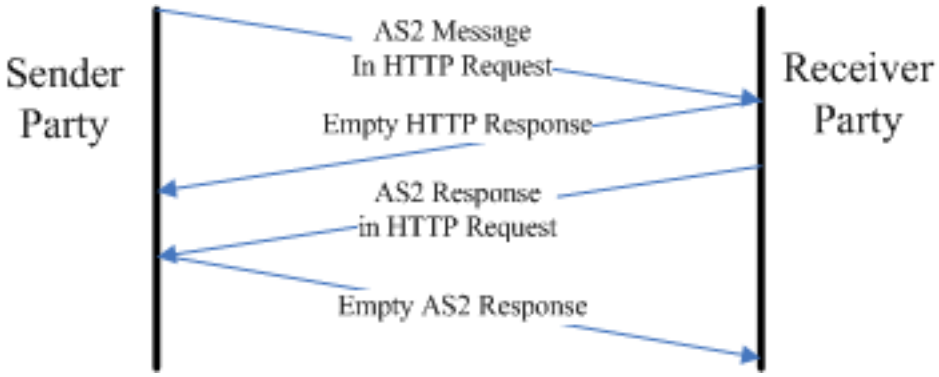
Synchronous reply

AS2 message receipts are encapsulated in the HTTP response.



Asynchronous reply

AS2 message receipts will be delivered through another HTTP connection from recipient to sender.



Receipt Return URL

Description	This is the endpoint URL of Hermes or another compatible messaging gateway for receiving receipts. It is always the inbound endpoint URL of the **send** partnership. For example: **Sender (A)** IP address: `1.1.1.1:8080` **Recipient (B)** IP address: `1.1.1.2:8080` AS2 inbound endpoint : `/corvus/httpd/as2/inbound` Then the Receipt Return URL for an AS2 message from **sender (A)** is the inbound endpoint of **sender (A)**, which is `http://1.1.1.1:8080/corvus/httpd/as2/inbound/` This field is only applicable to **send** partnerships.
Dependencies	[*Request Receipt* = `true`], [*Asynchronous Receipt* = `true`]

Message Compression Required

Description	Indicates whether the sender must compress outgoing AS2 messages in this partnership. This field is only applicable for **send** partnerships.
Options	[`true` = outgoing AS2 messages must be compressed], [`false` = outgoing AS2 messages must not be compressed]

Message Signing Required

Description	Indicates whether the sender must digitally sign AS2 messages using their private key. This field is only applicable for **send** partnerships.
Options	[`true` = outgoing AS2 messages must be digitally signed], [`false` = outgoing AS2 messages must not be digitally signed]

Signing Algorithm

Description	The algorithm used to digitally sign outgoing AS2 messages in this partnership.
Options	[`SHA1`], [`MD5`]

Message Encryption Required

Description	Indicates whether the sender must encrypt AS2 messages using the recipient's public certificate defined in Certificate for Encryption. The encryption method is based on the S/MIME standard. This field is only applicable for **send** partnerships.
Dependencies	[*Certificate for Encryption* **REQUIRED**]
Options	[`true` = outgoing AS2 messages must be encrypted], [`false` = outgoing AS2 messages must not be encrypted]

Encryption Algorithm

Description	The algorithm used to encrypt outgoing AS2 messages in this partnership.
Options	[`3DES`], [`RC2`]

Certificate for Encryption

Description	The certificate (`.cer`) file for encrypting outgoing AS2 messages using the public key exported by the recipient. The recipient should use the keystore in the AS2 plugin to export the public certificate for the sender. AS2 default keystore location: *<HERMES2_HOME>*`/plugins/corvus-as2/security` The keystore must be in PKCS12 format. See *Message Encryption Required* for details.

MIC Algorithm

Description	The algorithm used to create message digests/hashes for outgoing AS2 messages in this partnership.
Options	[`SHA1`], [`MD5`]

Maximum Retries

Description	The maximum number of retries allowed for the sender to attempt delivering an AS2 message. Hermes tries to deliver the AS2 message under the specification of reliable messaging until exceeding the maximum number of retries. There will be a time interval between each attempt, which is defined in *Retry Interval (ms)*. It is **RECOMMENDED** that the value of this field be between `1-10`.

Retry Interval (ms)

Description	The time interval (milleseconds) between each consecutive attempt to deliver an AS2 message. It is **RECOMMENDED** that the value of this field be between `30000-300000`.

Message Signature Enforced

Description	Indicates whether incoming AS2 messages must be digitally signed. If enabled, AS2 messages in this partnership must be digitally signed by the sender before the message is received by the recipient. This field is only applicable to **recieve** partnerships.
Options	[`true` = incoming AS2 messages must be digitally signed], [`false` = incoming As2 messages may not be digitally signed]

Message Encryption Enforced

Description	Indicates whether incoming AS2 messages must be encrypted. It enforced, AS2 message in this partnership must be encrypted by the sender before the message is received by the recipient. This field is only applicable to **recieve** partnerships.
Options	[`true` = incoming AS2 messages must be encrypted], [`false` = incoming AS2 messages may not be encrypted]

Certificate for Verification

Description	The certificate (`.cer`) file for verifying incoming digitally signed AS2 messages using the public key generated by the sender. The sender should use the keystore in the AS2 plugin to export the public certificate for the recipient. AS2 default keystore location: *<HERMES2_HOME>*`/plugins/corvus-as2/security` The keystore must be in PKCS12 format. See *Message Signing Required* for details.

Configuring Hermes

Introduction

This document acts as a configuration reference for Hermes. The configurations of Hermes are specified by various XML-based configuration files. By modifying these files, administrators or developers can configure all the settings such as the location of the message database and log file locations.

The intended audience of this document includes system administrators, application developers and plugin developers of the Hermes system. It assumes the audience has some background knowledge of the following:

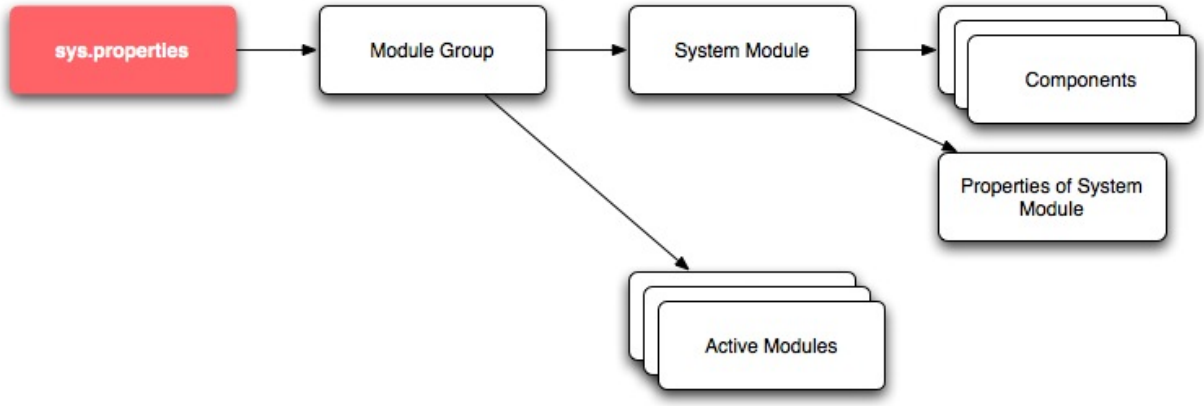
- Java Standard Edition
- XML
- AS2
- ebXML Messaging Services
- Public Key Infrastructure
- Application server compliant to Servlet 2.x specification
- Databases

Overview on loading property files

Hermes has employed a module-group-component architecture where you can define your own module for each application. You can then assign a property file for each component and the Hermes Core System will load them.

There are two loading mechanisms, one for the core system and one for the plugins. The two are almost identical except for their initial definitions.

Let's take a look at how the core system modules are loaded.



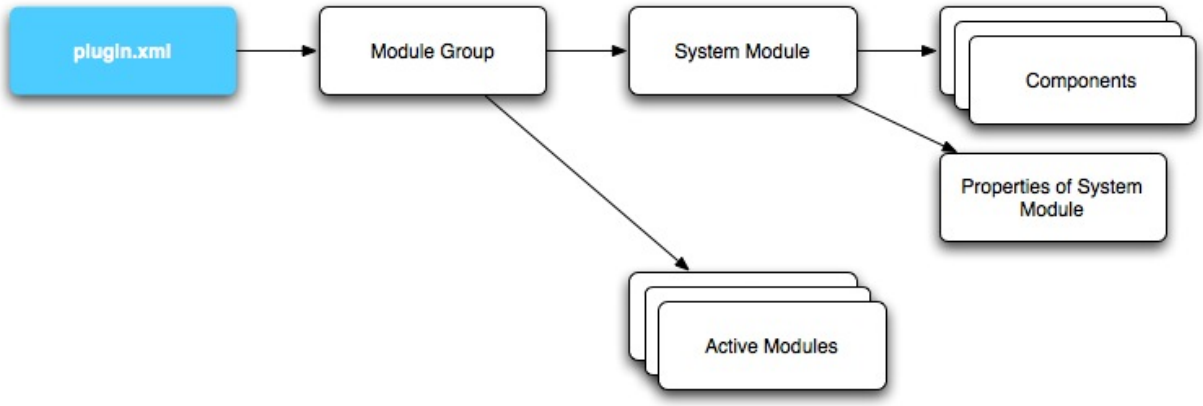
For the core system, Hermes will look for the existence of a file named `sys.properties` from the classpath which contains the location of the module-group definition file. E.g.,

```
sys.module.group=doc-processor.module-group.xml
```

From there, the system will look for the specified XML file and load up the modules specified within. The modules could be System Modules, which contain infrastructure components such as logging and database access, or they could be Active Modules, which perform business functions such as sending messages.

```
<module id="piazza.corvus" name="Piazza Corvus" version="1.0">
  <component id="logger" name="System Logger">
    <class>hk.hku.cecid.piazza.commons.util.LoggerLog4j</class>
    <parameter name="config"
               value="hk/hku/cecid/piazza/corvus/core/conf/corvus.log.properties.xml"␣
↪/>
    <parameter name="category" value="hk.hku.cecid.piazza" />
  </component>
...
</module>
```

For the plugins, instead of looking for a file named `sys.properties`, Hermes will look for a file named `plugin.xml` instead.



From within, a parameter with the value `module-group-descriptor` will define the location of the module-group definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="hk.hku.cecid.edi.as2"
  name="Corvus AS2 Plugin"
  version="1.0.1"
  provider-name="hk.hku.cecid"
  class="hk.hku.cecid.edi.as2.AS2Processor"
>
  <parameters>
    <parameter name="module-group-descriptor"
               value="hk/hku/cecid/edi/as2/conf/as2.module-group.xml"/>
  </parameters>
...
</plugin>
```

Hermes core system properties

You can modify the following information either through the web admin interface or by manipulating the XML configuration files directly.

The configuration files are stored in *<WEBAPPS_LOCATION>*`/corvus/WEB-INF/classes/hk/hku/cecid/piazza/corvus/core/conf` (where *<WEBAPPS_LOCATION>* is the web application repository of the application server).

Properties	Configuration file
1. Hermes location 2. Plugin location for Hermes 3. SSL trust store information 4. HTTP/HTTPS proxy server 5. Encoding settings for core system 6. Connection timeout settings	`corvus.properties.xml`
7. Log file location and level of logging	`corvus.log.properties.xml`

Hermes location

You can change the location of Hermes by modifying this element:

```
<corvus>
  <home>/corvus</home>
  ...
</corvus>
```

XPath	Expected information
`/corvus/home`	The location in which Hermes is installed. Note that the specified path is an absolute path.

Plugin location for Hermes

You can change the plugin location of Hermes by modifying this element:

```
<corvus>
  ...
  <plugin>
    ...
    <registry>/corvus/plugins</registry>
    <descriptor>plugin.xml</descriptor>
    ...
  </plugin>
  ...
</corvus>
```

XPath	Expected information
`/corvus/plugin/registry`	The location in which Hermes plugins are installed. By default, it should be the `plugins` directory under the home directory where Hermes is installed. Note that the specified path is an absolute path.
`/corvus/plugin/descriptor`	The name of the XML file which Hermes will use when loading the module-group-component.

SSL trust store information

```
<corvus>
  ...
    <environment>
      <properties>
```

```
        ...
      <javax.net.ssl.trustStore>/j2sdk1.4.2_04/jre/lib/security/cacerts
      </javax.net.ssl.trustStore>
      <javax.net.ssl.trustStorePassword>password
      </javax.net.ssl.trustStorePassword>
      ...
    </properties>
  <environment>
</corvus>
```

XPath	Expected information
/corvus/environment/properties/javax.net.ssl.trustStore	The location of the Java keystore which is used for establishing SSL connections. The keystore should contain the certificates of trusted certificate authorities. To maintain the keystore, the reader should use the **keytool** provided by **JDK**. For more information, the reader may reference http://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html.
/corvus/environment/properties/javax.net.ssl.trustStorePassword	The password used to access the keystore specified above.

HTTP/HTTPS proxy server

```
<corvus>
  ...
    <environment>
      <properties>
        <http.proxyHost>proxy.csis.hku.hk</http.proxyHost>
        <http.proxyPort>8282</http.proxyPort>
        <https.proxyHost>proxy.csis.hku.hk</https.proxyHost>
        <https.proxyPort>8282</https.proxyPort>
        ...
      </properties>
    <environment>
</corvus>
```

XPath	Expected information
/corvus/environment/properties/http.proxyHost	The hostname or IP address of the proxy host that Hermes will establish HTTP connections with for outgoing messages.
/corvus/environment/properties/http.proxyPort	The TCP port of the proxy server specified above.
/corvus/environment/properties/https.proxyHost	The hostname or IP address of the proxy host that Hermes will establish HTTPS connections with for outgoing messages.
/corvus/environment/properties/https.proxyPort	The TCP port of the proxy server specified above.

Encoding settings for core system

```
<corvus>
  ...
    <encoding>
      <servlet-request>UTF-8</servlet-request>
      <servlet-response>text/html;UTF-8</servlet-response>
```

```
    </encoding>
  ...
</corvus>
```

XPath	Expected information
/corvus/encoding/ servlet-request	The encoding of incoming HTTP or HTTPS requests. UTF-8 is the recommended value because it can handle most written languages.
/corvus/encoding/ servlet-response	The encoding of outgoing HTTP or HTTP responses.

Connection timeout settings

```
<corvus>
    ...
  <properties>
    ...
    <sun.net.client.defaultConnectTimeout>30000</sun.net.client.defaultConnectTimeout>
    <sun.net.client.defaultReadTimeout>300000</sun.net.client.defaultReadTimeout>
    ...
  </properties>
  ...
</corvus>
```

XPath	Expected information
/corvus/properies/sun. net.client. defaultConnectTimeout	It specifies the timeout (in milliseconds) for establishing HTTP or HTTPS connections for outgoing messages. 30 seconds is the recommended value.
/corvus/properies/sun. net.client. defaultReadTimeout	It specifies the timeout (in milliseconds) for reading from input streams when a HTTP or HTTPS connection is established. 300 seconds is the recommended value.

Log file location and level of logging

To change the settings of the log written by the core system, you will need to modify the XML file named corvus.log.properties.xml. Configuring the logging module is the same as configuring Apache Log4j. Note that for configuring the logs of plugins, you need to edit another configuration file.

```
<log4j:configuration debug="null" threshold="null" xmlns:log4j="http://jakarta.apache.
↪org/log4j/">
<appender name="corvus" class="org.apache.log4j.RollingFileAppender">
  <param name="File" value="/corvus/corvus.log"/>
  <param name="Encoding" value="UTF-8"/>
  <param name="MaxFileSize" value="100KB"/>
  <param name="MaxBackupIndex" value="1"/>
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} [%-12.12t] &lt;%-
↪5p&gt; &lt;%m&gt;%n"/>
  </layout>
</appender>
<category additivity="true" name="hk.hku.cecid.piazza">
  <priority value="debug"/>
  <appender-ref ref="corvus"/>
</category>
</log4j:configuration>
```

XPath	Expected information
/log4j:configurationcategory/priority	The log level of core system logging. The available levels are debug, info, warn, error and fatal. If you set the value as debug, all logs will be printed.
/log4j:configurationcategory/appender-ref@ref	The name of the appender element to be used for logging. The appender element specifies how to generate log files. In the above example, the appender named corvus is used. The settings of the appender are specified by the referenced appender element.
/log4j:configurationappender@class	The appender specified by the appender configuration element. Apache Log4j provides a series of appenders, such as RollingFileAppender and DailyRollingFileAppender.
/log4j:configurationappender@name	The name of the appender configuration element. /log4j:configurationcategory/appender-ref@ref should reference the appender configuration element by this name.
/log4j:configurationappender/param[@name='File']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is File)	The path of the core system log from this appender.
/log4j:configurationappender/param[@name='Encoding']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is Encoding)	The encoding to be used for the log file.
/log4j:configurationappender/param[@name='MaxFileSize']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is MaxFileSize)	If the size of a log file has grown to exceed this limit, a new log file will be written and the old log file will be backed up. An index will be appended to the name of the old log file. (e.g. corvus.log.1).
/log4j:configurationappender/param[@name='MaxBackupIndex']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is MaxBackupIndex)	The maximum number of log files that will be backed up. For example, if it is set to 10, the maximum number of backed up log files will be 10 and their filenames will be *xxx*.log.1, *xxx*.log.2, ..., *xxx*.log.10.
/log4j:configurationappender/layout/param[@name='ConversionPattern']/@value	The pattern used when writing the log file.

Hermes plugin properties

AS2 plugin

In the directory *<HERMES_2_PLUGINS_LOCATION>*/corvus-as2/conf/hk/hku/cecid/edi/as2/conf, there are some configuration files for Hermes's AS2 plugin. Which configuration file you should edit depends on the property:

Properties	Configuration file
Log file location and level of logging	`as2.log.properties.xml`
Connection to message database	`as2.module.core.xml`
Location of keystore for signing outgoing messages	
Location of message repository	

Log file location and level of logging

To change the location of the log file, you will need to modify the XML file named `as2.log.properties.xml`.

```
<log4j:configuration debug="null" threshold="null" xmlns:log4j="http://jakarta.apache.
↪org/log4j:configuration">
<appender name="as2" class="org.apache.log4j.RollingFileAppender">
  <param name="File" value="/as2.log"/>
  <param name="Encoding" value="UTF-8"/>
  <param name="MaxFileSize" value="100KB"/>
  <param name="MaxBackupIndex" value="1"/>
  <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern"
        value="%d{yyyy-MM-dd HH:mm:ss} [%-12.12t] &lt;%-5p&gt; &lt;%m&gt;%n"/>
  </layout>
</appender>
<category additivity="true" name="hk.hku.cecid.edi.as2">
  <priority value="debug"/>
  <appender-ref ref="as2"/>
</category>
</log4j:configuration>
```

XPath	Expected information
/log4j:configuration/category/priority	The log level of the AS2 plugin logging. The available levels are debug, info, warn, error and fatal. If you set the value as debug, all logs will be printed.
/log4j:configuration/category/appender-ref@ref	The name of the appender element to be used for logging. The appender element specifies how to generate log files. In the above example, the appender named as2 is used. The settings of the appender are specified by the referenced appender element.
/log4j:configuration/appender@class	The appender specified by this appender configuration element. Apache Log4j provides a series of appenders, such as RollingFileAppender and DailyRollingFileAppender.
/log4j:configuration/appender@name	The name of this appender configuration element. /log4j:configuration/category/appender-ref@ref should reference the appender configuration element by this name.
/log4j:configuration/appender/param[@name='File']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is File)	The path of the AS2 log of this appender.
/log4j:configuration/appender/param[@name='Encoding']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is Encoding)	The encoding to be used for the log file.
/log4j:configuration/appender/param[@name='MaxFileSize']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is MaxFileSize)	If the size of a log file has grown to exceed this limit, another new log file will be written and the old log file will be backed up. An index will be appended to the name of the old log file (e.g. as2.log.1).
/log4j:configuration/appender/param[@name='MaxBackupIndex]/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is MaxBackupIndex)	The maximum number of log files that will be backed up. For example, if it is set to 10, the maximum number of backed up log files will be 10 and their filenames will be *xxx*.log.1, *xxx*.log.2, ..., *xxx*.log.10.
/log4j:configuration/appender/layout/param[@name='ConversionPattern']/@value	The pattern used when writing the log file.

Connection to message database

```
<module>
...
<component id="daofactory" name="AS2 DAO Factory">
  <class>
  hk.hku.cecid.piazza.commons.dao.ds.SimpleDSDAOFactory
  </class>
  <parameter name="driver" value="org.postgresql.Driver" />
  <parameter name="url"
             value="jdbc:postgresql://localhost:5432/as2" />
  <parameter name="username" value="corvus" />
```

```
  <parameter name="password" value="corvus" />
  <parameter name="pooling" value="true" />
  <parameter name="maxActive" value="20" />
  <parameter name="maxIdle" value="10" />
  <parameter name="maxWait" value="-1" />
  <parameter name="config"
             value="hk/hku/cecid/edi/as2/conf/as2.dao.xml" />
</component>
...
</module>
```

XPath	Expected information
/module/component[@id='daofactory']/class	The Java class to use when establishing a database connection. You can select: • hk.hku.cecid.piazza.commons.dao.ds.SimpleDSDAOFactory, if you want AS2 to manage the database connection pool • hk.hku.cecid.piazza.commons.dao.ds.DataSourceDAOFactory, if you want the application server to manage the database connection pool, which is accessible through the Java Naming and Directory Interface (JNDI) name.
Belows are fields you need to fill in if you are using SimpleDSDAOFactory	
/module/component[@id='daofactory']/parameter[@name='driver']/@value	The JDBC driver that should be used. The driver for Postgres is provided by default. The driver should be placed in the directory *<TOMCAT_HOME>*/webapps/corvus/WEB-INF/lib, where we suppose the web application repository is configured as *<TOMCAT_HOME>*/webapps.
/module/component[@id='daofactory']/parameter[@name='url']/@value	The URL for establishing the database connection. Please refer to the documentation of the JDBC driver for the syntax. For PostgreSQL, the syntax is jdbc:postgresql://*<IP or hostname of the database>*/*<message database name for AS2>*
/module/component[@id='daofactory']/parameter[@name='username']/@value	The username to connect to the database.
/module/component[@id='daofactory']/parameter[@name='password']/@value	The password for the username specified.
/module/component[@id='daofactory']/parameter[@name='pooling']/@value	The boolean value (true/false) specifying if connection pooling should be used.
/module/component[@id='daofactory']/parameter[@name='maxActive']/@value	The maximum number of active threads.
/module/component[@id='daofactory']/parameter[@name='maxIdle']/@value	The maximum number of threads that can remains idle.
/module/component[@id='daofactory']/parameter[@name='maxWait']/@value	The maximum amount of time (milliseconds) that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or −1 to wait indefinitely.
/module/component[@id='daofactory']/parameter[@name='config']/@value	Additional configuration files that will be used by the plugin. You should just leave it as is.
Belows are fields you need to fill in if you are using DataSourceDAOFactory	
datasource	The JNDI name of the data source for connecting to the message database, e.g. java:/comp/env/jdbc/as2db

Location of keystore for signing outgoing messages

```
<module id="as2.core" name="Corvus AS2" version="1.0">
...
<component id="keystore-manager" name="AS2 Key Store Manager">
```

```
  <class>hk.hku.cecid.piazza.commons.security.KeyStoreManager</class>
  <parameter name="keystore-location" value="corvus.p12"/>
  <parameter name="keystore-password" value="password"/>
  <parameter name="key-alias" value="corvus"/>
  <parameter name="key-password" value="password"/>
  <parameter name="keystore-type" value="PKCS12"/>
  <parameter name="keystore-provider"
             value="org.bouncycastle.jce.provider.BouncyCastleProvider"/>
</component>
...
</module>
```

XPath	Expected information
/module/component[@id='keystore-manager']/ parameter[@name='keystore-location']/@value	The path of the keystore for signing outgoing messages.
/module/component[@id='keystore-manager']/ parameter[@name='keystore-password']/@value	The password for accessing the keystore.
/module/component[@id='keystore-manager]/ parameter[@name='key-alias']/@value	The alias of the private key for a digital signature.
/module/component[@id='keystore-manager]/ parameter[@name='key-password']/@value	The password protecting the private key for a digital signature.
/module/component[@id='keystore-manager]/ parameter[@name='keystore-type']/@value	The keystore format. It is either PKCS12 or JKS.

Location of the message repository

Outgoing Repository:

```
<module id="as2.core" name="Corvus AS2" version="1.0">
...
<component id="outgoing-payload-repository" name="AS2 Outgoing Payload Repository">
<class>
hk.hku.cecid.edi.as2.module.PayloadRepository
</class>
  <parameter name="location" value="/as2-outgoing-repository" />
  <parameter name="type-edi" value="application/EDIFACT" />
  <parameter name="type-x12" value="application/EDI-X12" />
  <parameter name="type-eco" value="application/edi-consent" />
  <parameter name="type-xml" value="application/XML" />
</component>
...
</module>
```

XPath	Expected information
`/module/component[id='outgoing-payload-repository']/class`	The Java class responsible for handling outgoing payload. You should just leave it as is.
`/module/component[id='outgoing-payload-repository']/parameter[@name='location']/@value`	The directory that will store the outgoing payload. E.g., `c:program fileshermes2repositoryas2-outgoing-repository`
`/module/component[id='outgoing-payload-repository']/parameter[@name='type-edi']/@value`	You should leave these fields as is.
`/module/component[id='outgoing-payload-repository']/parameter[@name='type-x12']/@value`	
`/module/component[id='outgoing-payload-repository']/parameter[@name='type-eco']/@value`	
`/module/component[id='outgoing-payload-repository']/parameter[@name='type-xml']/@value`	

```xml
<module id="as2.core" name="Corvus AS2" version="1.0">
...
<component id="incoming-payload-repository" name="AS2 Incoming Payload Repository">
  <class>
    hk.hku.cecid.edi.as2.module.PayloadRepository
  </class>
  <parameter name="location" value="/as2-incoming-repository" />
  <parameter name="type-edi" value="application/EDIFACT" />
  <parameter name="type-x12" value="application/EDI-X12" />
  <parameter name="type-eco" value="application/edi-consent" />
  <parameter name="type-xml" value="application/XML" />
</component>
...
</module>
```

XPath	Expected information
/module/component[id='incoming-payload-repository']/class	The Java class responsible for handling incoming payloads. You should just leave it as is.
/module/component[id='outgoing-payload-repository']/parameter[@name='location']/@value	The directory that will store the outgoing payload. E.g., c:/program files/hermes2/repository/as2-incoming-repository
/module/component[id='outgoing-payload-repository']/parameter[@name='type-edi']/@value	You should leave these fields as is.
/module/component[id='outgoing-payload-repository']/parameter[@name='type-x12']/@value	
/module/component[id='outgoing-payload-repository']/parameter[@name='type-eco']/@value	
/module/component[id='outgoing-payload-repository']/parameter[@name='type-xml']/@value	

Original Message Repository (a temporary message repository used when Hermes is composing or receiving AS2 messages):

```
<module id="as2.core" name="Corvus AS2" version="1.0">
...
<component id="original-message-repository" name="AS2 Original Message Repository">
  <class>
    hk.hku.cecid.edi.as2.module.MessageRepository
  </class>
  <parameter name="location" value="/as2-message-repository" />
  <parameter name="is-disabled" value="false" />
</component>
...
</module>
```

XPath	Expected information
/module/component[id='original-payload-repository']/class	The Java class responsible for handling original messages. You should just leave it as is.
/module/component[id='original-payload-repository']/parameter[@name='location']/@value	The directory that will store outgoing payloads. E.g., c:/program files/hermes2/repository/as2-message-repository
/module/component[id='original-payload-repository']/parameter[@name='is-disabled']/@value	This flag indicates if the original message should be stored locally.

ebMS plugin

In the directory *<HERMES_2_PLUGINS_LOCATION>*/corvus-ebms/conf/hk/hku/cecid/ebms/spa/conf, there are some configuration files for Hermes's ebMS plugin. The configuration file you should edit depends on the property:

Properties	Configuration file
Log file location and level of logging	log4j.properties.xml
Connections to message database	ebms.module.xml
Location of keystore for signing outgoing messages	
Location of keystore for S/MIME decryption (incoming messages)	

Log file location and level of logging

To change the location of the log file, you will need to modify the XML file named log4j.properties.xml

```
<log4j:configuration debug="null" threshold="null" xmlns:log4j="http://jakarta.apache.
↪org/log4j/">
  <appender name="ebms" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/ebms.log"/>
    <param name="Encoding" value="UTF-8"/>
    <param name="MaxFileSize" value="100KB"/>
    <param name="MaxBackupIndex" value="1"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
          value="%d{yyyy-MM-dd HH:mm:ss} [%-12.12t] &lt;%-5p&gt; &lt;%m&gt;%n"/>
    </layout>
  </appender>
  <category additivity="true" name="hk.hku.cecid.ebms">
    <priority value="debug"/>
    <appender-ref ref="ebms"/>
  </category>
</log4j:configuration>
```

XPath	Expected information
/log4j:configuration/category/priority	The log level of the ebMS plugin logging. The available levels are debug, info, warn, error and fatal. If you set the value as debug, all logs will be printed.
/log4j:configuration/category/appender-ref@ref	The name of the appender element to be used for logging. The appender element specifies how to generate log files. In the above example, the appender named RollingFileAppender is used. The settings of the appender are specified by the referenced appender element.
/log4j:configuration/appender@class	The appender specified by this appender configuration element. Apache Log4j provides a series of appenders, such RollingFileAppender and DailyRollingFileAppender.
/log4j:configuration/appender@name	The name of this appender configuration element. /category/appender-ref@ref should reference the appender configuration element by this name.
/log4j:configuration/appender/param[@name='File']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is File)	The path of the ebMS log of this appender.
/log4j:configuration/appender/param[@name='Encoding']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is Encoding)	The encoding to be used for the log file.
/log4j:configuration/appender/param[@name='MaxFileSize']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is MaxFileSize)	If the size of a log file has grown to exceed this limit, another log file will be written and the old log file will be backed up. An index will be appended to the name of the old log file (e.g. ebms.log.1).
/log4j:configuration/appender/param[@name='MaxBackupIndex']/@value (i.e. The value attribute of the param element under the appender element, whose name attribute is MaxBackupIndex)	The maximum number of log files that will be backed up. For example, if it is set to 10, the maximum number of backed up log files will be 10 and their filenames will be *xxx*.log.1, *xxx*.log.2, ..., *xxx*.log.10.
/log4j:configuration/appender/layout/``param[@name='ConversionPattern']/@value	The pattern used when writing the log file.

Connection to message database

```
<module>
...
<component id="daofactory" name="System DAO Factory">
  <class>
  hk.hku.cecid.piazza.commons.dao.ds.SimpleDSDAOFactory
  </class>
  <parameter name="driver" value="org.postgresql.Driver" />
  <parameter name="url" value="jdbc:postgresql://localhost:5432/ebms" />
  <parameter name="username" value="corvus" />
  <parameter name="password" value="corvus" />
```

```
  <parameter name="pooling" value="true" />
  <parameter name="maxActive" value="30" />
  <parameter name="maxIdle" value="10" />
  <parameter name="maxWait" value="-1" />
  <parameter name="testOnBorrow" value="true" />
  <parameter name="testOnReturn" value="false" />
  <parameter name="testWhileIdle" value="false" />
  <parameter name="validationQuery" value="SELECT now()" />
  <parameter name="config">hk/hku/cecid/ebms/spa/conf/ebms.dao.xml</parameter>
</component>
...
</module>
```

XPath	Expected information
/module/component[@id='daofactory']/class	The Java class to use when establishing a database connection. You can select: • hk.hku.cecid.piazza.commons.dao.ds.SimpleDSDAOFactory, if you want ebMS to manage the database connection pool. • hk.hku.cecid.piazza.commons.dao.ds.DataSourceDAOFactory, if you want the application server to manage the database connection pool, which is accessible through the Java Naming and Directory Interface (JNDI) name.
Belows are fields you need to fill in if you are using SimpleDSDAOFactory	
/module/component[@id='daofactory']/parameter[@name='driver']/@value	The JDBC driver that should be used. The driver for Postgres is provided by default. The driver should be placed in the directory *<TOMCAT_HOME>*/webapps/corvus/WEB-INF/lib, where we suppose the web application repository is configured as *<TOMCAT_HOME>*/webapps.
/module/component[@id='daofactory']/parameter[@name='url']/@value	The URL for establishing the database connection. Please refer to the documentation of the JDBC driver for the syntax. For PostgreSQL, the syntax is jdbc:postgresql://*<IP or hostname of the database>*/*<message database name for AS2>*
/module/component[@id='daofactory']/parameter[@name='username']/@value	The username to connect to the database.
/module/component[@id='daofactory']/parameter[@name='password']/@value	The password for the username specified.
/module/component[@id='daofactory']/parameter[@name='pooling']/@value	The boolean value (true/false) specifying if connection pooling should be used.
/module/component[@id='daofactory']/parameter[@name='maxActive']/@value	The maximum number of active threads.
/module/component[@id='daofactory']/parameter[@name='maxIdle']/@value	The maximum number of threads that can remains idle.
/module/component[@id='daofactory']/parameter[@name='maxWait']/@value	The maximum amount of time (milliseconds) that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely.
/module/component[@id='daofactory']/parameter[@name='testOnBorrow']/@value	Parameter used by system during testing, please keep it unchanged
/module/component[@id='daofactory']/parameter[@name='testOnReturn']/@value	Parameter used by system during testing, please keep it unchanged
/module/component[@id='daofactory']/parameter[@name='tesWhileIdle']/@value	Parameter used by system during testing, please keep it unchanged
/module/component[@id='daofactory']/parameter[@name='validateQuery']/@value	Parameter used by system during testing, please keep it unchanged
/module/component[@id='daofactory']/parameter[@name='config']/@value	Additional configuration files that will be used by the plugin. You should just leave it as is.
Belows are fields you need to fill in if you are using DataSourceDAOFactory	
datasource	The JNDI name of the data source for connecting to the message database, e.g. java:/comp/env/jdbc/ebmsdb

Location of keystore for signing outgoing messages

```
<module id="ebms.main" name="Ebms Plugin" version="1.0">
...
<component id="keystore-manager-for-signature" name="Key Store Manager for Digital␣
↪Signature">
  <class>hk.hku.cecid.piazza.commons.security.KeyStoreManager</class>
  <parameter name="keystore-location" value="corvus.p12"/>
  <parameter name="keystore-password" value="password"/>
  <parameter name="key-alias" value="corvus"/>
  <parameter name="key-password" value="password"/>
  <parameter name="keystore-type" value="PKCS12"/>
  <parameter name="keystore-provider"
             value="org.bouncycastle.jce.provider.BouncyCastleProvider"/>
  </component>
...
</module>
```

XPath	Expected information
/module/ component[@id='keystore-manager-for-signature']/ parameter[@name='keystore-location']/@value	The path of the keystore for signing outgoing messages.
/module/ component[@id='keystore-manager-for-signature']/ parameter[@name='keystore-password']/@value	The password for accessing the keystore.
/module/ component[@id='keystore-manager-for-signature']/ parameter[@name='key-alias']/@value	The alias of the private key for digital signature.
/module/ component[@id='keystore-manager-for-signature']/ parameter[@name='key-password']/@value	The password protecting the private key for digital signature.
/module/ component[@id='keystore-manager-for-signature']/ parameter[@name='keystore-type']/@value	The keystore format. It is either PKCS12 or JKS.

Location of keystore for S/MIME decryption (incoming messages)

```
<module id="ebms.main" name="Ebms Plugin" version="1.0">
...
  <component id="keystore-manager-for-decryption" name="Key Store Manager for␣
↪Decryption (ebMS over SMTP)">
    <class>hk.hku.cecid.piazza.commons.security.KeyStoreManager</class>
    <parameter name="keystore-location" value="corvus.p12"/>
    <parameter name="keystore-password" value="password"/>
    <parameter name="key-alias" value="corvus"/>
    <parameter name="key-password" value="password"/>
    <parameter name="keystore-type" value="PKCS12"/>
    <parameter name="keystore-provider" value="org.bouncycastle.jce.provider.
↪BouncyCastleProvider"/>
  </component>
...
</module>
```

XPath	Expected information
`/module/component[@id='keystore-manager-for-decryption']/parameter[@name='keystore-location']/@value`	The path of the keystore for decrypting incoming messages with S/MIME encryption.
`/module/component[@id='keystore-manager-for-decryption']/parameter[@name='keystore-password']/@value`	The password for accessing the keystore
`/module/component[@id='keystore-manager-for-decryption']/parameter[@name='key-alias']/@value`	The alias of the private key for the decryption.
`/module/component[@id='keystore-manager-for-decryption']/parameter[@name='key-password']/@value`	The password protecting the private key for digital signatures.
`/module/component[@id='keystore-manager-for-decryption']/parameter[@name='keystore-type']/@value`	The keystore format. It is either `PKCS12` or `JKS`.

See also

- *Developing Hermes Applications*
- OASIS ebXML Message Service Specification 2.0
- MIME-based Secure Peer-to-Peer Business Data Interchange over the Internet Using HTTP AS2

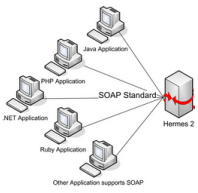
Using Hermes API

Introduction

Hermes has implemented web services to communicate with external applications. The main advantages of using web services are to reduce coupling between external applications and Hermes, and to allow external applications to integrate Hermes seamlessly using any programming languages that support sending SOAP Messages with Attachments or calling RESTful APIs. This article helps developers write a client talking to Hermes using web services based on SOAP or RESTful APIs.

For more information on the installation and partnerships of Hermes, please refer to *Installing Hermes* and *The First Step*.

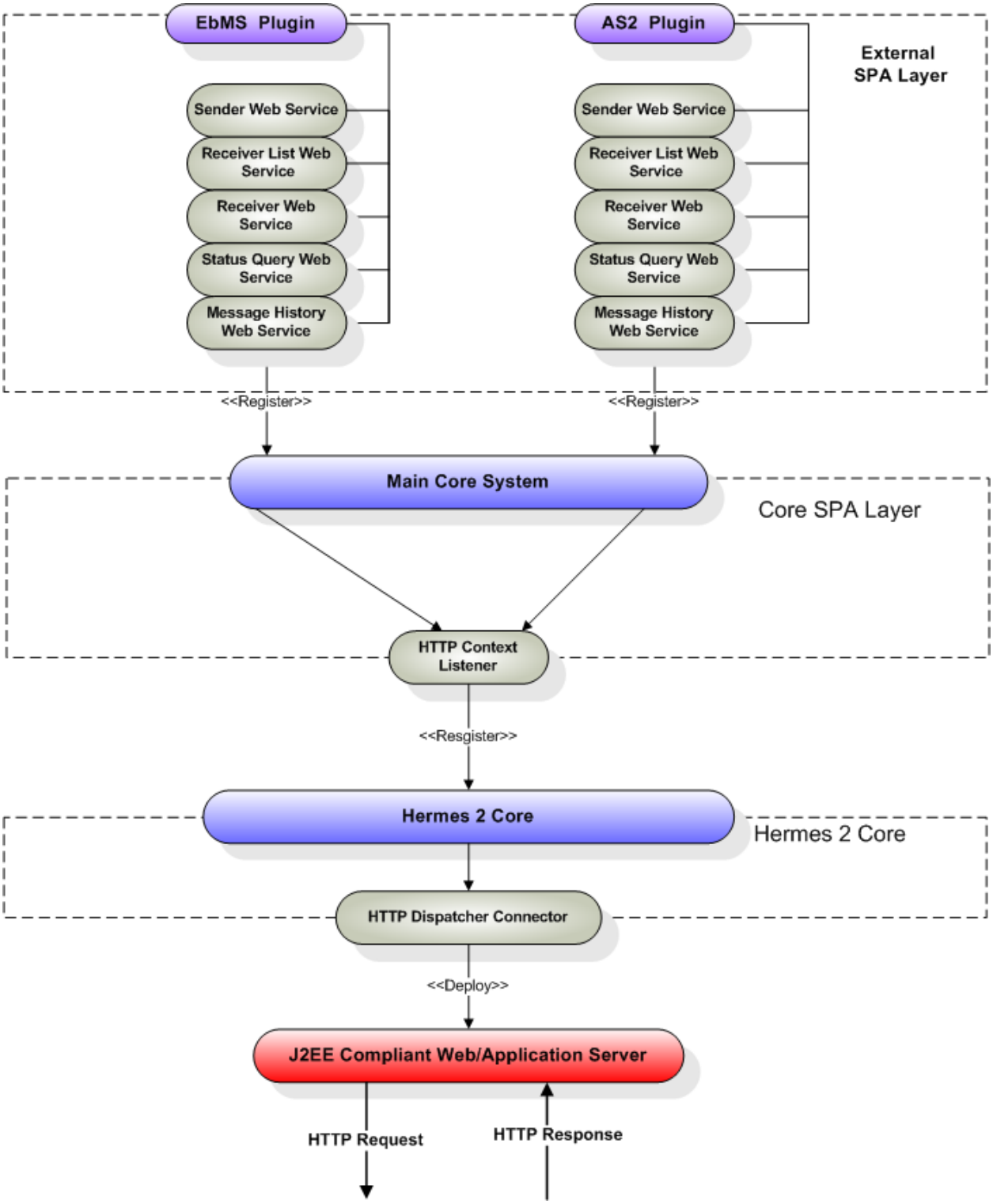
To choose whether to use SOAP or REST APIs in your application, you may refer to Understanding SOAP and REST Basics and Differences for guidance.



Overview

Here is a brief summary about the communication architecture between Hermes and external applications. The core of Hermes can attach to any J2EE compliant web server as a servlet. The core itself provides neither any web services or HTTP listeners, nor any functionality related to messaging. All features in Hermes are derived from the core using SPA (Simple Plugin Architecture).

One of the core SPAs, called Main Plugin (shown below in the core SPA layer), provides an HTTP context listener that accepts HTTP requests at the specified context path (extension point) for external invocation. The protocol-specific SPA ebMS and AS2 plugins (shown below in the external SPA layer) make use of this listener to provide all SOAP and REST web services.



In the default Hermes installation, each of ebMS 2.0 and AS2 plugins supports the following registered web services in Hermes:

Table 3.1: ebMS 2.0 Hermes API

Functionality	REST(ebMS)[1]	SOAP(ebMS)
Send message	POST:corvus/api/message/send/ebms	/corvus/httpd/ebms/sender
List received message ID	GET:/corvus/api/message/receive/ebms	/corvus/httpd/ebms/receiver_list
Download received message payload	POST:/corvus/api/message/receive/ebms	/corvus/httpd/ebms/receiver
Get message status	GET:/corvus/api/message/send/ebms	/corvus/httpd/ebms/status
Reset message status	POST:/corvus/api/message/redownload/ebms	/corvus/httpd/ebms/permitdl
Query message with parameters	GET:/corvus/api/message/history/ebms	/corvus/httpd/ebms/msg_history
Add partnership	POST:/corvus/api/partnership/ebms	NIL
Delete partnership	DELETE /corvus/api/partnership/ebms/{pid}	NIL
Update partnership	POST:/corvus/api/partnership/ebms	NIL
Get partnerships	GET:/corvus/api/partnership/ebms	NIL

Table 3.2: AS2 Hermes API

Functionality	REST(AS2)[1]	SOAP(AS2)
Send Message	POST:/corvus/api/message/send/as2	/corvus/httpd/as2/sender
List received message ID	GET:/corvus/api/message/receive/as2	/corvus/httpd/as2/receiver_list
Download received message payload	POST:/corvus/api/message/receive/as2	/corvus/httpd/as2/receiver
Get message status	GET:/corvus/api/message/send/as2	/corvus/httpd/as2/status
Query message with parameters	GET:/corvus/api/message/history/as2	/corvus/httpd/as2/msg_history
Add partnership	POST:/corvus/api/partnership/as2	NIL
Delete partnership	DELETE /corvus/api/partnership/as2/{pid}	NIL
Update partnership	POST:/corvus/api/partnership/as2	NIL
Get partnerships	GET:/corvus/api/partnership/as2	NIL

Note:

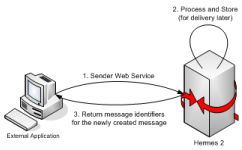
- To make an REST API request, the simplest way is to use `curl` as a command line REST client, or Postman as a GUI based client is a useful tool too.
- To enhance the security of Hermes REST API, HTTP Basic Authenication is enabled for the Rest API. Please place the base64 encoded username:password in the HTTP Header as below : `HTTP Header:Authorization = basic base64encode[username:pwd]` where the username and pwd are defined in `tomcat-users.xml`

ebMS 2.0 Web Service

Send message

This is a web service interface for external parties to request Hermes to send an ebMS message to another Hermes or an ebMS compliant messaging gateway. The service provides a message identifier to the sender for future reference. This is the main channel for external applications to deliver ebMS messages using Hermes.

[1]



SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/ebms/sender`

Request message

Instead composing the entire ebMS messages, the sender simply needs to send a web service request to Hermes with key parameters including `CPA ID`, `Service` and `Action`. These 3 key parameters identify the sending partnership in Hermes that will be used to configure the ebMS message.

The elements in the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP request is shown below.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <tns:cpaId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[CPA_Id]</
↪tns:cpaId>
      <tns:service xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Service]</
↪tns:service>
      <tns:action xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Action]</
↪tns:action>
      <tns:convId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Conversation_Id]
↪</tns:convId>
      <tns:fromPartyId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[From_Party_
↪Id]</tns:fromPartyId>
      <tns:fromPartyType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[From_
↪Party_Type]</tns:fromPartyType>
      <tns:toPartyId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[To_Party_Id]
↪</tns:toPartyId>
      <tns:toPartyType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[To_Party_
↪Type]</tns:toPartyType>
      <tns:refToMessageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">
↪[Reference_Message_Id]</refToMessageId>
      <tns:serviceType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Service_
↪Type]</tns:serviceType>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
<!-- Attached payloads... -->
```

The descriptions of the elements in the SOAP body are as follows:

Element	Mandatory	Description
<cpaId>, <service>, <action>	Yes	They are the CPA Id, Service and Action elements in the ebMS messages sent by Hermes. These three fields are used to identify the partnership used to send and receive the ebMS messages by the sending and receiving parties respectively. **These are required to identify a registered partnership in Hermes.**
<convId>	Yes	This corresponds to the conversation id element in the ebMS messages sent by Hermes.
<fromPartyId>	Yes	This identifies the sender. [ebMS v2_0 3.1.1] It corresponds to the PartyId element in From element of ebMS messages sent by Hermes.
<fromPartyType>	Yes	This identifies the domain of the sender. It corresponds to the type attribute of PartyId in the From element of ebMS messages sent by Hermes.
<toPartyId>	Yes	This identifies the receiver. [ebMS v2_0 3.1.1] It corresponds to the PartyId element in To element of ebMS messages sent by Hermes.
<toPartyType>	Yes	This identifies the domain of the receiver. It corresponds to the type attribute of PartyId in the From element of ebMS messages sent by Hermes.
<refToMessageId>	No	This corresponds to the RefToMessageId of ebMS messages sent by Hermes.
<serviceType>	No	A type identifier for the ebXML service defined in the partnership.

Response message

The elements inside the SOAP body uses namespace URI http://service.ebms.edi.cecid.hku.hk/.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
        <SOAP-ENV:Body>
            <message_id xmlns="http://service.ebms.edi.cecid.hku.hk/">[newly_created_
↪message_id]</message_id>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In the SOAP request message, the <message_id> element is the message identifier assigned by Hermes in

the sending party. The sending application can use it for later reference and status tracking with *Get message status* web service.

REST[1]

Request message

```
$ curl -X POST --data '{"partnership_id":"<partnership_id>", "from_party_id":"<from>",
↪ "to_party_id":"<to>", "conversation_id":"<conv>", "payload":"<payload>"}' http://
↪<HOST>:<PORT>/corvus/api/message/send/ebms
```

Response message

```
{
    "id": "<message_id>"
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

List received message ID

This web service is used by the application of the receiving party to retrieve message identifiers of received and processed ebMS messages that have not been downloaded. These message identifiers will be used to retrieve message payloads with *Download received message payload* web service.

SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/ebms/receiver_list`

Request message

The elements in the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:cpaId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[CPA_Id]</
↪tns:cpaId>
        <tns:service xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Service]</
↪tns:service>
        <tns:action xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Action]</
↪tns:action>
        <tns:convId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Conversation_
↪Id]</tns:convId>
        <tns:fromPartyId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[From_
↪Party_Id]</tns:fromPartyId>
        <tns:fromPartyType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[From_
↪Party_Type]</tns:fromPartyType>
        <tns:toPartyId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[To_Party_Id]
↪</tns:toPartyId>
        <tns:toPartyType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[To_Party_
↪Type]</tns:toPartyType>
        <tns:numOfMessages xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Number_
↪of_messages]</tns:numOfMessages>
```

```
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The descriptions of the elements in the SOAP body are as follows:

Element	Mandatory	Description
<cpaId>, <service>, <action>	Yes	The CPA Id, Service and Action elements in ebMS messages sent by Hermes. These three fields identify the partnership used to send ebMS messages. **These are required to query the list of available messages**.
<convId>	No	Only the identifiers of messages with a matching Conversation Id will be retrieved.
<fromPartyId>	No	Only the identifiers of messages with a matching From Party Id will be retrieved.
<fromPartyType>	No	Only the identifiers of messages with a matching From Party Type will be retrieved.
<toPartyId>	No	Only the identifiers of messages with a matching To Party Id will be retrieved.
<toPartyType>	No	Only the identifiers of messages with a matching To Party Type will be retrieved.
<numOfMessages>	No	The maximum number of message identifiers retrieved by this request.

Response message

The elements inside the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <messageIds xmlns="http://service.ebms.edi.cecid.hku.hk/">
            <messageId>[downloadable_message_id]</messageId>
            <messageId>[downloadable_message_id]</messageId>
        </messageIds>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Each element in the `messageIds` represents the message identifier of an ebMS message received by Hermes.

Please note that a message is considered to be downloaded only when the message body has been downloaded by *Download received message payload* SOAP web service. If your application never calls *Download received message payload* SOAP web service to download the messages, the same set of message identifiers will always be retrieved.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/message/receive/ebms?partnership_id=
↪<partnership_id>
```

REST reponse message

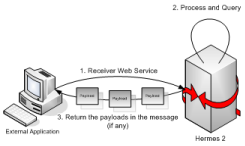
```
{
    "message_ids": [
        {
            "id": "<message_id>",
            "timestamp": 1234567890,
            "status": "<status>"
        }
    ]
}
```

Please note that a message is considered to be downloaded when the message id is returned by this REST API call.

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Download received message payload

This web service is used by the application of the receiving party to retrieve message payloads of received ebMS messages. After the message payloads have been downloaded, the message will be marked as received, and its message identifier will no longer be retrieved by *List received message ID* web service.



SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/ebms/receiver`

Request message

The elements in the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[id_of_
↪message_to_download]</tns:messageId>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `<messageId>` element contains a message identifier which obtained from *List received message ID* web service.

Response message

The element inside the SOAP body is using namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <hasMessage xmlns="http://service.ebms.edi.cecid.hku.hk/">[true_if_payload_in_
↪message]</hasMessage>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
<!--
    Attached payloads...
-->
```

If a payload is associated with the message identifier, the `<hasMessage>` element will have the value `true`. If the received ebMS message has payloads, the response message will have one or more SOAP attachments. Each SOAP attachment has a content type, which is set by the sending application.

Please note that a message is considered to be downloaded when the message is returned by this SOAP request.

REST[1]

Request message

```
$ curl -X POST --data '{"message_id":"<message_id"}' http://<HOST>:<PORT>/corvus/api/
↪message/receive/ebms
```

Response message

```
{
    "id": "<message_id>",
    "cpa_id": "<cpa>",
    "service": "<service>",
    "action": "<action>",
    "from_party_id": "<from>",
    "to_party_id": "<to>",
    "conversation_id": "<conv>",
    "timestamp": 1234567890,
    "status": "<status>",
    "payloads": [
        {
            "payload": "<content>"
        }
    ]
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Get message status

This web service is used by the application of the sending party to retrieve the status of a delivered ebMS message.

The message status is a two-character code indicating the progress of an ebMS message. It provides a tracking service to monitor ebMS messages requested from Hermes.

SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/ebms/status`

Request message

The elements in the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[id_of_
↪message_to_download]</tns:messageId>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `<messageId>` element contains a message identifier obtained from *Send message* web service response or *List received message ID* web service.

Response message

The element inside the SOAP body is using namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <messageInfo xmlns="http://service.ebms.edi.cecid.hku.hk/">
            <status>[status]</status>
            <statusDescription>[statusDescription]</statusDescription>
            <ackMessageId>[ackMessageId]</ackMessageId>
            <ackStatus>[ackStatus]</ackStatus>
            <ackStatusDescription>[ackStatusDescription]</ackStatusDescription>
        </messageInfo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The descriptions of the elements in the SOAP body are as follows:

Element	Description
`<status>`	The current status of the ebMS message.
`<statusDescription>`	A text description of the current status.
`<ackMessageId>`	The message identifier of the associated acknowledgment (if any).
`<ackStatus>`	The current status of the associated acknowledgment (if any).
`<ackStatusDescription>`	A text description of the associated acknowledgment (if any).

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/message/send/ebms?id=<message_id>
```

Response message

```
{
    "message_id": "<message_id>",
    "status": "<status>"
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Reset message status

This web service is used by the application of the receiving party to reset the status of a downloaded ebMS message from `DL` (delivered) to `PS` (processed), so that it can be redownloaded again.

SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/ebms/permitdl`

Request message

The elements in the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">
            [The_message_id_you_want_to_redownload]
        </tns:messageId>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `<messageId>` element contains a message identifier obtained from the ebMS sender web service response or the ebMS receiver list web service.

Response message

The element inside the SOAP body is using namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <message_id xmlns="http://service.ebms.edi.cecid.hku.hk/">[newly_created_
↪message_id]</message_id>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In the SOAP request message, the `<message_id>` element is the `message identifier` where they are the same if reset status successfully.

REST[1]

Request message

```
$ curl -X POST --data '{"message_id":"<message_id>"}' http://<HOST>:<PORT>/corvus/api/
↪message/redownload/ebms
```

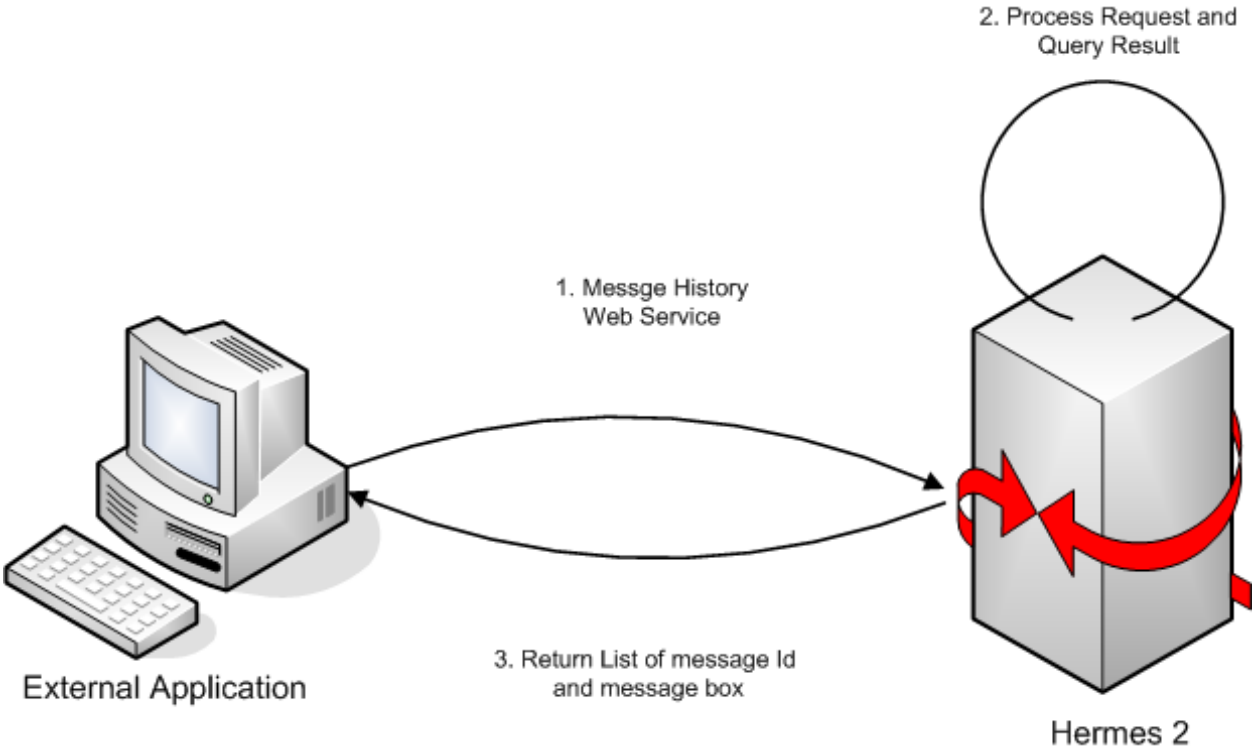
Response message

```
{
    "id": "<message_id>"
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Query message with parameters

This web service is used by the application of the sending or receiving party to query messages according to specific parameters.



SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/ebms/msg_history`

Request message

The elements in the SOAP body use the namespace URI `http://service.ebms.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageBox xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Box]</
↪tns:messageBox>
<tns:status xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Status]</
↪tns:status>
<tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Id]</
↪tns:messageId>
<tns:conversationId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Conversation_
↪Id]</tns:conversationId>
<tns:cpaId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[CPA_Id]</tns:cpaId>
<tns:service xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Defined_Service_with_
↪trading_party]</tns:service>
<tns:action xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Action]</tns:action>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response message

The element <messageList> inside the SOAP body uses the namespace URI http://service.ebms.edi.cecid.hku.hk/.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <messageList xmlns="http://service.ebms.edi.cecid.hku.hk/">
            <messageElement>
                <messageId>[message_id]</messageId>
                <messageBox>[message_box_containing_this_message]</messageBox>
            </messageElement>
            <messageElement>
                <messageId>[message_id]</messageId>
                <messageBox>[message_box_containing_this_message]</messageBox>
            </messageElement>
            <messageElement>...</messageElement>
            <messageElement>...</messageElement>
        </messageList>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The descriptions of the elements in the SOAP body are as follows:

Element	Description
<messageList>	A list of retrieved message elements (if any).
<messageElement>	A complex element containing messageId and messageBox values of a retrieved message.
<messageId>	The message identifier of a retrieved message.
<messageBox>	The message box of a retrieved message.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/message/history/ebms?message_id=
↪<message_id>&message_box=<message_box>&conversation_id=<cid>&cpa_id=<cpa_id>&
↪service=<service>&action=<action>&status=<status>&limit=<limit>
```

Response message

```
{
    "message_ids": [
        {
            "id": "<id>",
            "cpa_id": "<cpa_id>",
            "service": "<service>",
            "action": "<action>",
            "conversation_id": "<conversation_id>",
            "message_box": "<message_box>",
            "timestamp": "<timestamp>",
            "status": "<status>"
        }
```

```
    ]
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Add partnership

The ebMS Add Partnership web service is used by the application of the sending and receiving party to create partnership. For further details about ebMS partnership, please refer to *Setting Up ebMS 2.0 Partnerships*.

REST[1]

Request message

```
$ curl -X POST -- data '{"id":"<partnership_id>", "cpa_id":"<cpa>", "service":"
↪<service>", "action":"<action>", "transport-endpoint":"http://<RECEIVER HOST>:
↪<RECEIVER PORT>/corvus/httpd/ebms/inbound"}' \
http://<SENDER_HOST>:<SENDER_PORT>/corvus/api/partnership/ebms
```

Response message

```
{
    "id" : "<partnership_id>"
}
```

Delete partnership

The ebMS delete Partnership web service is used by the application of the sending and receiving party to delete partnership.

REST[1]

Request message

```
$ curl -X DELETE http://<HOST>:<PORT>/corvus/api/partnership/ebms/<partnership_id>
```

Response message

```
{
    "id": "<partnership_id>",
    "success": true
}
```

Update partnership

The ebMS update Partnership web service is used by the application of the sending and receiving party to update partnership. For further details about ebMS partnership, please refer to *Setting Up ebMS 2.0 Partnerships*.

REST[1]

Request message

```
$ curl -X POST -- data '{"id":"<partnership_id>", "cpa_id":"<cpa>", "service":"
↪<service>", "action":"<action>", "transport-endpoint":"http://<RECEIVER HOST>:
↪<RECEIVER PORT>/corvus/httpd/ebms/inbound"}' \
http://<SENDER_HOST>:<SENDER_PORT>/corvus/api/partnership/ebms
```

Response message

```
{
    "id": "<partnership_id>"
}
```

Get partnerships

The ebMS get Partnership web service is used by the application of the sending and receiving party to get all partnership details.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/partnership/ebms
```

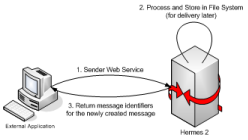
Response message

```
{
    "partnerships": [
        {
            "id": "<partership_id>",
            "cpa_id": "<cpa>",
            "service": "<service>",
            "action": "<action>",
            "disabled": false,
            "transport_endpoint": "http://<HOST>:<PORT>/corvus/httpd/ebms/inbound",
            "ack_requested": null,
            "signed_ack_requested": null,
            "duplicate_elimination": null,
            "message_order": null,
            "retries": 0,
            "retry_interval": 0,
            "sign_requested": false,
            "sign_certicate": null
        }
    ]
}
```

AS2 Web Service

Send Message

This web service is used by the application of the sending party to request Hermes to send an AS2 message to another Hermes or a compatible messaging gateway. The service returns a message identifier to the application for future reference.



SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/as2/sender`

Request message

The elements in the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:as2_from xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[as2_from]</
↪tns:as2_from>
        <tns:as2_to xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[as2_to]</
↪tns:as2_to>
        <tns:type xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[type]</tns:type>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
<!-- Attached payloads... -->
```

The descriptions of the elements in the SOAP body are as follows:

Element	Mandatory	Description
<as2_from>, <as2_to>	Yes	The values of the From and To fields in AS2 messages sent through the partnership by Hermes. These fields are used to identify the sending partnership. **These are required to identify the message destination.**
<type>	Yes	A three-character code indicating the content type of the sent payload. The available codes are: • edi, for the content type application/EDIFACT. • x12, for the content type application/EDI-X12. • eco, for the content type application/edi-consent. • xml, for the content type application/XML. • bin, for the content type application/octet-stream. For other values, Hermes will assume the content type of the payload is application/deflate, which means that the payload is compressed by Zip.

Response message

The elements inside the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <message_id xmlns="http://service.as2.edi.cecid.hku.hk/">[newly_created_
↪message_Id]</message_id>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The <message_id> element is the identifier of the sent message that can be used for later reference and status tracking with *Get message status* web service.

REST[1]

Request message

```
$ curl -X POST --data '{  "as2_from": <as2_from>, "as2_to": <as2_to>, "type": <type>,
↪"payload": <payload>}' http://<HOST>:<PORT>/corvus/api/message/send/as2
```

Response message

```
{
    "id": "<message_id>"
}
```

Note: To try the REST API, the simplest way is to use `curl` as a command line REST client, or Postman as a GUI based client is a useful tool too.

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

List received message ID

This web service is used by the application of the receiving party to retrieve message identifiers of received AS2 messages which have not been downloaded by the application. The message identifiers will be used to retrieve message payloads using *Download received message payload* web service.

SOAP

Service endpoint: `http://<HERMES_HOST>:<HERMES_PORT>/corvus/httpd/as2/receiver_list`

Request message

The elements in the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:as2_from xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[as2_from]</
↪tns:as2_from>
        <tns:as2_to xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[as2_to]</
↪tns:as2_to>
        <tns:numOfMessages xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">
↪[numOfMessages]</tns:numOfMessages>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The descriptions of the elements in the SOAP body are as follows:

Element	Mandatory	Description
`<as2_from>`, `<as2_to>`, `<as2_to>`	Yes	The values of the `From` and `To` fields in AS2 messages sent through the partnership by Hermes. These fields are used to identify the sending partnership. **These are required to query messages associated with the specified partnership.**
`<numOfMessages>`	No	The maximum number of message identifiers retrieved by this request.

Response message

The elements inside the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <messageIds xmlns="http://service.as2.edi.cecid.hku.hk/">
            <messageId>[downloadable_message_id]</messageId>
            <messageId>[downloadable_message_id]</messageId>
        </messageIds>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Each <downloadable_message_id> element in the response message represents the identifier of an AS2 message received by Hermes.

Please note that a message is considered to be downloaded only when the message body has been downloaded by *Download received message payload* SOAP web service. If your application never calls *Download received message payload* SOAP web service to download the messages, the same set of message identifiers will always be retrieved.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/message/receive/as2?partnership_id=
↪<partnership_id>
```

REST reponse message

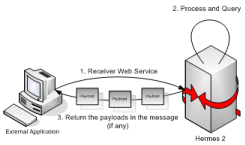
```
{
    "message_ids": [
        {
            "id": "<message_id>",
            "timestamp": 1234567890,
            "status": "<status>"
        }
    ]
}
```

Please note that a message is considered to be downloaded when the message id is returned by this REST API call.

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Download received message payload

This web service is used by the application of the receiving party to retrieve the message payloads of received AS2 messages. After the payloads have been downloaded, the message will be marked as received, and the message identifier of the message will no longer be retrieved by the AS2 receiver list service.



SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/as2/receiver`.

Request message

The elements in the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:messageId xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[id_of_
↪message_to_download]</tns:messageId>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response message

The element inside the SOAP body is using namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
     <SOAP-ENV:Header/>
     <SOAP-ENV:Body>
         <hasMessage xmlns="http://service.as2.edi.cecid.hku.hk/">[true_if_payload_in_
↪message]</hasMessage>
     </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
.<!-- Attached payloads... -->
```

If a payload is associated with the message identifier, then `<hasMessage>` will have the value `true`. If the received AS2 message has payloads, the response message will have one or more SOAP attachments. Each SOAP attachment has a content type, which is set by the sender application.

Please note that a message is considered to be downloaded when the message is returned by this SOAP request.

REST[1]

Request message

```
$ curl -X POST --data '{"id":"<message_id"}' http://<HOST>:<PORT>/corvus/api/message/
↪receive/as2
```

Response message

```
{
    "id": "<id>",
    "as2_from": "<as2_from>",
    "as2_to": "<as2_to>",
    "timestamp": 1234567890,
    "status": "<status>",
    "payloads": [
        {
            "payload": "<payload>"
        }
```

```
    ]
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Get message status

This web service is used by the application of the sending party to retrieve the message status of a sent or received AS2 message respectively.

SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/as2/status`.

Request message

The elements in the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <tns:messageId xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[id_of_
↪message_to_download]</tns:messageId>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response message

The element `<messageInfo>` inside the SOAP body is using the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <messageInfo xmlns="http://service.as2.edi.cecid.hku.hk/">
            <status>[status]</status>
            <statusDescription>[statusDescription]</statusDescription>
            <mdnMessageId>[mdnMessageId]</mdnMessageId>
            <mdnStatus>[mdnStatus]</mdnStatus>
            <mdnStatusDescription>[mdnStatusDescription]</mdnStatusDescription>
        </messageInfo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The descriptions of the elements in the SOAP body are as follows:

Element	Description
`<status>`	The current status of the AS2 message.
`<statusDescription>`	A text description of the current status.
`<mdnMessageId>`	The message identifier of the associated receipt (if any).
`<mdnStatus>`	The current status of the associated receipt.
`<mdnStatusDescription>`	A text description of the associated receipt.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/message/send/as2?id=<message_id>
```

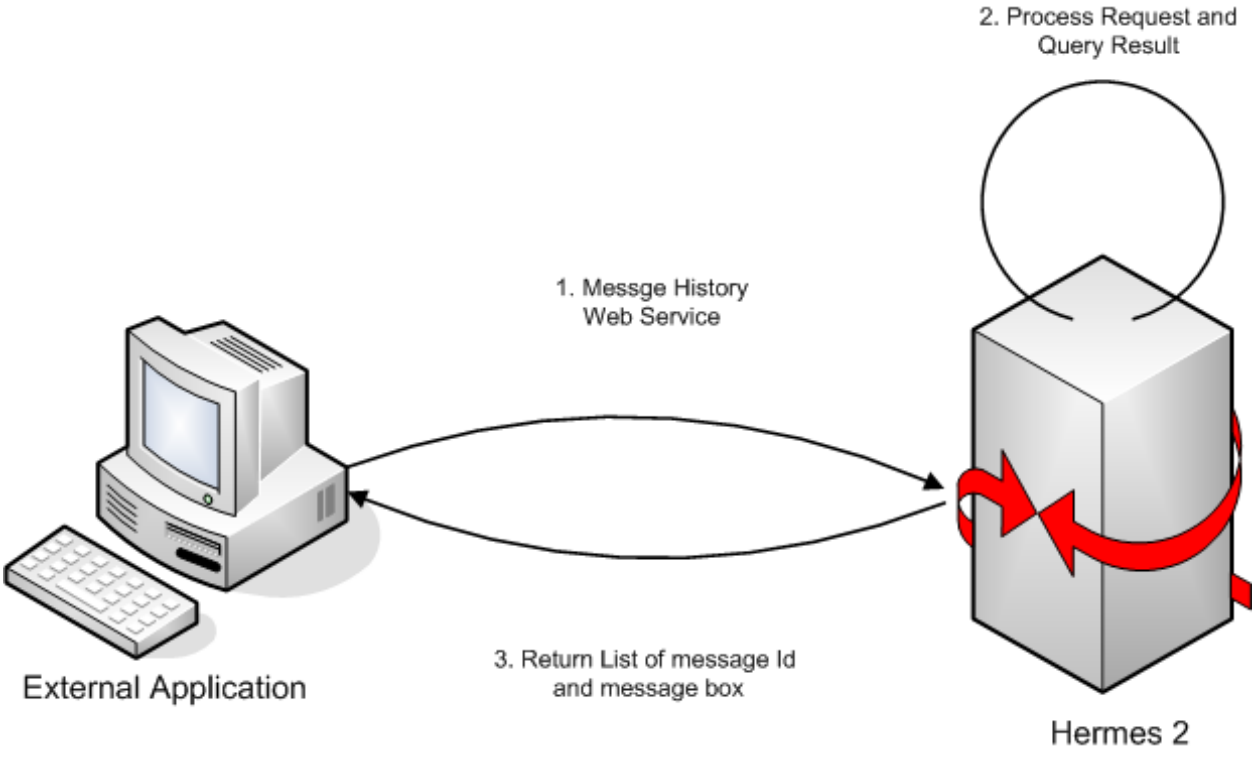
Response message

```
{
    "message_id": "<message_id>",
    "status": "<status>"
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Query message with parameters

This web service is used by the application of the sending or receiving party to query messages according to specific parameters.



SOAP

Service endpoint: `http://<HOST>:<PORT>/corvus/httpd/as2/msg_history`

Request message

The elements in the SOAP body use the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP request is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageBox xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[Message_Box]</
↪tns:messageBox>
<tns:status xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[Message_Status]</
↪tns:status>
<tns:messageId xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[Message_Id]</
↪tns:messageId>
<tns:as2From xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[AS2_From_Party]</
↪tns:as2From>
<tns:as2To xmlns:tns="http://service.as2.edi.cecid.hku.hk/">[AS2_To_Party]</tns:as2To>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response message

The element `<messageList>` in the SOAP body uses the namespace URI `http://service.as2.edi.cecid.hku.hk/`.

A sample SOAP response is shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <messageList xmlns="http://service.as2.edi.cecid.hku.hk/">
            <messageElement>
                <messageId>[message_id]</messageId>
                <messageBox>[message_box_containing_this_message] </messageBox>
            </messageElement>
            <messageElement>
                <messageId>[message_id]</messageId>
                <messageBox>[message_box_containing_this_message]</messageBox>
            </messageElement>
            <messageElement>...</messageElement>
            <messageElement>...</messageElement>
        </messageList>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The descriptions of the elements in the SOAP body are as follows:

Element	Description
`<messageList>`	The list of retrieved message elements.
`<messageElement>`	A complex element containing the `messageId` and `messageBox` values of the retrieved message.
`<messageId>`	The message identifier of the retrieved message.
`<messageBox>`	The message box of the retrieved message.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/message/history/as2?message_id=<message_
↪id>&message_box=<message_box>&as2_from=<as2_from>&as2_to=<as2_to>&status=<status>&
↪limit=<limit>
```

Response message

```
{
    "message_ids": [
        {
            "id": "<message_Id>",
            "as2_from": "<as2_from>",
            "as2_to": "<as2_to>",
            "message_box": "<message_box>",
            "timestamp": 1234567890,
            "status": "<status>"
        }
    ]
}
```

For the details specification of this REST API, please refer to HERMES RESTful OpenAPI Specification.

Add partnership

The AS2 Add Partnership web service is used by the application of the sending and receiving party to create partnership. For further details about AS2 partnership, please refer to *Setting Up AS2 Partnerships*.

REST[1]

Request message

```
$ curl -X POST -- data '{"id":"<partnership_id>", "as2_from":"<as2_from>", "as2_to":"
↪<as2_to>", "disabled":<true/false>, "sync_reply": "string", "subject": <subject>,
↪"recipient_address": <recipient_address>, "hostname_verified": <Yes/No>, "receipt_
↪address": <receipt_address>, "receipt_requested": <Yes/No>, "outbound_sign_required
↪": <Yes/No>, "outbound_encrypt_required": <Yes/No>,\
                        "outbound_compress_required": <Yes/No>, "receipt_sign_
↪required": <Yes/No>, "inbound_sign_required": <Yes/No>, "inbound_encrypt_required":
↪<Yes/No>, "retries": <no_of_retries>, "retry_interval": <retry_interval>, "sign_
↪algorithm": <sha1/md5>, "encrypt_algorithm": <3des/rc2>, "mic_algorithm": <sha1/md5>
↪, "encrypt_certicate": <cert_path>, "verify_certicate": <cert_path> }' \
  http://<SENDER_HOST>:<SENDER_PORT>/corvus/api/partnership/as2
```

Response message

```
{
    "id": "<partnership_id>"
}
```

Delete partnership

The AS2 delete Partnership web service is used by the application of the sending and receiving party to delete partnership.

REST[1]

Request message

```
$ curl -X DELETE http://<HOST>:<PORT>/corvus/api/partnership/as2/<partnership_id>
```

Response message

```
{
    "id": "<partnership_id>",
    "success": true
}
```

Update partnership

The ebMS update Partnership web service is used by the application of the sending and receiving party to update partnership. For further details about AS2 partnership, please refer to *Setting Up AS2 Partnerships*.

REST[1]

Request message

```
$ curl -X POST -- data '{"id":"<partnership_id>", "as2_from":"<as2_from>", "as2_to":"
↪<as2_to>", "disabled":<true/false>, "sync_reply": "string", "subject": <subject>,
↪"recipient_address": <recipient_address>, "hostname_verified": <Yes/No>, "receipt_
↪address": <receipt_address>, "receipt_requested": <Yes/No>, "outbound_sign_required
↪": <Yes/No>, "outbound_encrypt_required": <Yes/No>,\
                        "outbound_compress_required": <Yes/No>, "receipt_sign_
↪required": <Yes/No>, "inbound_sign_required": <Yes/No>, "inbound_encrypt_required":
↪<Yes/No>, "retries": <no_of_retries>, "retry_interval": <retry_interval>, "sign_
↪algorithm": <sha1/md5>, "encrypt_algorithm": <3des/rc2>, "mic_algorithm": <sha1/md5>
↪, "encrypt_certicate": <cert_path>, "verify_certicate": <cert_path> }' \
  http://<SENDER_HOST>:<SENDER_PORT>/corvus/api/partnership/as2
```

Response message

```
{
    "id": "<partnership_id>"
}
```

Get partnerships

The AS2 get Partnership web service is used by the application of the sending and receiving party to get all partnership details.

REST[1]

Request message

```
$ curl -X GET http://<HOST>:<PORT>/corvus/api/partnership/as2
```

Response message

```
{
    "partnerships": [
        {
```

```
            "id": "<partnership_id>",
            "as2_from": "<as2_from>",
            "as2_to": "<as2_to>",
            "disabled": true,
            "sync_reply": "<sync_reply>",
            "subject": "<subject>",
            "recipient_address": "<recipient_address>",
            "hostname_verified": "<yes_or_no>",
            "receipt_address": "<receipt_address>",
            "receipt_requested": "<yes_or_no>",
            "outbound_sign_required": "<yes_or_no>",
            "outbound_encrypt_required": "<yes_or_no>",
            "outbound_compress_required": "<yes_or_no>",
            "receipt_sign_required": "<yes_or_no>",
            "inbound_sign_required": "<yes_or_no>",
            "inbound_encrypt_required": "<yes_or_no>",
            "retries": 2,
            "retry_interval": 10,
            "sign_algorithm": "<sha1_or_md5>",
            "encrypt_algorithm": "<3des_or_rc2>",
            "mic_algorithm": "<sha1_or_md5>",
            "encrypt_certicate": "<cert_path>",
            "verify_certicate": "<cert_path>"
        }
    ]
}
```

Note:

- If error occurs when processing REST API request, it will return an error JSON response.

```
{
    "code": "<error code>",
    "message": "<error description>"
}
```

Table 3.3: Error code and message

Error code	Error Message
10000	ERROR_UNKNOWN
10001	ERROR_MISSING_REQUIRED_PARAMETER
10002	ERROR_PROTOCOL_UNSUPPORTED
10003	ERROR_READING_DATABASE
10004	ERROR_WRITING_DATABASE
10005	ERROR_READING_REQUEST
10006	ERROR_PARSING_REQUEST
10007	ERROR_RECORD_ALREADY_EXIST
10008	ERROR_DATA_NOT_FOUND
10009	ERROR_WRITING_MESSAGE
10010	ERROR_SENDING_MESSAGE
10011	ERROR_EXTRACTING_PAYLOAD_FROM_MESSAGE
10012	ERROR_UNKNOWN_ACTION

See also

- *The First Step*
- *Setting Up ebMS 2.0 Partnerships*
- *Setting Up AS2 Partnerships*
- HERMES RESTful OpenAPI Specification

Using Sample Clients

These sample Java clients demonstrate Hermes messaging flow. They provide a set of sample code for writing web service client applications connecting to Hermes.

Maintaining partnerships

A partnership must be registered on Hermes to send messages since partnerships contain information about your trading partner. A separate partnership is required to receive messages.

If you would like more information, please refer to the articles *What is an ebMS 2.0 partnership?* and *What is an AS2 partnership?*.

Let's take a look at the program parameters.

```
as2-partnership [partnership-xml] [config-xml] [log-path]
```

partnership-xml	The filepath of the partnership configuration file. Default is *<HERMES2_HOME>*/config/as2-partnership.xml.
config-xml	The filepath of the message configuration file. Default is *<HERMES2_HOME>*/config/as2-partnership/as2-request.xml.
log-path	The filepath of the logger to log query result or error. Default is *<HERMES2_HOME>*/logs/as2-partnership.log.

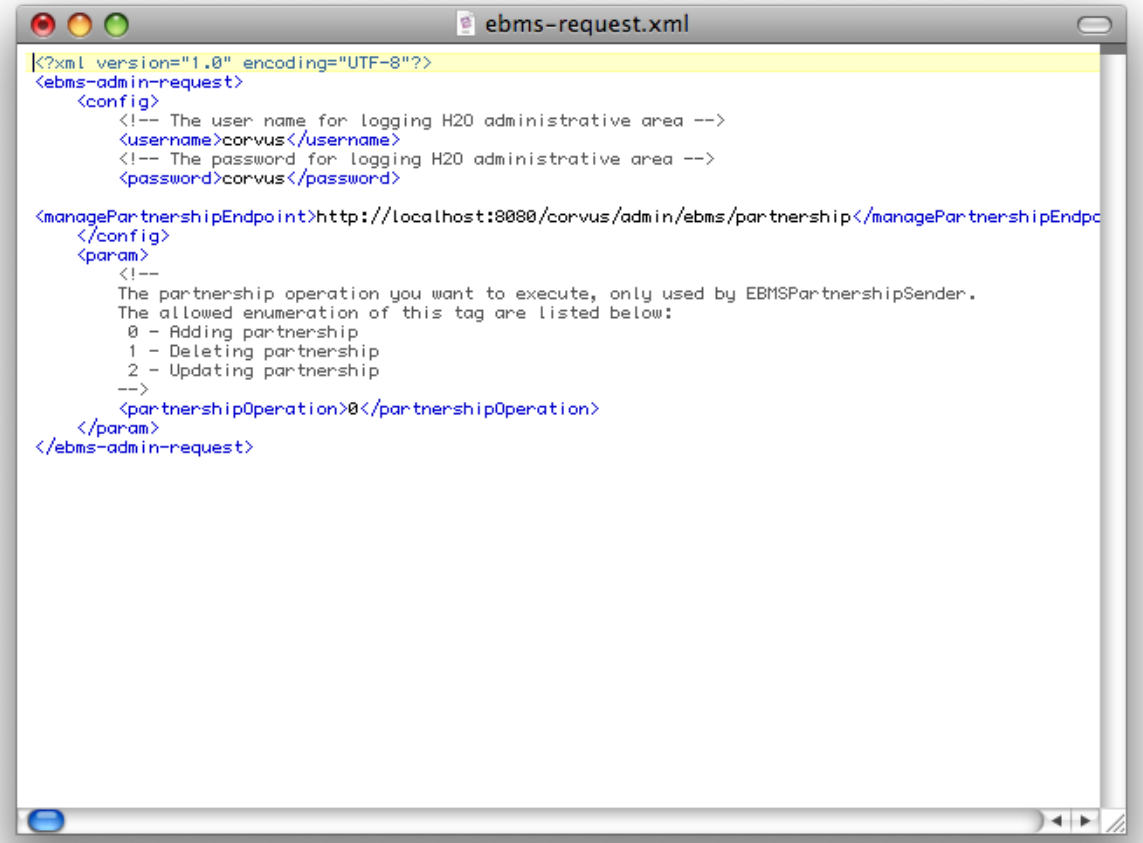
```
ebms-partnership [partnership-xml] [config-xml] [log-path]
```

partnership-xml	The filepath of the partnership configuration file. Default is *<HERMES2_HOME>*/config/ebms-partnership.xml.
config-xml	The filepath of the message configuration file. Default is *<HERMES2_HOME>*/config/ebms-partnership/ebms-request.xml.
log-path	The filepath of the logger to log query result or error. Default is *<HERMES2_HOME>*/logs/ebms-partnership.log.

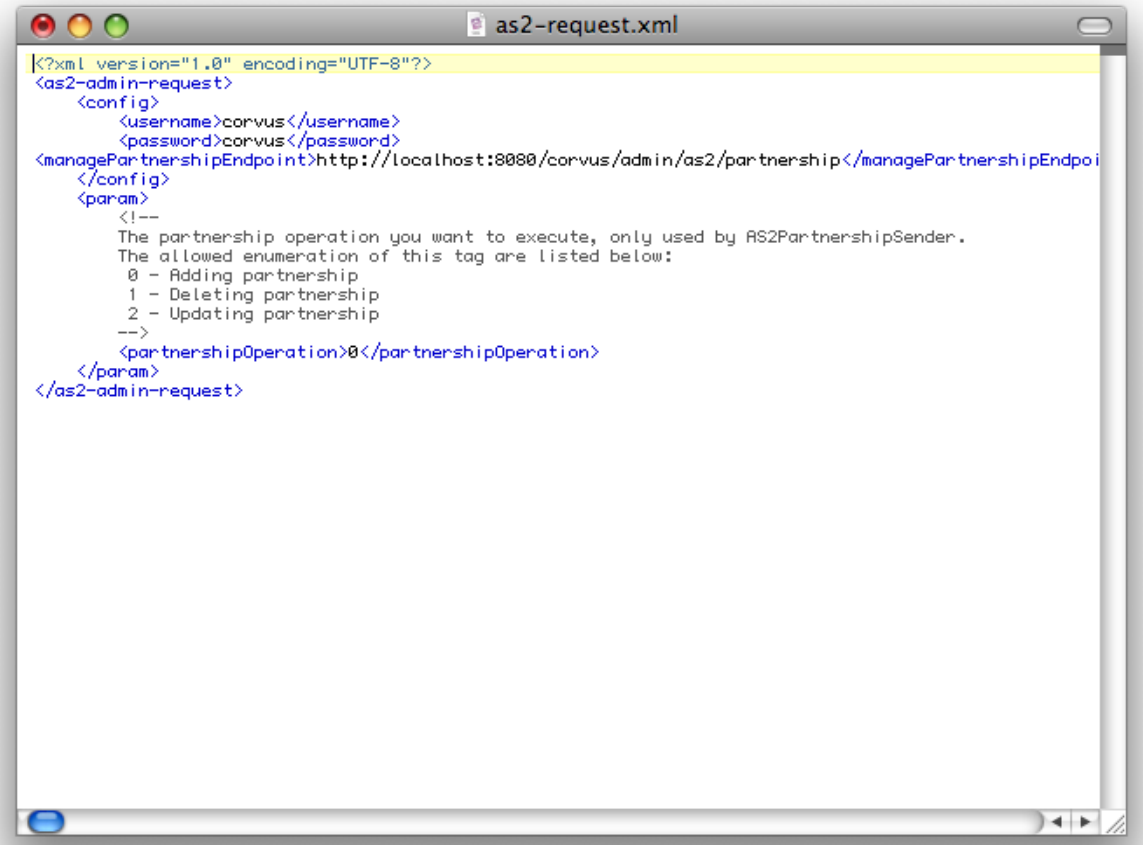
config-xml

Here is sample content of the config-xml files. These files are named `ebms-request.xml` and `as2-request.xml`, and placed under *<HERMES2_HOME>*`/config/ebms-partnership` and *<HERMES2_HOME>*`/config/as2-partnership` respectively.

Configuration file for ebMS:



Configuration file for AS2:



partnership-xml

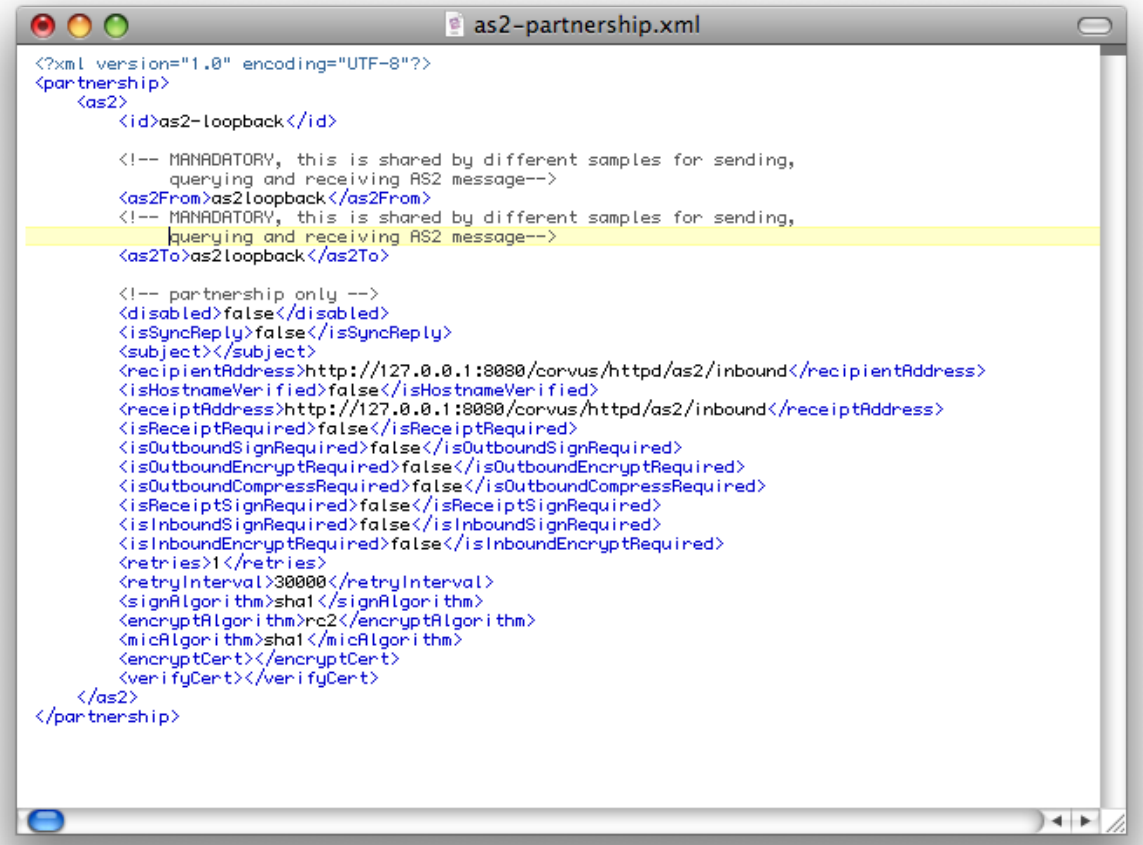
Sample content of the partnership-xml files are shown below. For more details, please refer to the articles *Setting Up ebMS 2.0 Partnerships* and *Setting Up AS2 Partnerships*.

Sample ebMS partnership:

ebms-partnership.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<partnership>
    <ebms>
        <id>ebms-loopback</id>

        <!-- MANADATORY, this is shared by different samples for sending,
             querying and receiving ebMS message-->
        <cpaId>cpaid</cpaId>
        <!-- MANADATORY, this is shared by different samples for sending,
             querying and receiving ebMS message-->
        <service>http://localhost:8080/corvus/httpd/ebms/inbound</service>
        <!-- MANADATORY, this is shared by different samples for sending,
             querying and receiving ebMS message-->
        <action>action</action>

        <!-- partnership only -->
        <disabled>false</disabled>
        <syncReplyMode>none</syncReplyMode>
        <transportEndpoint>http://localhost:8080/corvus/httpd/ebms/inbound</transportEndpoint>
        <transportProtocol>http</transportProtocol>
        <ackRequested>never</ackRequested>
        <ackSignRequested>never</ackSignRequested>
        <dupElimination>never</dupElimination>
        <actor></actor>
        <messageOrder>NotGuaranteed</messageOrder>
        <persistDuration></persistDuration>
        <retries>1</retries>
        <retryInterval>30000</retryInterval>
        <signRequested>false</signRequested>
        <dsAlgorithm></dsAlgorithm>
        <mdAlgorithm></mdAlgorithm>
        <encryptRequested></encryptRequested>
        <encryptAlgorithm></encryptAlgorithm>
        <signCert></signCert>
        <encryptCert></encryptCert>
        <isHostnameVerified></isHostnameVerified>
    </ebms>
</partnership>
```

Sample AS2 partnership:



ebMS

We have created two sample programs, **ebms-send** and **ebms-history**, to demonstrate how to communicate with Hermes web services.

Sending an ebMS message

ebms-send is a sample program to demonstrate how to upload an ebMS message to Hermes using the sender web service in the ebMS plugin. You can pack your payload as a SOAP message and send it to this service at the endpoint `http://<HOST>:<PORT>/corvus/httpd/ebms/sender`.

The elements in a SOAP request are shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:cpaId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [CPA_id] </tns:cpaId>
<tns:service xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [Service] </
↪tns:service>
<tns:action xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [Action] </tns:action>
<tns:convId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [Conversation_Id] </
↪tns:convId>
```

```
<tns:fromPartyId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [From_Party_ID] </
↪tns:fromPartyId>
<tns:fromPartyType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [From_Party_
↪Type] </tns:fromPartyType>
<tns:toPartyId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [To_Party_ID] </
↪tns:toPartyId>
<tns:toPartyType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [To_Party_Type] </
↪tns:toPartyType>
<tns:refToMessageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [Reference_
↪Message_Id] </refToMessageId>
<tns:serviceType xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> [Service_Type] </
↪tns:serviceType>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more information on the elements in the SOAP body of a send request, please refer to *SOAP*.

Before sending an ebMS message, make sure that a partnership is registered. Please refer to the section *Maintaining Partnerships* for more information.

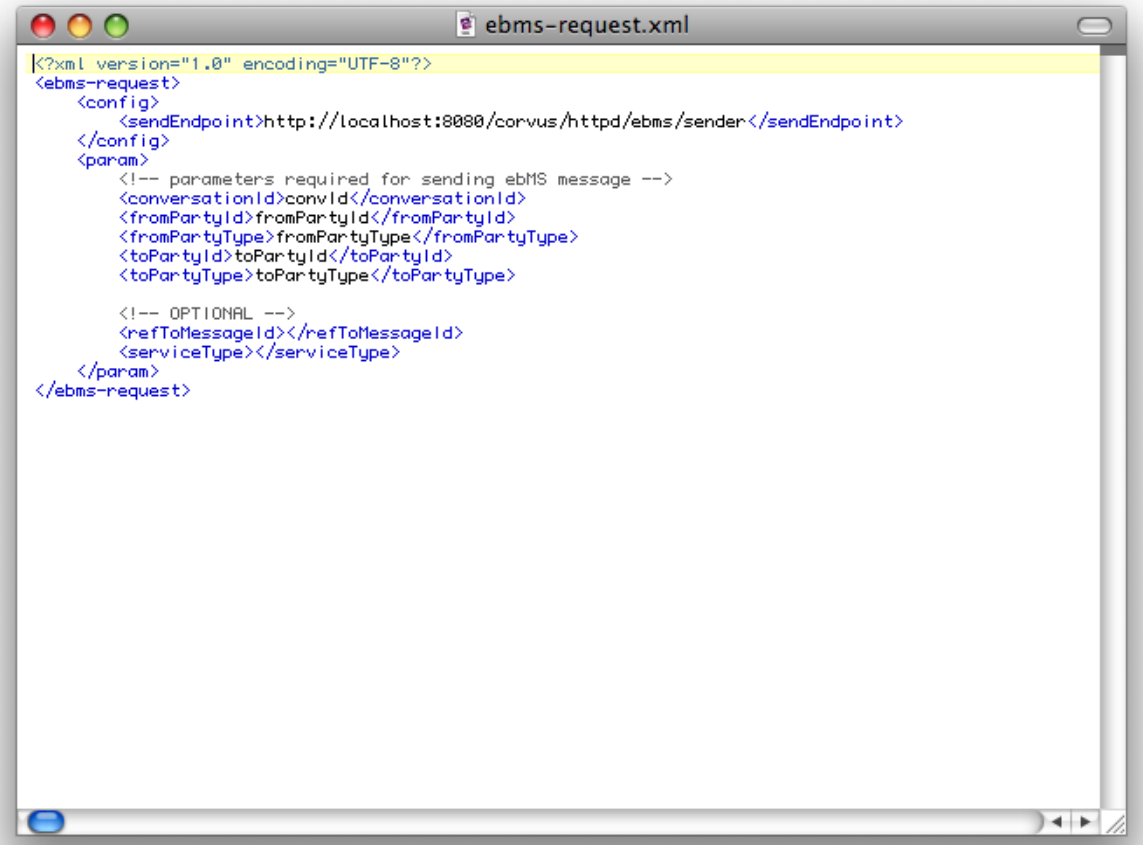
Let's take a look at the program parameters.

```
ebms-send [partnership-xml] [config-xml] [log-path] [payload-path]
```

partnership-xml	The filepath of the partnership configuration file. Default is *<HERMES2_HOME>*/config/ebms-partnership.xml.
config-xml	The filepath of the message configuration file. Default is *<HERMES2_HOME>*/config/ebms-send/ebms-request.xml.
log-path	The filepath of the logger to log query result or error. Default is *<HERMES2_HOME>*/logs/ebms-send.log.
payload (optional)	The filepath of the payload attached in the message. Default is *<HERMES2_HOME>*/config/ebms-send/testpayload.

config-xml

Here is sample content of the config-xml file. This file is named `ebms-request.xml`, and placed under *<HERMES2_HOME>*`/config/ebms-send`.



The following table explains the use of each element:

`<sendEndpoint>`	Refers to the address of the ebMS send service. It should be `http://<HOST>:<PORT>/corvus/httpd/ebms/sender`
`<conversationId>`	Identifies which conversation this message belongs to. **This is required for Hermes to create a valid message.**
`<fromPartyId>` `<fromPartyType>` `<toPartyId>` `<toPartyType>`	Identifies the sender and receiver. **These are required for Hermes to retrieve the message destination.**
`<refToMessageId>`	The message id that is targeted to respond to.
`<serviceType>`	A type identifier for the ebXML service defined in the partnership.

You only need to change `<sendEndpoint>` to contain the correct address.

partnership-xml

Another configuration file needed is partnership-xml, which is named `ebms-partnership.xml` and placed under `<HERMES2_HOME>/config` by default.

Sample content is shown below:

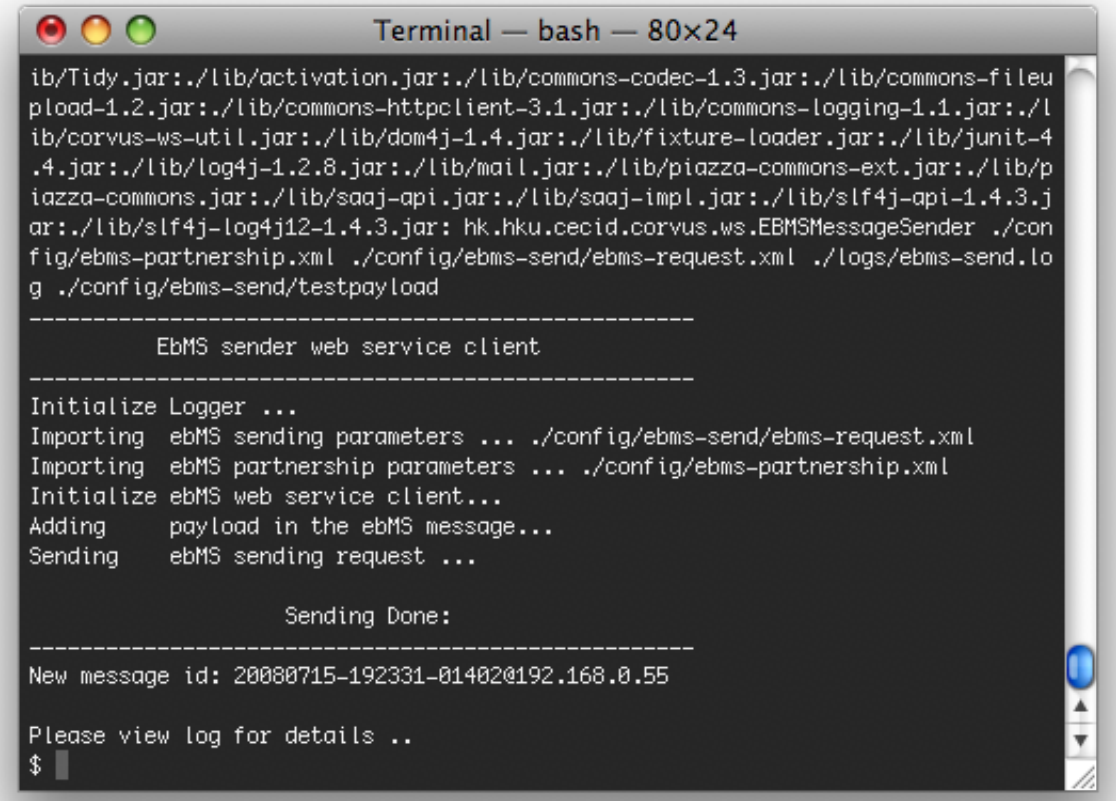
ebms-partnership.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<partnership>
    <ebms>
        <id>ebms-loopback</id>

        <!-- MANADATORY, this is shared by different samples for sending,
             querying and receiving ebMS message-->
        <cpaId>cpaid</cpaId>
        <!-- MANADATORY, this is shared by different samples for sending,
             querying and receiving ebMS message-->
        <service>http://localhost:8080/corvus/httpd/ebms/inbound</service>
        <!-- MANADATORY, this is shared by different samples for sending,
             querying and receiving ebMS message-->
        <action>action</action>

        <!-- partnership only -->
        <disabled>false</disabled>
        <syncReplyMode>none</syncReplyMode>
        <transportEndpoint>http://localhost:8080/corvus/httpd/ebms/inbound</transportEndpoint>
        <transportProtocol>http</transportProtocol>
        <ackRequested>never</ackRequested>
        <ackSignRequested>never</ackSignRequested>
        <dupElimination>never</dupElimination>
        <actor></actor>
        <messageOrder>NotGuaranteed</messageOrder>
        <persistDuration></persistDuration>
        <retries>1</retries>
        <retryInterval>30000</retryInterval>
        <signRequested>false</signRequested>
        <dsAlgorithm></dsAlgorithm>
        <mdAlgorithm></mdAlgorithm>
        <encryptRequested></encryptRequested>
        <encryptAlgorithm></encryptAlgorithm>
        <signCert></signCert>
        <encryptCert></encryptCert>
        <isHostnameVerified></isHostnameVerified>
    </ebms>
</partnership>
```

The mandatory elements are necessary to construct a SOAP message according to the WSDL. For more information, please read the article *Setting Up ebMS 2.0 Partnerships*.

Once you have configured these parameters correctly, the program can be executed. A message id will be displayed if the program has successfully executed.

Here is sample output from the program:



ebMS history query

ebms-history demonstrates the use of the message history web service (**msg-history**) in the ebMS plugin. There are several criteria defined for message history queries. By passing these criteria to Hermes through SOAP messages, the target results can be retrieved.

The message history service endpoint is `http://<HOST>:<PORT>/corvus/httpd/msg_history`.

The required elements in a SOAP request are as follows:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageBox xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Box]</
↪tns:messageBox>
<tns:status xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Status]</
↪tns:status>
<tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Id]</
↪tns:messageId>
<tns:conversationId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Conversation_
↪Id]</tns:conversationId>
<tns:cpaId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[CPA_Id]</tns:cpaId>
<tns:service xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Defined_Service_with_
↪trading_party]</tns:service>
```

```
<tns:action xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Action]</tns:action>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Let's take a look at the program parameters.

```
ebms-history [config-xml] [log-path]
```

config-xml	The filepath of the message configuration file. Default is `./config/ebms-history/ebms-request.xml`.
log-path	The filepath of the logger to log query result or error. Default is `./logs/ebms-history.log`.

config-xml

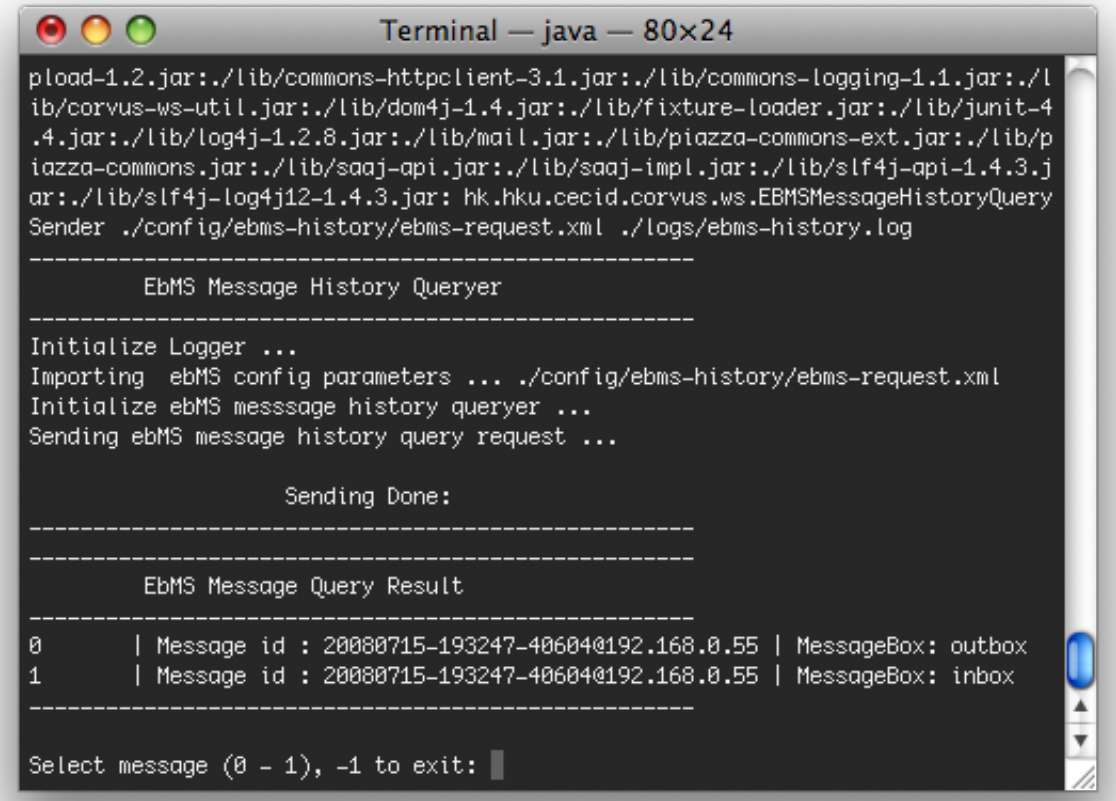
Here is sample content of the config-xml. This config-xml is named `ebms-request.xml`, and placed under *<HERMES2_HOME>*`/config/ebms-history`. There are several elements listed as search criteria. You can use the wildcard `%` in the values, and you can comment out unwanted elements.

ebms-request.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ebms-msg-history-request>
    <config>
        <endpoint>http://127.0.0.1:8080/corvus/httpd/ebms/msg_history</endpoint>
    </config>
    <param>
        <!--  These fields inside "Criteria" are uses for message history query -->
        <criteria>
            <!-- (Optional) Enter message ID Here, you can use % as wildcard -->
            <!-- <messageId>%localhost%</messageId> -->

            <!-- Choose either "INBOX" or "OUTBOX",
                 empty string stand for searching on both message box-->
            <!-- <messageBox>INBOX</messageBox> -->

            <!-- (Optional) Enter ConversationId here, you can use % as wildcard -->
            <!-- <conversationId></conversationId> -->

            <!-- (Optional) Enter Cpaid here, you can use % as wildcard -->
            <!-- <cpaid></cpaid> -->

            <!-- Enter Service string here, you can use % as wildcard -->
            <!-- <service></service> -->

            <!-- Enter Action string here, you can use % as wildcard -->
            <!-- <action></action> -->

            <!-- Enter target Message Status here, You can chose
                    DL  for  Delivered
                    DF  for  Delivery Failure
                    PE for   Processed Error
                    PS  for  Processed
                    PR  for  Processing
                    RC  for  Received
                empty string means searching on ALL types of message status            -->
            <!-- <status>PS</status> -->
        </criteria>
    </param>
```

Program operation

If the query has successfully executed, the result will be similar to the following:



After the messages are displayed by the program, you can perform further action by choosing your target message. If the message is placed under **OUTBOX**, the program will query its current status. If the message is placed under **INBOX**, the program will download the payload(s) if available.

Retrieving message payloads

There is a receiver web service provided by the ebMS plugin to retrieve messages. The receiver service endpoint is `http://<HOST>:<PORT>/corvus/httpd/ebms/receiver`.

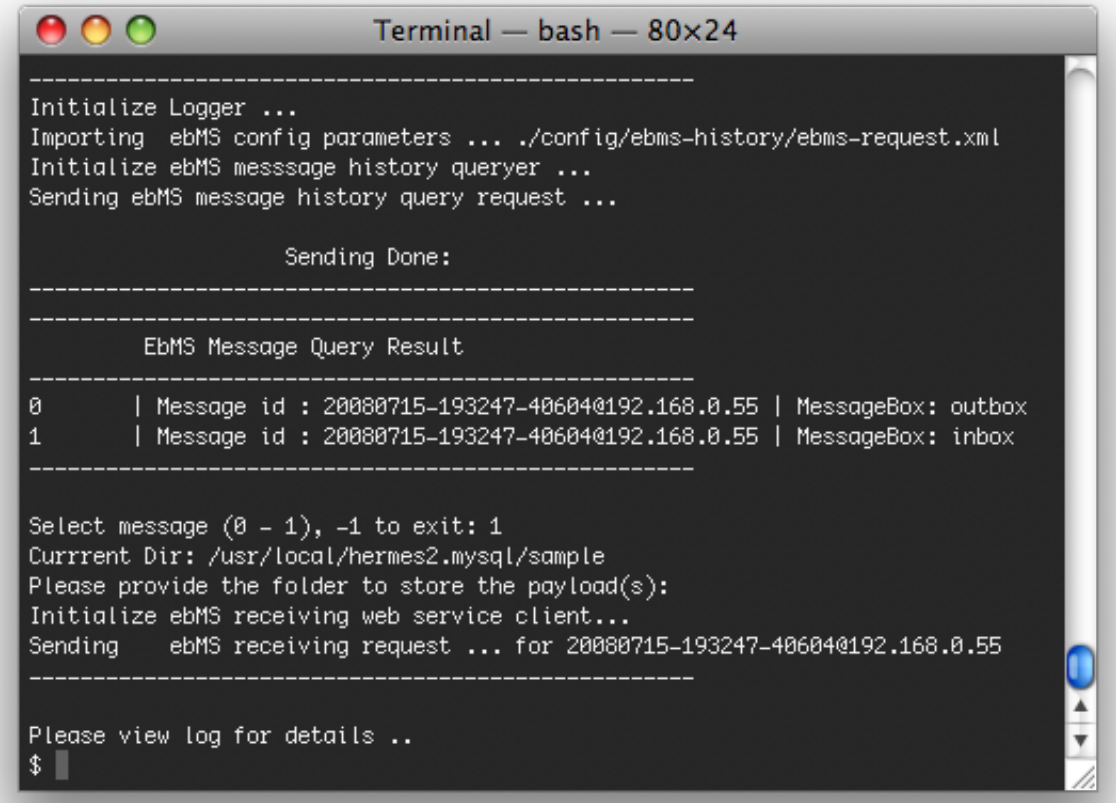
The required elements in a SOAP request are the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> <MessageId></
↪tns:messageId>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The message id is the only criteria needed to retrieve the payload of the target message. However, the payload is only

available once. If the payload of a message has already been downloaded, the program will not be able to retrieve it again.

The program will ask for a directory to store the payload, which will be stored with the name `ebms.<MessageId>.Payload.<IndexofPayload>`.



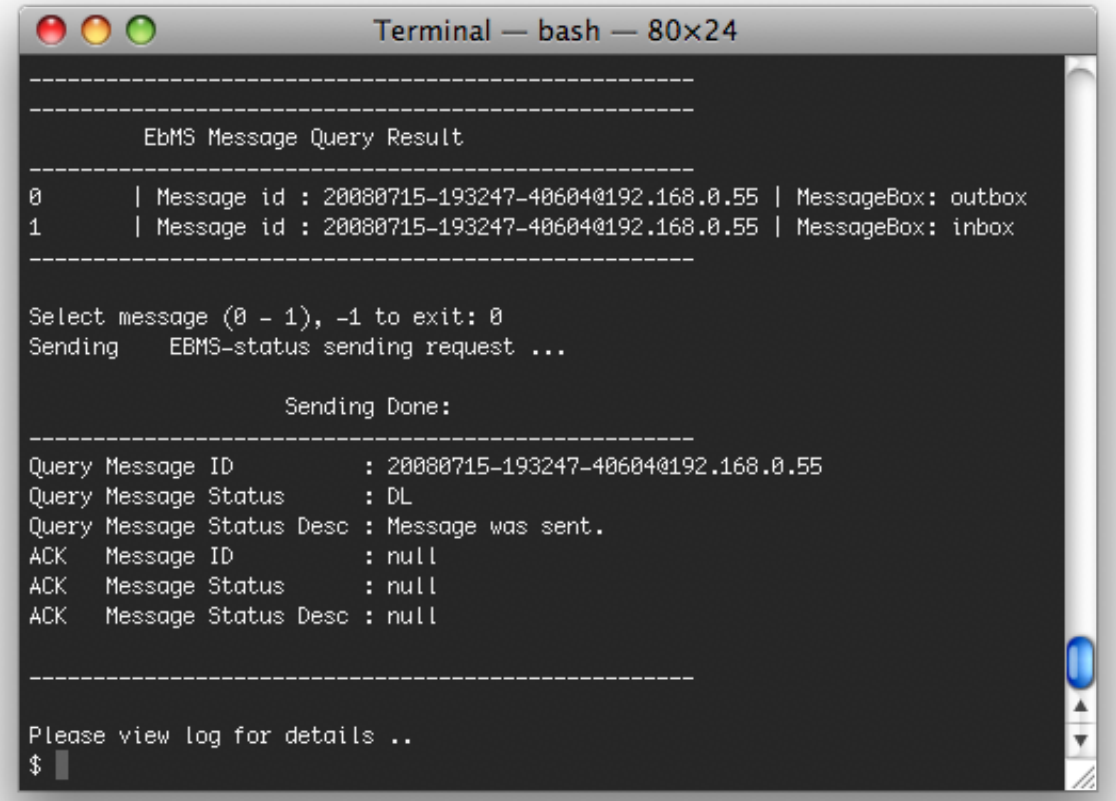
Check outgoing message status

To check the status of outgoing messages, the program uses the status web service provided in the ebMS plugin. This service cannot check the status of incoming messages.

The required elements in a SOAP request are the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> <MessageId></
↪tns:messageId>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The program lists the message status along with a simple description:



AS2

We created similar sample programs for AS2 as well. The programs **as2-send** and **as2-history** are used to demonstrate how to communicate with Hermes web services through AS2 SOAP messages.

Sending an AS2 message

as2-send is a sample program to demonstrate how to upload a message to Hermes using the sender web service in the AS2 plugin. You can pack your payload as a SOAP message and send it to this service with the endpoint `http://<HOST>:<PORT>/corvus/httpd/as2/sender`.

The required elements in a SOAP request are shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:as2_from xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> <as2_from> </tns:as2_
↪from>
<tns:as2_to xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> <as2_to> </tns:as2_to>
<tns:type xmlns:tns="http://service.ebms.edi.cecid.hku.hk/"> <type> </tns:type>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
.
```

```
.
.
Attached Payload
```

<as2_from> and <as2_to> contain the partnership information and <type> contains the content type of the payload.

Before sending an AS2 message, check that a partnership is registered. Please refer to the section *Maintaining Partnerships* for more information.

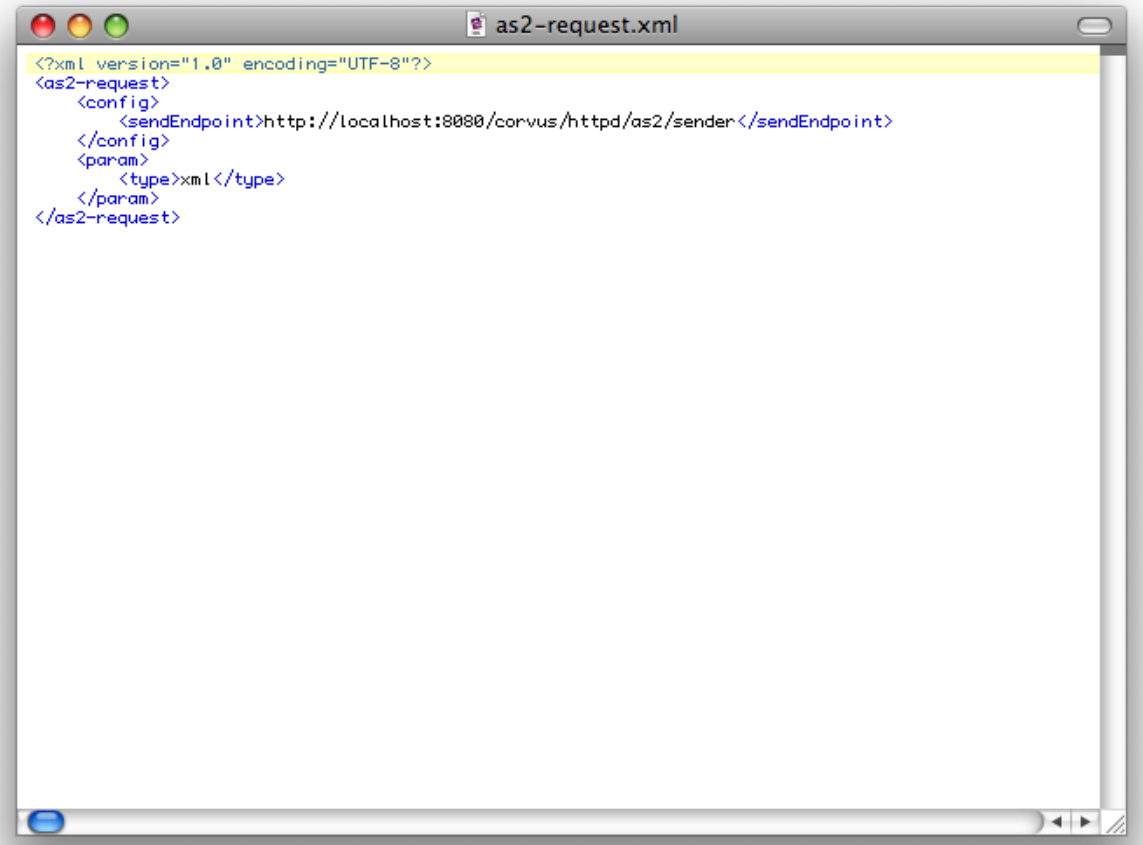
Let's take a look at the program parameters.

```
as2-send [partnership-xml] [config-xml] [log-path] [payload-path]
```

partnership-xml	The filepath of the partnership configuration file. Default is *<HERMES2_HOME>*/config/as2-partnership.xml.
config-xml	The filepath of the message configuration file. Default is *<HERMES2_HOME>*/config/as2-send/as2-request.xml.
log-path	The filepath of the logger to log query result or error. Default is *<HERMES2_HOME>*/logs/as2-send.log.
payload (optional)	The filepath of the payload attached in the message. Default is *<HERMES2_HOME>*/config/as2-send/testpayload.

config-xml

Below is sample content of the config-xml file. This file is named as2-request.xml, and placed under *<HERMES2_HOME>*/config/as2-send.



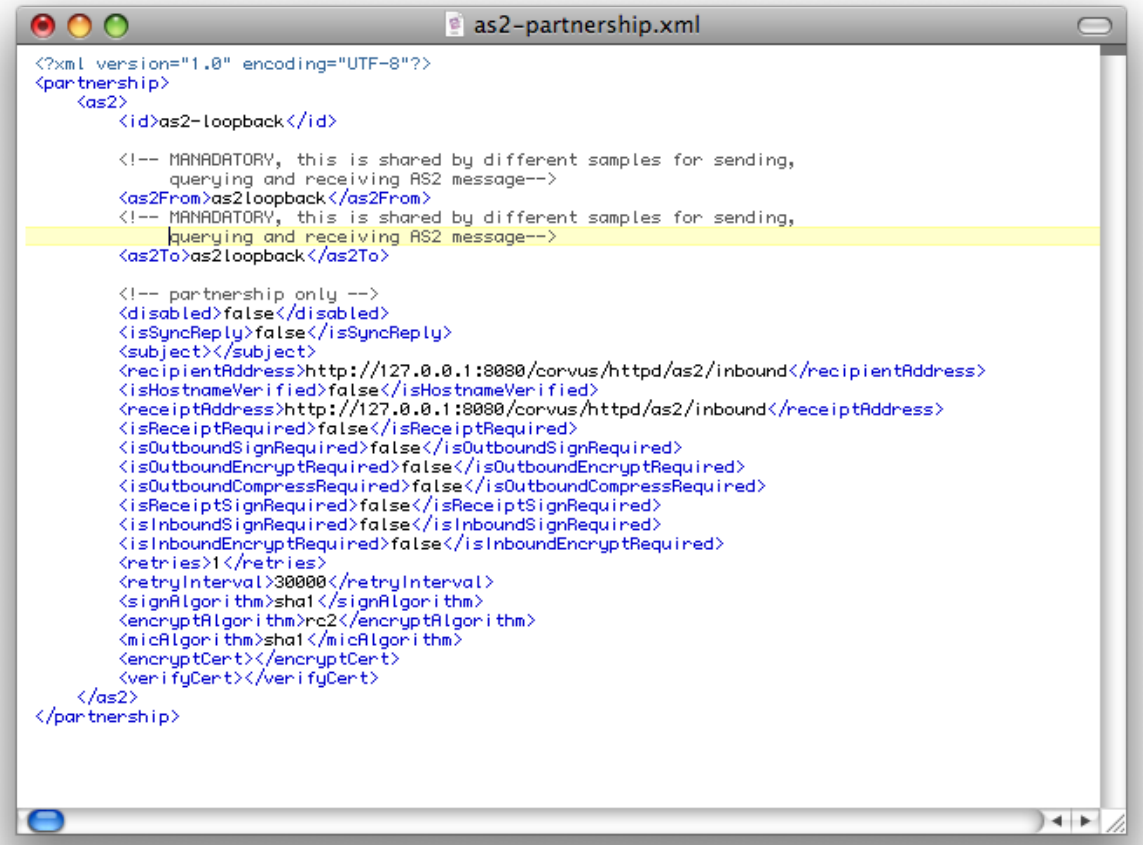
The elements are explained in the table below:

`<sendEndpoint>`	Refers to the address of the AS2 sender web service. It should be `http://<HOST>:<PORT>/corvus/httpd/as2/sender`.
`<type>`	Specify the content type. For more information, please refer to AS2 Sender Web Service. Only `<sendEndpoint>` has to be changed to contain the correct address.

partnership-xml

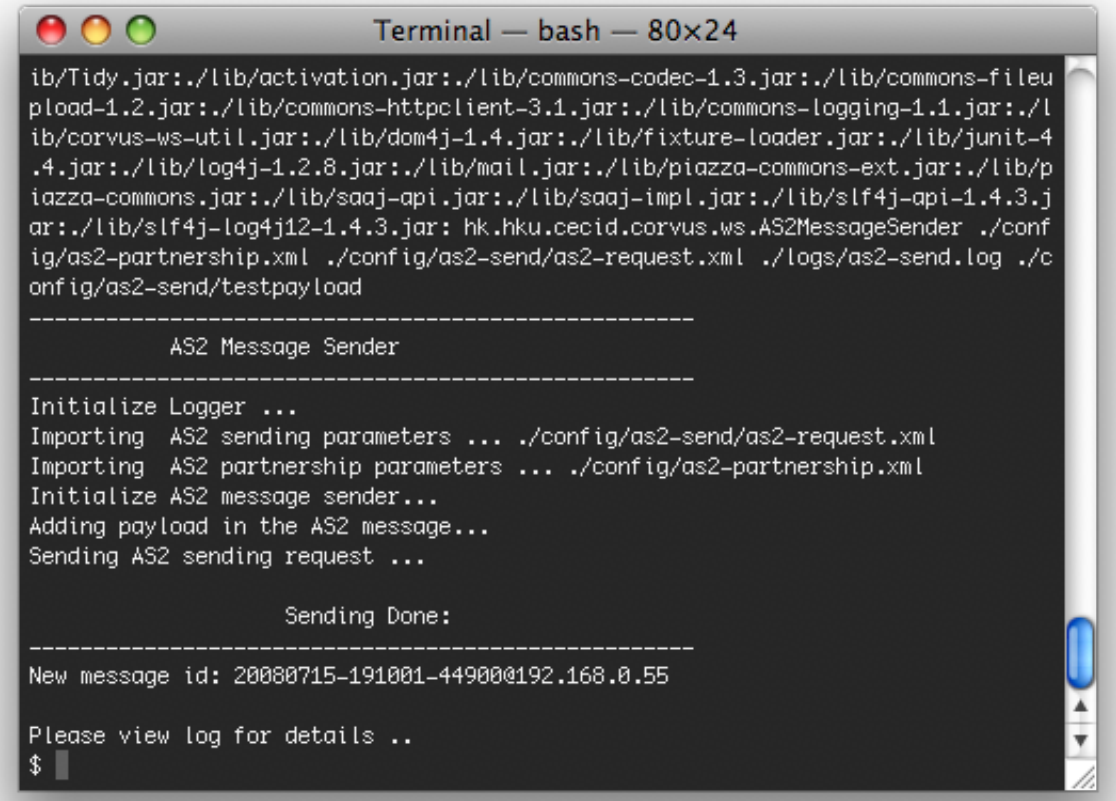
Another configuration file is the partnership-xml, which is named `as2-partnership.xml` and placed under `<HERMES2_HOME>/config` folder by default.

Sample content is shown below:



`<as2From>` and `<as2To>` are required to construct a SOAP message according to the WSDL. For more information, please refer to AS2 Partnership.

Once you have configured these parameters, you can execute the program. A message id will be returned if the program has been successfully executed. Below is sample output from the program.



AS2 history query

as2-history is a demo program that utilizes the message history web service in the AS2 plugin. The web service is called **msg-history**. There are several criteria defined for message history queries. By passing these criteria to Hermes 2 through SOAP messages, you can retrieve your target messages.

The message history web service endpoint is `http://<HOST>:<PORT>/corvus/httpd/as2/msg_history`.

The required elements in a SOAP request are shown below:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageBox xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Box]</
↪tns:messageBox>
<tns:status xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Status]</
↪tns:status>
<tns:messageId xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[Message_Id]</
↪tns:messageId>
<tns:as2From xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[AS2_From_Party]</
↪tns:as2From>
<tns:as2To xmlns:tns="http://service.ebms.edi.cecid.hku.hk/">[AS2_To_Party]</
↪tns:as2To>
```

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using this service, you can search for messages using message properties as well as partnership information.

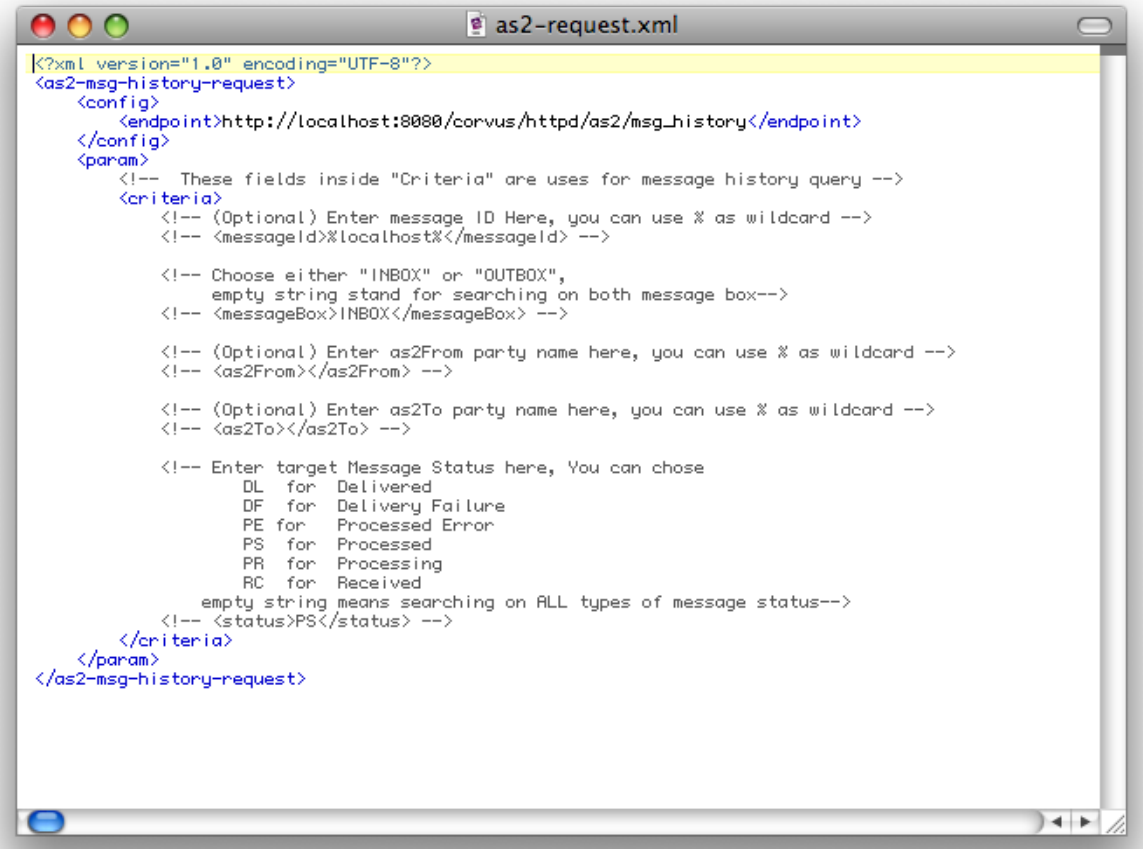
Let's take a look at the program parameters.

```
as2-history [config-xml] [log-path]
```

config-xml	The filepath of the message configuration file. Default is *<HERMES2_HOME>*/config/as2-history/as2-request.xml.
log-path	The filepath of the logger to log query result or error. Default is *<HERMES2_HOME>*/logs/as2-history.log.

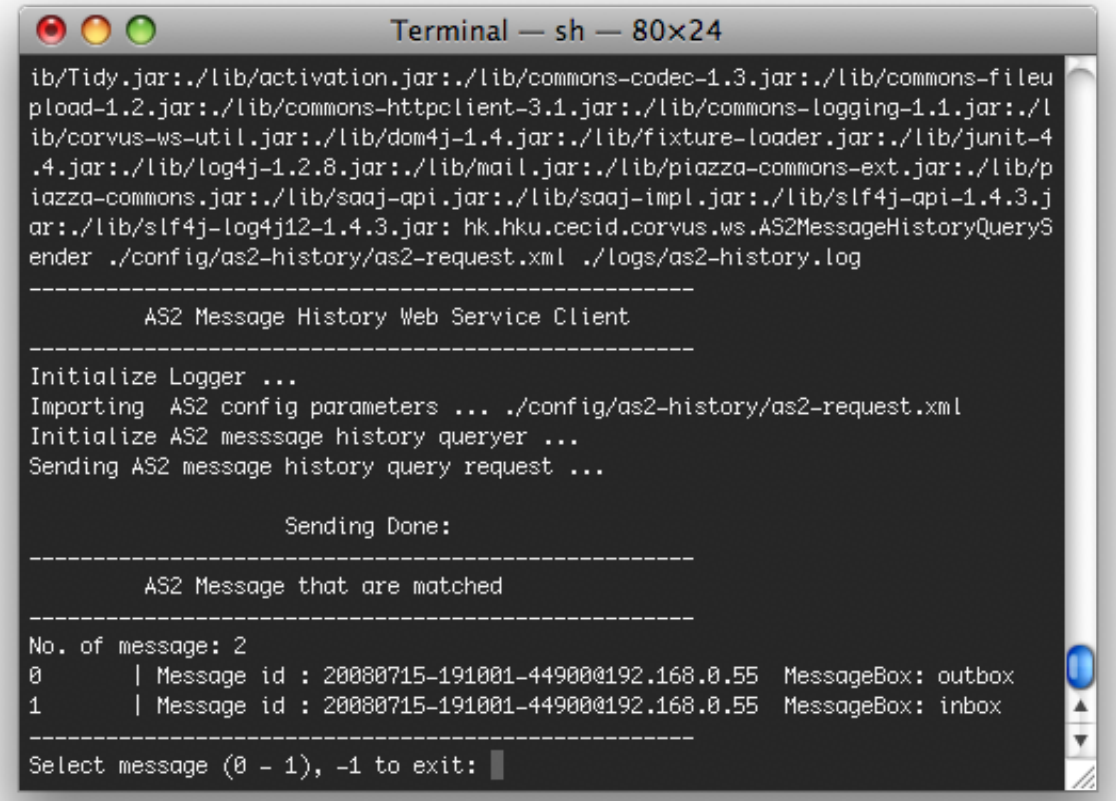
config-xml

Below is sample content of the config-xml file. This file is named as2-request.xml, and placed under *<HERMES2_HOME>*/config/as2-history. There are several elements available to use as searching criteria. You can use the wildcard character % in the values and comment out unwanted elements.



Program operation

If the query has been executed successfully, the result will be similar to following:



Messages are listed in ascending order according to the timestamp of the message (i.e. the earliest message will be listed with index 0). After the results are listed, you can choose your target message. If the message is placed under **OUTBOX**, the program will query its current status. If the message is placed under **INBOX**, the program will download the payload if available.

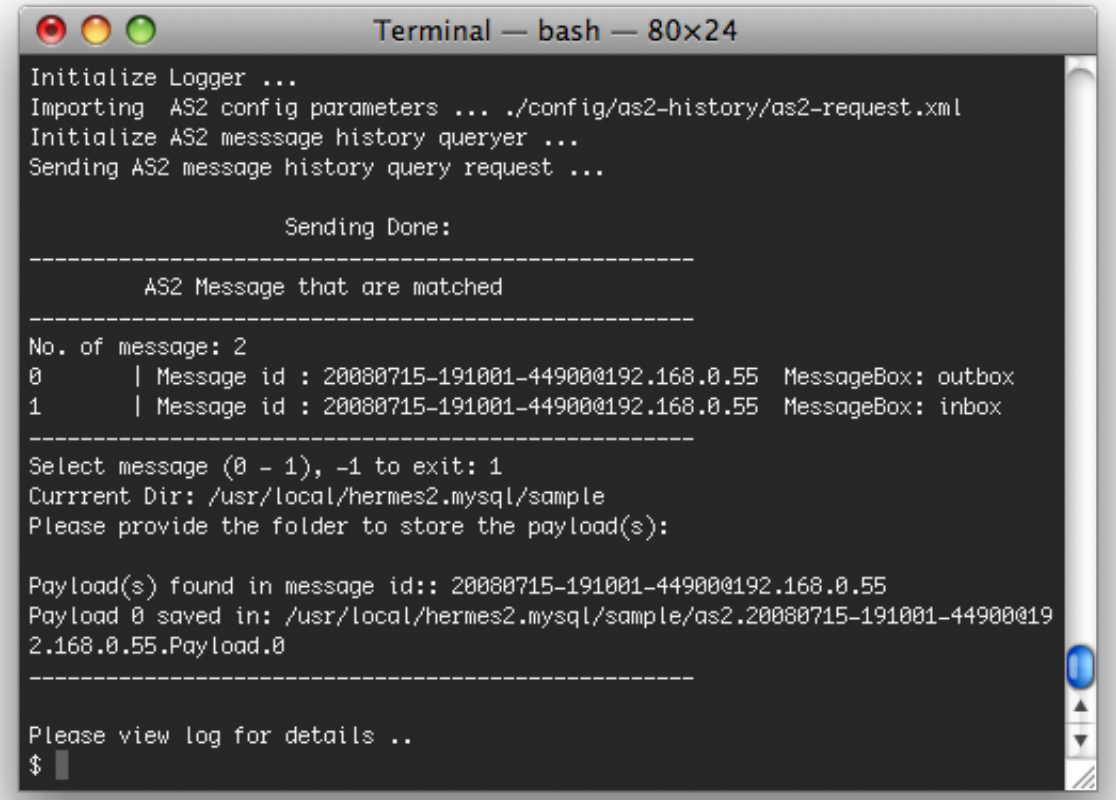
Retrieve message payload

There is a receiver service provided by the AS2 plugin for retrieving messages. The receiver service endpoint is `http://<HOST>:<PORT>/corvus/httpd/as2/receiver`.

The required elements in a SOAP request are the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageId xmlns:tns="http://service.as2.edi.cecid.hku.hk/"> [Message_Id]</
↪tns:messageId>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You can input the id of your target message in the SOAP message to retrieve its payload. However, the payload is only available once. If the payload has already been downloaded, the program will not be able to retrieve it again.



As shown above, the program will ask for a directory to store the payload(s). Each payload will be stored with the name `as2.<MessageId>.Payload.<IndexofPayload>`.

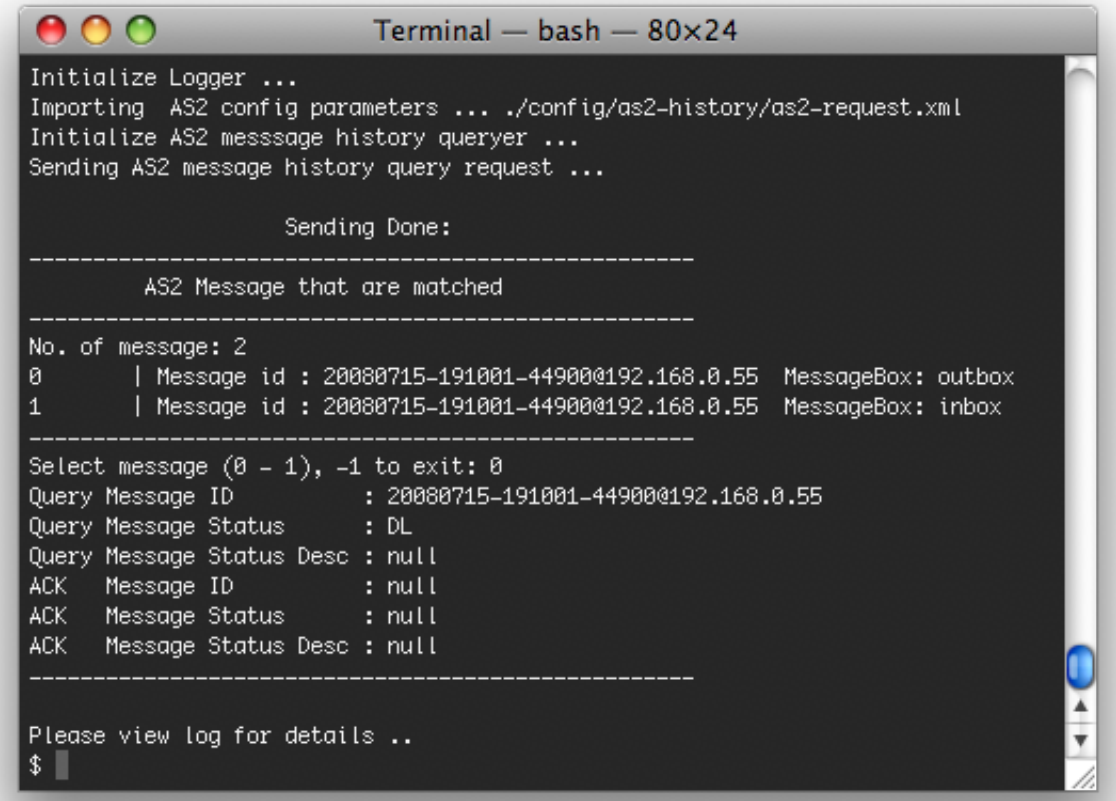
Check outgoing message status

To check the status of outgoing messages, the program uses the status web service provided in the AS2 plugin. This service cannot check the status of incoming messages.

The required elements in a SOAP request are the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<tns:messageId xmlns:tns="http://service.as2.edi.cecid.hku.hk/"> [Message_ID]</
↪tns:messageId>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here is sample output from the program:



The program will display the message status along with a simple description.

See also

- *The First Step*
- *Setting Up ebMS 2.0 Partnerships*
- *Setting Up AS2 Partnerships*
- *Using Hermes API*

Configuring Secure Messaging

Sign and Verify Message

In order to store a private key for message signing, a keystore is needed. Under current implementation, only PKCS12 keystore is supported. If Hermes was installed using the installer, there are keystore files placed in the folder called `security` under both ebMS and AS2/AS2 Plus plugins.

To enable message signing, you need to configure the plugin with a corresponding keystore. You can set the default keystore settings when running the installer or you can create a new customized keystore. To learn more about generating a keystore, please refer to *Generate a PKCS12 Keystore and Certificate*.

Sender configuration

To instruct Hermes to perform message signing with the correct private key, the corresponding Keystore Manager should be configured with the correct parameters.

Here are descriptions of the parameters:

keystore-location	Absolute file path pointing to the keystore file.
keystore-password	Password to access to keystore.
key-alias	Name of the private key.
key-password	Password to retrieve the private key. (**PKCS12** standard: `key-password` is equal to `keystore-password`)
keystore-type	The type of the keystore. This must be `PKCS12`.
keystore-provider	The class provider to handle the keystore. `org.bouncycastle.jce.provider.BouncyCastleProvider`

ebMS Sender Settings

Open the configuration file named `ebms.module.xml` that is placed in the `conf` folder of the ebMS plugin. A component named `keystore-manager-for-signature` is defined to manage the keystore.

```
<component id="keystore-manager-for-signature"
           name="Key Store Manager for Digital Signature">
    <class>hk.hku.cecid.piazza.commons.security.KeyStoreManager</class>
    <parameter name="keystore-location"
               value="/corvus/plugins/hk.hku.cecid.ebms/security/corvus.p12"␣
↪/>
    <parameter name="keystore-password" value="password" />
    <parameter name="key-alias" value="corvus" />
    <parameter name="key-password" value="password" />
    <parameter name="keystore-type" value="PKCS12" />
    <parameter name="keystore-provider"
               value="org.bouncycastle.jce.provider.BouncyCastleProvider" />
</component>
```

AS2/AS2 Plus Sender Settings

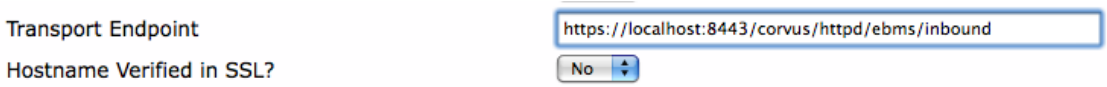
Open the configuration file named `as2.module.core.xml` that is placed in the `conf` folder of the AS2/AS2 Plus plugin. A component named `keystore-manager` is defined to manage the keystore.

```
<component id="keystore-manager" name=" AS2 Key Store Manager">
    <class>hk.hku.cecid.piazza.commons.security.KeyStoreManager</class>
    <parameter name="keystore-location" value="corvus.p12" />
    <parameter name="keystore-password" value="password" />
    <parameter name="key-alias" value="corvus" />
    <parameter name="key-password" value="password" />
    <parameter name="keystore-type" value="PKCS12" />
    <parameter name="keystore-provider"
               value="org.bouncycastle.jce.provider.BouncyCastleProvider" />
</component>
```

Receiver configuration

For a receiver to verify the signature, a public certificate should be provided by the sender through the partnership maintenance page.

Set the value of *Signing Required* to `true`. For detailed settings of the partnership, please refer to *Setting Up AS2 Partnerships* or *Setting Up ebMS 2.0 Partnerships*.



Send Messages Through HTTPS

SSL server authentication

To enable server authentication in Tomcat, a truststore and a keystore have to be configured in Hermes and Tomcat respectively.

On the sending side, a truststore is defined in `corvus.properties.xml`, which is where the certificates of trusted servers are stored. When the sending Hermes tries to establish a secure connection, the receiving Hermes will provide a public certificate for the sender to identify their identity. If this certificate is self-signed, it should be added to the truststore defined on the sending side.

On the receiving side, a keystore is defined in the `server.xml` of Tomcat. The keystore contains its paired private key and public certificate. If the keystore is self-signed, the certificate has to be exported, then imported to the trustore of the sending Hermes.

The details of this procedure are shown below. For information about how to create a keystore and generate a public certificate, please refer to the section *Generate a PKCS12 Keystore and Certificate*.

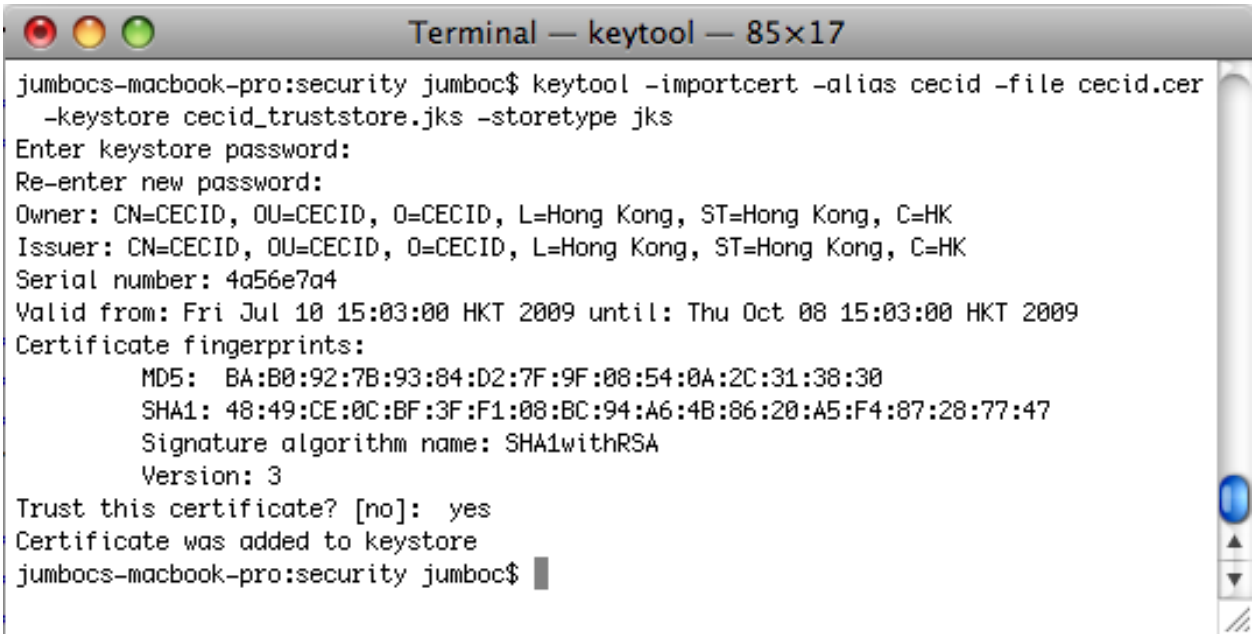
Sender configuration

As mentioned before, a truststore needs to be configured. In this example, a JKS keystore is used as a truststore as it is much simpler to import a self-signed certificate.

If there is no keystore file found, **`Keytool`** can be used to create a new keystore:

```
keytool -importcert -file {filepath-and-name-of-certificate} -alias {key-alias} -
↪keystore {filepath-and-name-of-keystore} -storetype jks -storepass {password}
```

The program will display the certificate information and ask for confirmation. Enter `yes` after verifying the details.



Open `corvus.properties.xml`. The definition of the truststore can be found under the `environment` component.

Here are descriptions of the parameters:

`trustStore`	The absolute file path to the keystore.
`trustStorePass`	The password to access the keystore.
`trustStoreType`	The type of the keystore. Both PKCS12 and JKS are supported.

If asynchronous replies are enabled for the receiving partnership, the same configuration needs to be made for Hermes on both sides, however the roles are reversed.

Receiver configuration

Once a keystore has been created, `server.xml` needs to be modified to specify the keystore parameters.

1. Uncomment the connector definition on port `8443`.
2. Add the following attributes for keystore configuration.

`keystoreFile`	An absolute file path to the keystore file.
`keystorePass`	The password to access the keystore.
`keystoreType`	The type of keystore. Both PKCS12 and JKS are supported.
`keyalias`	Optional. If the keystore contains more than one key pair, specify the target key-pair with an alias.
`clientAuth`	Set this to `false` to indicate only Server Authentication is needed.

SSL client authentication

In addition to server authentication, client authentication can also be applied to Hermes to achieve secure connections for message deliveries.

Once the server authentication is complete, the receiving Hermes will ask for the identity of the sending Hermes. The sender will provide a public certificate to the receiver, which will be compared to the trusted certificates in the truststore.

Sender configuration

To store the private key and public certificate pair that identifies the sender, a keystore is needed.

Here are descriptions of the parameters:

`javax.net.ssl.keyStore`	The absolute file path to the keystore.
`javax.net.ssl.keyStorePassword`	The password to access the keystore.
`javax.net.ssl.keyStoreType`	The type of the keystore.

Receiver configuration

In order to store trusted certificates, a truststore needs to be declared in the `server.xml` of Tomcat.

Here are descriptions of the attributes:

`keystoreFile`	The absolute file path to the keystore.
`keystorePass`	The password to access the keystore.
`keystoreType`	The type of the keystore. Both PKCS12 and JKS are supported.
`clientAuth`	Set this to `true` to enforce client authentication.

Generate a PKCS12 Keystore and Certificate

To create a keystore and certificate, **`Keytool`** or **`OpenSSL`** can be used.

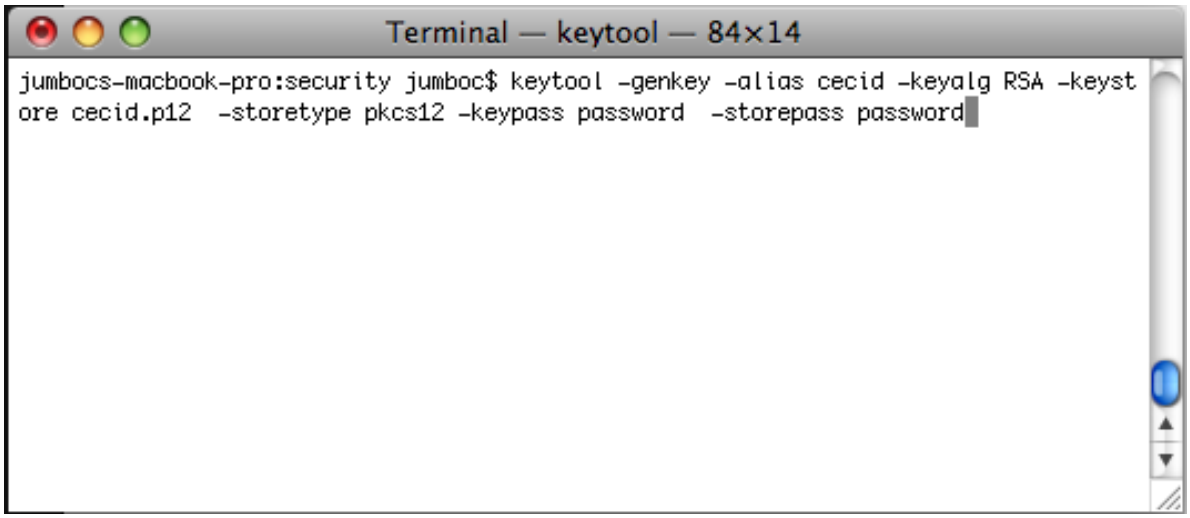
Using Keytool

`Keytool` is provided with Java SDK.

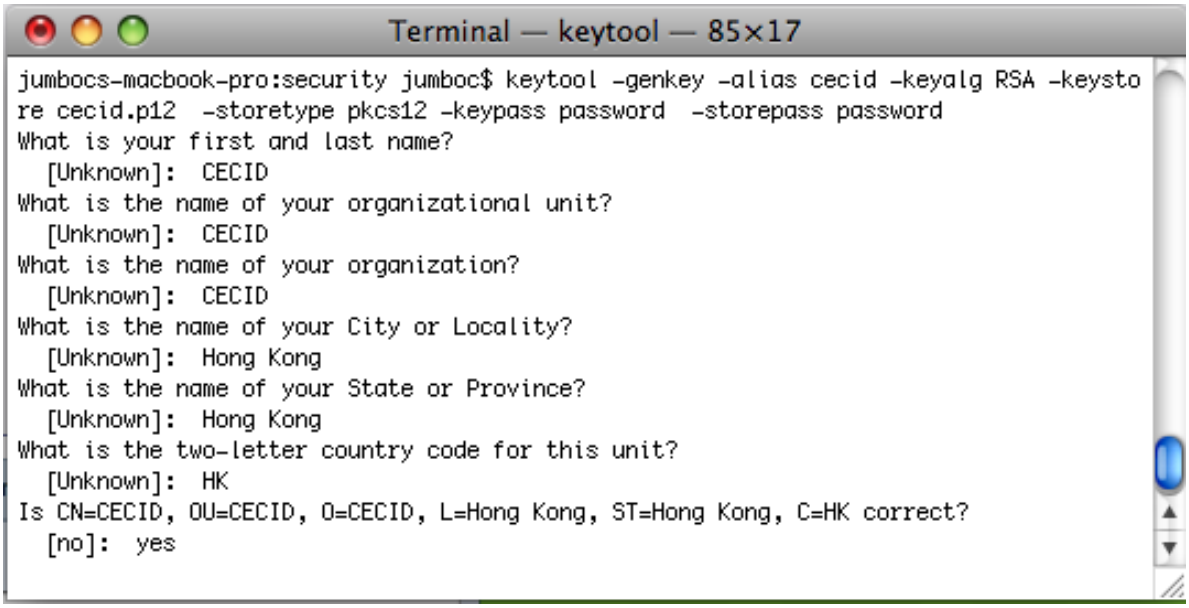
1. Invoke `keytool` with parameters.

```
keytool -genkey -alias {key-alias} -keyalg RSA -keystore {filepath-and-name-of-
↪keystore} -storetype pkcs12 -storepass {password} -keypass {password}
```

The same password value is used for `keypass` and `storepass` in this command.



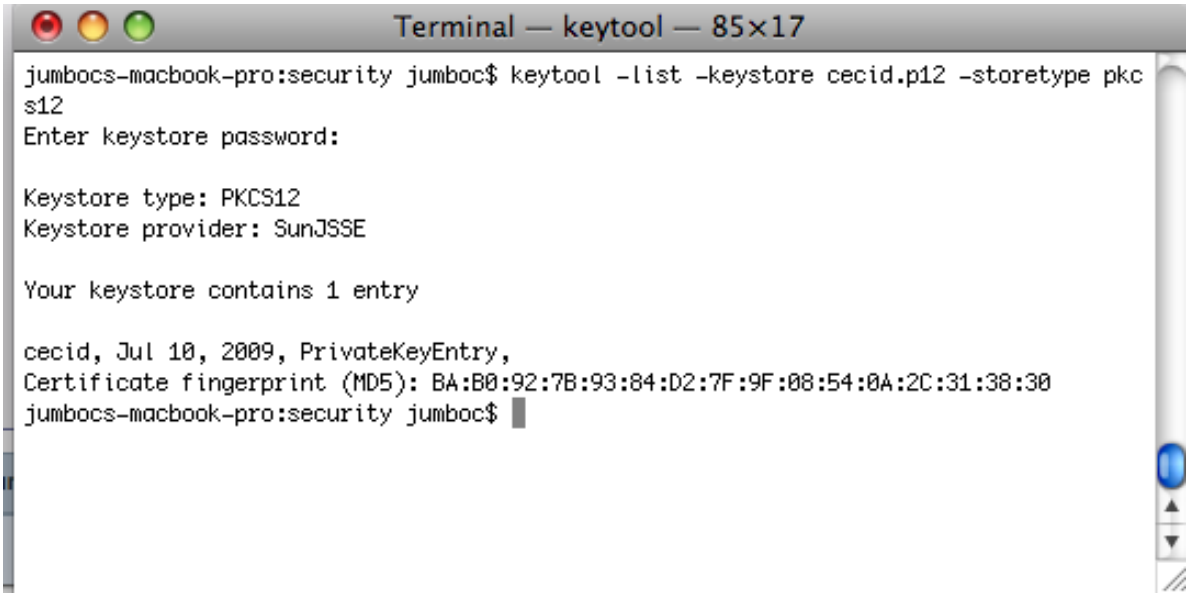
2. Input more detailed information.



After entering the information, a keystore will be created. It can be verified using **`Keytool`**.

```
keytool -list -keystore {filepath-and-name-of-keystore} -storetype pkcs12
```

The password specified in the `storepass` attribute is needed to access the keystore.

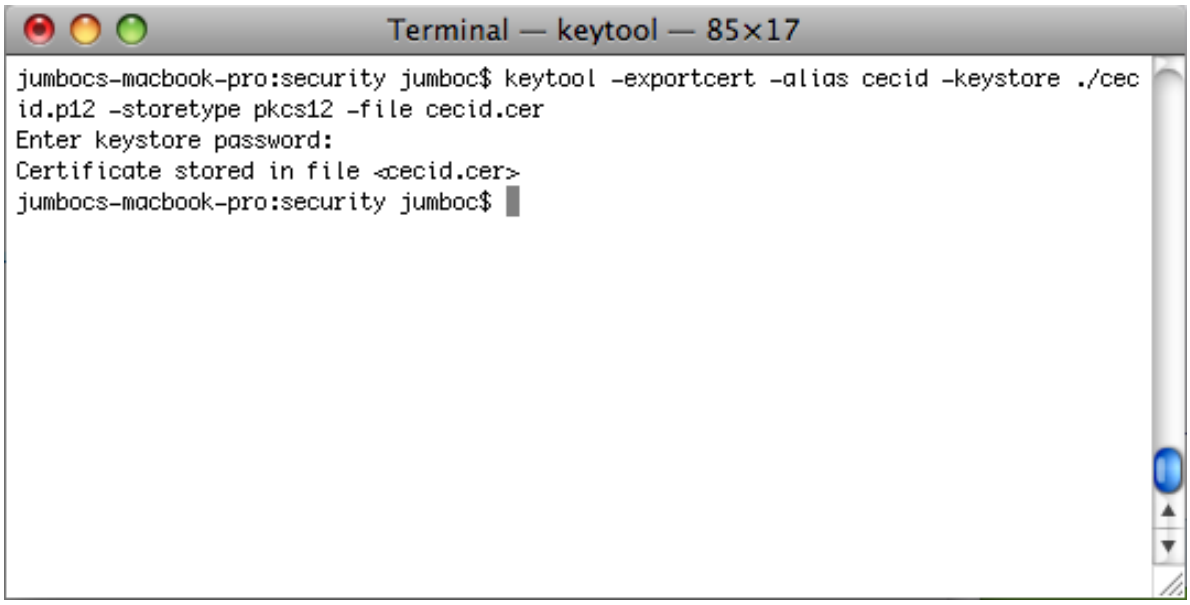


3. Export certificate.

   The private key has been generated and stored in the keystore, but a public certificate is still needed for the receiver to verify signatures.

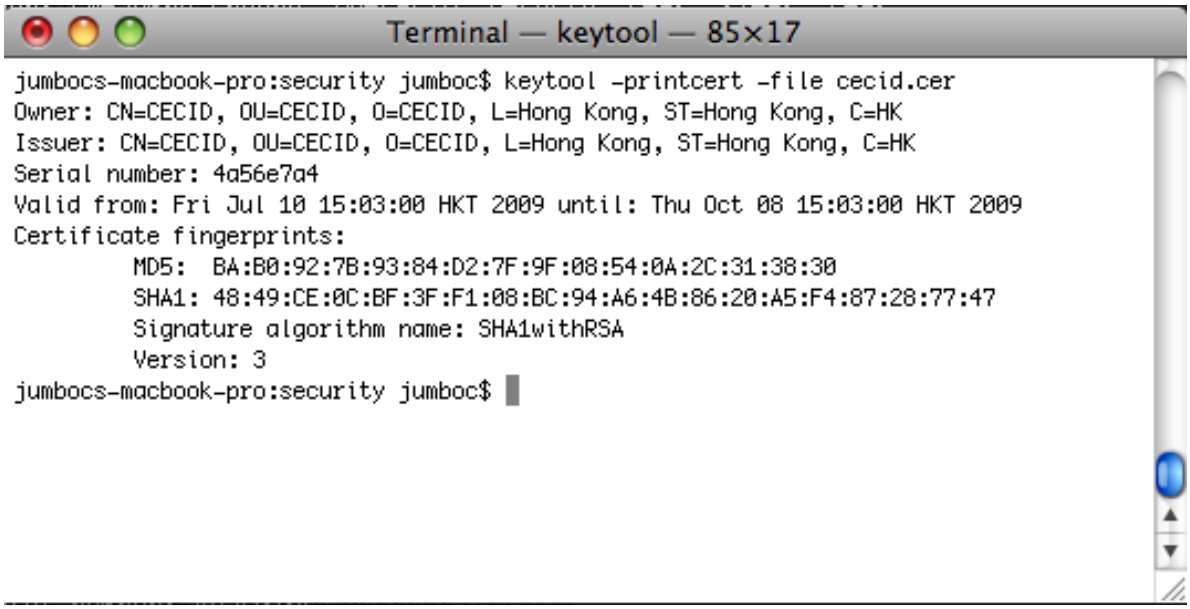
```
keytool -exportcert -alias {key-alias} -keystore {filepath-and-name-of-keystore} -
↪storetype pkcs12 -file {filepath-and-name-of-certificate}
```

   Enter the password specified in the `storepass` attribute to access the keystore.



The certificate can be verified with the following command:

```
keytool -printcert -file {filepath-and-name-of-certificate}
```



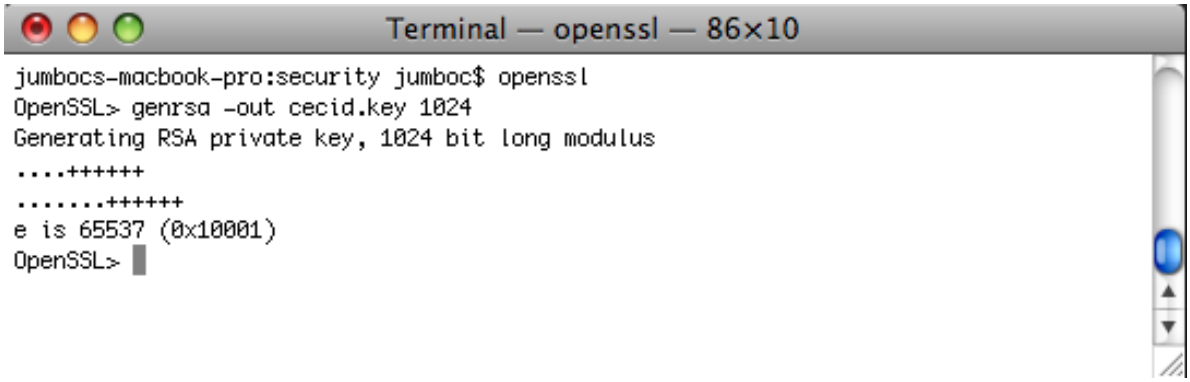
Using OpenSSL

OpenSSL can be found here.

1. Generate private key.

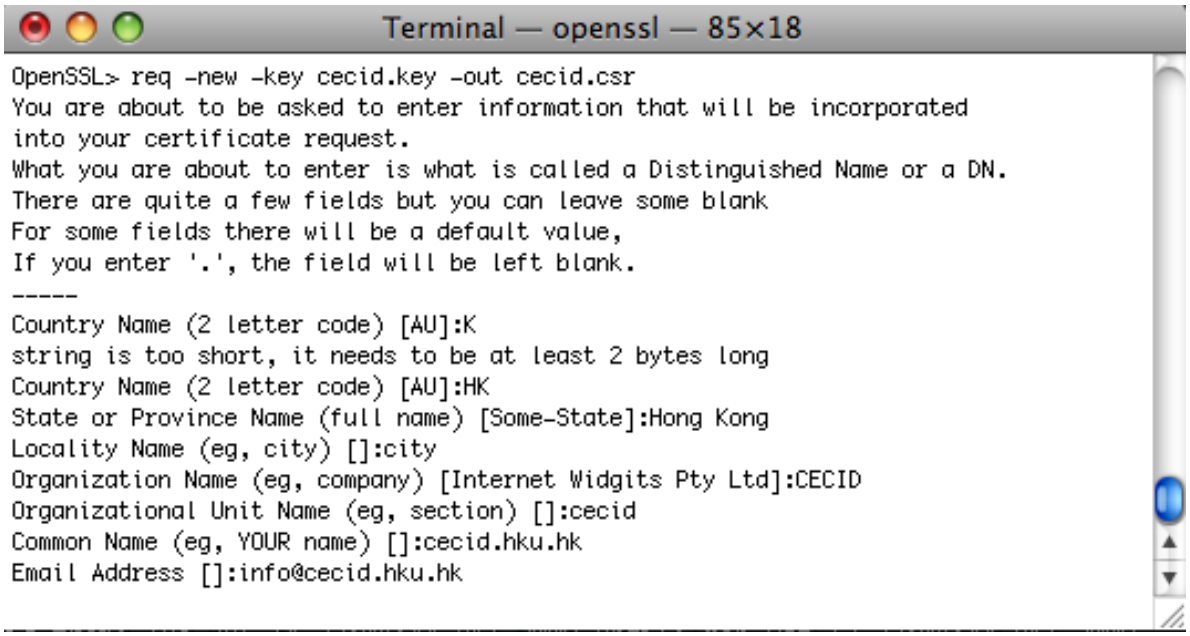
   Invoke `openssl` to enter the **OpenSSL** environment, then execute the following:

   ```
   genrsa -out {filepath-and-name-of-key} {length-in-bits}
   ```



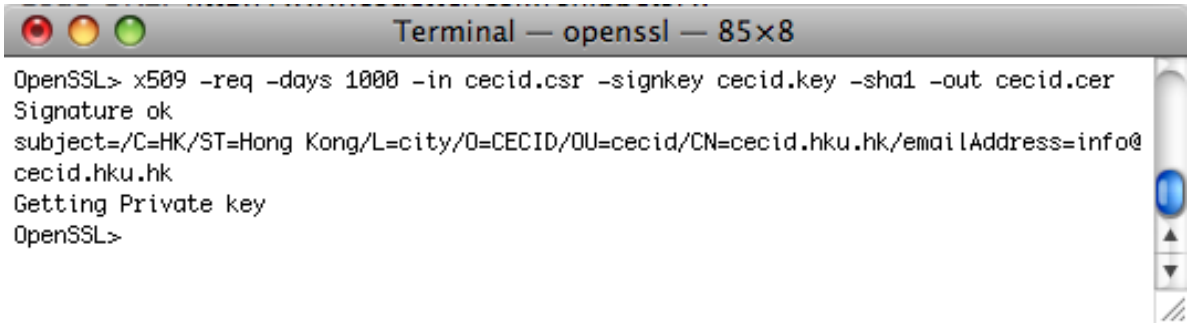
2. Generate certificate signing request.

```
req -new -key {filepath-and-name-of-key} -out {filepath-and-name-of-signing-
↪request}
```



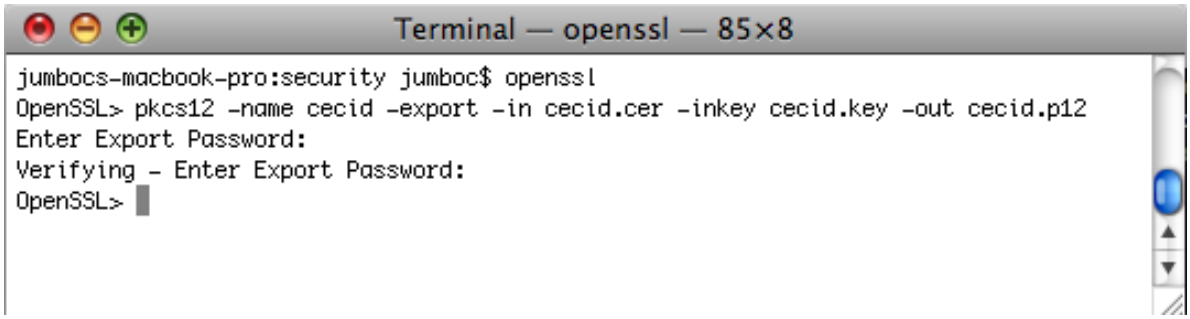
3. Generate self-signed certificate.

```
x509 -req -days {number-of-days-valid} -in {filepath-and-name-of-signing-request}␣
↪-signkey {filepath-and-name-of-key} -sha1 -out {filepath-and-name-of-
↪certificate}
```



4. Export to keystore in PKCS12 format.

```
pkcs12 -name {key-alias} -export -in {filepath-and-name-of-certificate} -inkey
↪{filepath-and-name-of-key} -out {filepath-and-name-of-keystore}
```



Supported Parameters

The following key pair algorithms and signature algorithms have been tested:

Key pair algorithm (`keyalg`)

`DSA`	Generates keypairs for the Digital Signature Algorithm
`RSA`[1]	Generates keypairs for the RSA algorithm (Signature/Cipher)

[1] `RSA` has been test with `keysize`= 1024, 2048, 4096.

Signature algorithm (`sigalg`)

SHA1withRSA	The signature algorithm with SHA-* and the RSA encryption algorithm as defined in the OSI Interoperability Workship, using the padding conversions described in PKCS1.
SHA256withRSA	
SHA512withRSA	
MD5withRSA	The MD2/MD5 with RSA encryption algorithm which users the MD2/MD5 digest algorithm and RSA to cread and verify RSA digital signatures as defined in PKCS1.
MD2withRSA	
SHA1withDSA	DSA with SHA-1 signature algorithm which uses the SHA-1 digest algorithm and DSA to create and verify DSA digital signatures as defined in FIPS PUB 186.

Parameter combinations

The following combinations of algorithms and parameters have been tested with ebMS and AS2:

ebMS

tool		Keytool			OpenSSL		
keysize		1024	2048	4096	1024	2048	4096
RSA	SHA1	ok	ok	ok	ok	ok	ok
	SHA256	ok	ok	ok	ok	ok	ok
	MD5	ok	ok	ok	ok	ok	ok
	SHA512	not supported	ok	not supported	not supported	ok	not supported
	MD2	not supported	ok	not supported	not supported	not supported	not supported
DSA	SHA1	ok	not supported	not supported	ok	not supported	not supported

AS2

tool		Keytool			OpenSSL		
keysize		1024	2048	4096	1024	2048	4096
RSA	SHA1	ok	ok	ok	ok	ok	ok
	SHA256	ok	ok	ok	ok	ok	ok
	MD5	ok	ok	ok	ok	ok	not supported
	SHA512	not supported	ok	not supported	not supported	ok	not supported
	MD2	not supported	ok	not supported	not supported	not supported	not supported
DSA	SHA1	not supported	not supported	not supported	not supported	not supported	not supported

See also

- Wiki Public Key Infrastructure (Wiki)
- Public Key Infrastructure (FreeMagazine)

Developing Hermes Applications

Overview

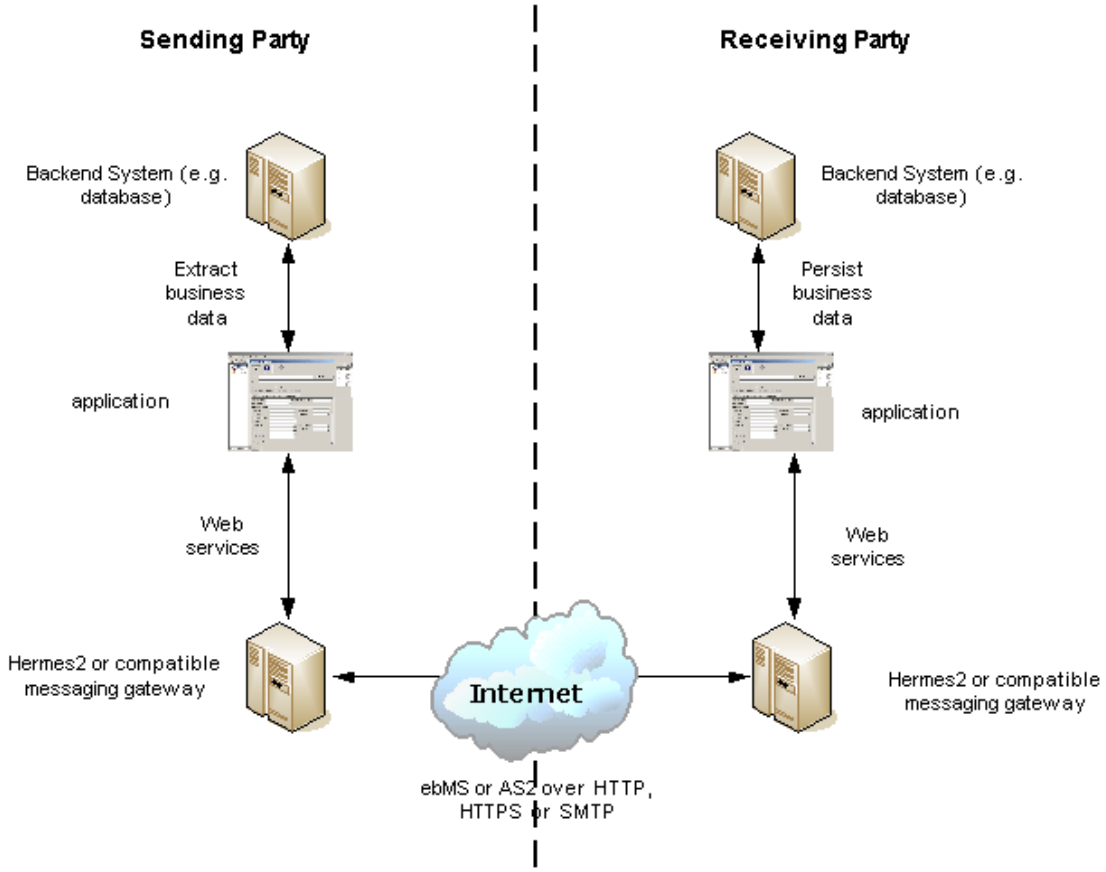
This application development guide provides guidelines for developing Hermes messaging applications, which exchange messages reliably and securely with Hermes. A Hermes server provides web services for an application to communicate with another Hermes server. These web services allow the application to:

- Request Hermes to send a payload to another Hermes or a compatible messaging gateway on the receiver party;
- Retrieve the message identifier of a received message which has not retrieved yet;

- Retrieve the payloads of a message which is identified by its message identifier; and
- Obtain the message status of an outgoing or incoming message.

For information about installing Hermes and communicating with Hermes using an external application, please refer to *Installing Hermes* and *Using Hermes API*.

Application Integration with Hermes



The above figure shows a typical architecture for integrating client applications with Hermes servers. Two parties exchange business messages with Hermes through ebMS 2.0 or AS2 over a transport protocol, such as HTTP, HTTPS or SMTP.

On the sender side, a backend system produces business data to be transferred to the receiver party. The sender application extracts the data from the backend system and submit them to Hermes through a web service. The sender's Hermes sends the data as payloads in a business message. Then, the application invokes a web service to check whether the message is delivered successfully.

On the receiver side, Hermes receives the message. The receiver application retrieves the message from Hermes, extracts the payloads from the message, and stores them into the receiver backend system.

The application interacts with Hermes using web services, i.e., SOAP or RESTful APIs. The benefits of using web services generally apply. For example,

- **Implementation-independent.** Since the application interacts with Hermes using web services, the application can be implemented in any programming languages, as long as web services are supported.

- **Firewall-friendly.** The web services provided by Hermes use HTTP as the transport protocol. The application calls Herms with web services over HTTP; this way, persistant and stateful connectivity between Hermes and the application is not required. Even when there is a firewall between Hermes and the application, the application can communicate with Hermes as long as HTTP connections between them are not blocked.

Prerequisites

The source code shown below comes the `Hermes loopback test`. The sample code assumes that Hermes is using `localhost` with port `8080` (i.e., the default port of Tomcat).

The sample code requires the following libraries:

- `activation.jar`
- `mail.jar`

Import Java Packages in Web Service Client

You are required to import the following Java packages to program a SOAP web service:

```
import java.net.URL;
import java.net.MalformedURLException;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPMessage;
```

Writing ebMS Messaging Client

Send ebMS Message

We need to create a SOAP message with the following 10 paremeters and send it to Hermes as a web service request.

- `cpaId`: Copllaboration Protocol Agreement ID
- `service`: Service
- `action`: Action
- `convId`: Conversation ID
- `fromPartyId`: Sender party ID
- `fromPartyType`: Sender party type
- `toPartyId`: Receiver party ID
- `toPartyType`: Receiver party type
- `refToMessageId`: Refer-to message Id

- `serviceType`: Service type

1. Define a namespace URI and prefix conforming to the WSDL, and define the endpoint URL of the ebMS sender web service.

```
private String nsURI = "http://service.ebms.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL senderWSURL = "http://localhost:8080/corvus/httpd/ebms/sender";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters. For example:

```
<cpaId> ebmscpaid </cpaId>
<service> http://localhost:8080/corvus/httpd/ebms/inbound <service>
<action> action </action>
<convId> convId </convId>
<fromPartyId> fromPartyId </fromPartyId>
<fromPartyType> fromPartyType </fromPartyType>
<toPartyId> toPartyId </toPartyId>
<toPartyType> toPartyType </toPartyType>
<refToMessageId> </refToMessageId>
<serviceType> </serviceType>
```

Compose a sample SOAP request to send an ebMS message:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("cpaId", nsPrefix, nsURI, cpaId));
soapBody.addChildElement(createElement("service", nsPrefix, nsURI, service));
soapBody.addChildElement(createElement("action", nsPrefix, nsURI, action));
soapBody.addChildElement(createElement("convId", nsPrefix, nsURI,␣
↪conversationId));
soapBody.addChildElement(createElement("fromPartyId", nsPrefix, nsURI,␣
↪fromPartyId));
soapBody.addChildElement(createElement("fromPartyType", nsPrefix, nsURI,␣
↪fromPartyType));
soapBody.addChildElement(createElement("toPartyId", nsPrefix, nsURI, toPartyId));
soapBody.addChildElement(createElement("toPartyType", nsPrefix, nsURI,␣
↪toPartyType));
soapBody.addChildElement(createElement("refToMessageId", nsPrefix, nsURI,␣
↪refToMessageId));
soapBody.addChildElement(createElement("serviceType", nsPrefix, nsURI,␣
↪serviceType));
```

The method `createElement` creates a SOAP element with the namespace prefix `nsPrefix`, the namespace URL `nsURI`, and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Attach a payload to the SOAP message if necessary. The example here uses a purchase order XML as the payload, so the associated content type is `application/xml`.

```
AttachmentPart attachmentPart = request.createAttachmentPart();
FileDataSource fileDS = new FileDataSource(new File("purchase_order.xml"));
attachmentPart.setDataHandler(new DataHandler(fileDS));
attachmentPart.setContentType("application/xml");
request.addAttachmentPart(attachmentPart);
```

5. Save the changes to the SOAP message.

```
request.saveChange();
```

6. Send the SOAP request to Hermes and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, senderWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

7. Process the SOAP response, extract the identifier of the requested ebMS message, and print it to `System.out` if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement messageIdElement = getFirstChild(responseBody, "message_id",␣
↪nsURI);
    System.out.println(messageIdElement == null ? null : messageIdElement.
↪getValue());
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name `message_id` and the namespace URI `nsURI`. An existing `message_id` is a registered identifier, showing the message has been successfully submitted to Hermes.

Hermes translates the SOAP request is now transformed into an ebMS message and saves it in its persistent storage. Then, the sender Hermes delivers the ebMS message to the receiver Hermes, which is specified in the SOAP request parameters, of which `cpaId`, `service` and `action` identify the partnership between the sender and receiver.

List ebMS Messages

We need to create a SOAP message with the following 9 parameters and send it to Hermes as a web service request.

- `cpaId`: Copllaboration Protocol Agreement ID
- `service`: Service
- `action`: Action
- `convId`: Conversation ID
- `fromPartyId`: Sender party ID
- `fromPartyType`: Sender party type
- `toPartyId`: Receiver party ID
- `toPartyType`: Receiver party type
- `numOfMessages`: Number of messages

1. Define a namespace URI and prefix conforming to the WSDL.

```
private String nsURI = "http://service.ebms.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private String URL receiverListWSURL = "http://localhost:8080/corvus/httpd/ebms/
↪receiver_list";
```

2. Create a SOAP message factory and an SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters. For example:

```
<cpaId> ebmscpaid </cpaId>
<service> http://localhost:8080/corvus/httpd/ebms/inbound <service>
<action> action </action>
<convId> convId </convId>
<fromPartyId> fromPartyId </fromPartyId>
<fromPartyType> fromPartyType </fromPartyType>
<toPartyId> toPartyId </toPartyId>
<toPartyType> toPartyType </toPartyType>
<numOfMessages> 100 </numOfMessages>
```

Compose a sample SOAP request to list ebMS messages:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("cpaId", nsPrefix, nsURI, cpaId));
soapBody.addChildElement(createElement("service", nsPrefix, nsURI, service));
soapBody.addChildElement(createElement("action", nsPrefix, nsURI, action));
soapBody.addChildElement(createElement("convId", nsPrefix, nsURI,␣
↪conversationId));
soapBody.addChildElement(createElement("fromPartyId", nsPrefix, nsURI,␣
↪fromPartyId));
soapBody.addChildElement(createElement("fromPartyType", nsPrefix, nsURI,␣
↪fromPartyType));
soapBody.addChildElement(createElement("toPartyId", nsPrefix, nsURI, toPartyId));
soapBody.addChildElement(createElement("toPartyType", nsPrefix, nsURI,␣
↪toPartyType));
soapBody.addChildElement(createElement("numOfMessages", nsPrefix, nsURI,␣
↪numOfMessages));
```

The method `createElement` creates a SOAP element with the namespace `nsPrefix`, the namespace URL and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, receiverListWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

6. Here is the SOAP response:

```
<soap-body>
    <messageIds>
        <messageId> ... </messageId>
        <messageId> ... </messageId>
        <messageId> ... </messageId>
        <messageId> ... </messageId>
    </messageIds>
</soap-body>
```

Process the SOAP response, extract the identifier of each requested message, and print it to `System.out` if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement messageIdsElement = getFirstChild(responseBody, "messageIds",␣
↪nsURI);
    Iterator messageIdElementIter = getChildren(messageIdsElement, "messageId",␣
↪nsURI);

    while(messageIdElementIter.hasNext()) {
        SOAPElement messageIdElement = (SOAPElement)messageIdElementIter.next();
        System.out.println(messageIdElement.getValue());
    }
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name :code‘messageIds‘ and namespace `nsURI`. It then extracts every `messageId` which represents an available message awaiting a further action.

Retrieve ebMS Message

We need to create a SOAP message with the identifier of the target message and send it to Hermes as the web service request.

1. Define a namespace URI and prefix conforming to the WSDL.

```
private String nsURI = "http://service.ebms.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL receiverWSURL = "http://localhost:8080/corvus/httpd/ebms/receiver";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<messageId> messageId </messageId>
```

Compose a sample SOAP request to retrieve an ebMS message:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("messageId", nsPrefix, nsURI, messageId));
```

The method `createElement` creates a SOAP element with the namespace prefix `nsPrefix`, the namespace URL `nsURI`, and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes and get a SOAP response.

```
SOAPMessage response = soapConn.call(request, receiverWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

Here is the SOAP response:

```
<soap-body>
    <hasMessage> ... </hasMessage>
</soap-body>
      .
      .
The attachment is formatted as a MIME part.
```

Process the SOAP response, and extract the payloads from the received ebMS message if available.

```
if (!responseBody.hasFault()){
    SOAPElement hasMessageElement = getFirstChild(responseBody, "hasMessage",␣
↪nsURI);
    ArrayList payloadsList = new ArrayList();
    if (hasMessageElement != null){
        Iterator attachmentPartIter = response.getAttachments();
        while(attachmentPartIter.hasNext()) {
            AttachmentPart attachmentPart = (AttachmentPart) attachmentPartIter.
↪next();
            InputStream ins = attachmentPart.getDataHandler().getInputStream();
            // Do something I/O to extract the payload to physical file.
        }
    }
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name `hasMessage` and the namespace URI `nsURI`. The boolean value of `hasMessage` represents the existence of a payload in this message.

The payload is extracted from the attachment part, and written to the input stream. This way, the data can be piped to a processor or saved as a file.

Get ebMS Message Status

We need to create a SOAP message with the identifier of the target message and send it to Hermes as the web service request.

1. Define a namespace URI and prefix conforming to the WSDL.

```
private String nsURI = "http://service.ebms.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL statusQueryWSURL = "http://localhost:8080/corvus/httpd/ebms/status";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<messageId> messageId </messageId>
```

Compose a sample SOAP request to get the status of an ebMS message:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("messageId", nsPrefix, nsURI, messageId));
```

The method `createElement` creates a SOAP element with the namespace prefix `nsPrefix`, the namespace URL `nsURI` and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes and get a SOAP response.

```
SOAPMessage response = soapConn.call(request, statusQueryWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

6. Here is the SOAP response:

```
<soap-body>
    <MessageInfo>
        <status> The current status of message </status>
        <statusDescription> The current status description of message </
↪statusDescription>
        <ackMessageId> The message id of acknowledgment / receipt if any </
↪ackMessageId>
        <ackStatus> The status of acknowledgment / receipt if any </ackStatus>
        <ackStatusDescription> The status description of acknowledgment / receipt␣
↪if any </ackStatusDescription>
    </MessageInfo>
</soap-body>
```

Process the SOAP response and extract the status information from the ebMS message if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement messageInfoElement = getFirstChild(responseBody, "messageInfo",␣
↪nsURI);
    System.out.println("Message Status : " + getFirstChild(messageInfoElement,
↪"status", nsURI);
```

```
    System.out.println("Message Status Desc : " +␣
↪getFirstChild(messageInfoElement, "statusDescription", nsURI);
    System.out.println("Ack Message Identifiers : " +␣
↪getFirstChild(messageInfoElement, "ackMessageId", nsURI);
    System.out.println("Ack Status : " + getFirstChild(messageInfoElement,
↪"ackStatus", nsURI);
    System.out.println("Ack Status Desc : " + getFirstChild(messageInfoElement,
↪"ackStatusDescription", nsURI);
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name `messageInfo` and the namespace URI `nsURI`. It then retrieves the status value from that element.

Get ebMS Message History

We need to create a SOAP message with the following 7 parameters and send it to Hermes as a web service request.

- `messageId`: Message ID
- `messageBox`: Message Box
- `conversationId`: Conversation ID
- `cpaId`: Collaboration Protocol Agreement ID
- `status`: Status
- `action`: Action
- `service`: Service

1. Define a namespace URI and prefix conforming to the WSDL.

```
private String nsURI = "http://service.ebms.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL msgHistoryWSURL = "http://localhost:8080/corvus/httpd/ebms/msg_history
↪";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<messageId> messageId </messageId>
<messageBox> messageBox </messageBox>
<conversationId> conversationId </conversationId>
<cpaId> cpaId </cpaId>
<service> service </service>
<action> action </action>
<status> status </status>
```

Compose a sample SOAP request to get an ebMS message history:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("messageId", nsPrefix, nsURI, messageId));
soapBody.addChildElement(createElement("messageBox", nsPrefix, nsURI,␣
↪messageBox));
```

```
soapBody.addChildElement(createElement("conversationId", nsPrefix, nsURI,␣
↪conversationId));
soapBody.addChildElement(createElement("cpaId", nsPrefix, nsURI, cpaId));
soapBody.addChildElement(createElement("service", nsPrefix, nsURI, service));
soapBody.addChildElement(createElement("fromPartyType", nsPrefix, nsURI,␣
↪fromPartyType));
soapBody.addChildElement(createElement("action", nsPrefix, nsURI, action));
soapBody.addChildElement(createElement("status", nsPrefix, nsURI, status));
```

The method `createElement` creates a SOAP element with the namespace `nsPrefix`, the namespace URL `nsURI`, and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, msgHistoryWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

6. Here is the SOAP response:

```
<soap-body>
    <messageList>
        <messageElement>
            <messageId> Message ID of this message </messageId>
            <messageBox> Message Box containing this message </messageBox>
        </messageElement>
        <messageElement>
            <messageId> Message ID of this message </messageId>
            <messageBox> Message Box containing this message </messageBox>
        </messageElement>
        <messageElement> ... </messageElement>
        <messageElement> ... </messageElement>
    </messageList>
</soap-body>
```

Process the SOAP response and iterate through the message history if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement msgList = SOAPUtilities.getElement(responseBody, "messageList",␣
↪nsURI, 0);

    Iterator msgIterator = msgList.getChildElements();
    while(msgIterator.hasNext()){

        List elementList = new ArrayList();

        SOAPElement messageElement = (SOAPElement)msgIterator.next();
```

```
        Iterator elements = messageElement.getChildElements();

        // MessageId
        SOAPElement msgId = (SOAPElement)(elements.next());

        // MessageBox
        SOAPElement msgBox = (SOAPElement)(elements.next());

        System.out.println("Message ID: " + (String)msgId.get(0) + "\t" +
↪"Message Box: " + msgBox.get(0));
    }
}
```

The method `getElement` gets the element with the name `messageList` and namespace URI :code‘nsURI‘. Then, a list of `messageElement` objects is extracted from `messageList`. Each `messageElement` object contains the values of `messageId` and `messageBox`.

Writing AS2 Messaging Client

Send AS2 Message

We need to create a SOAP message with the following from 3 parameters and send them to Hermes as a web service request.

- `as2_from`: AS2 sender
- `as2_to`: AS2 receiver
- `type`: Payload content type

1. Define a namespace URI and a prefix conforming to the WSDL.

```
private String nsURI = "http://service.as2.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL senderWSURL = "http://localhost:8080/corvus/httpd/as2/sender";
```

2. Create a SOAP message factor and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<as2_from> as2from </as2_from>
<as2_to> as2to <as2_to>
<type> type </type>
```

Compose a sample SOAP request to send an AS2 message:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("as2_from", nsPrefix, nsURI, this.
↪as2From));
soapBody.addChildElement(createElement("as2_to" , nsPrefix, nsURI, this.as2To));
soapBody.addChildElement(createElement("type" , nsPrefix, nsURI, this.type));
```

The method `createElement` creates a SOAP element with the namespace prefix `nsPrefix`, the namespace URL `nsURI`, and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Attach a payload if necessary. The following example uses a purchase order XML as the payload of the AS2 message, so the associated content type is `application/xml`.

   Note: Only *one* payload is allowed in the SOAP request for an AS2 message.

```
AttachmentPart attachmentPart = request.createAttachmentPart();
FileDataSource fileDS = new FileDataSource(new File("purchase_order.xml"));
attachmentPart.setDataHandler(new DataHandler(fileDS));
attachmentPart.setContentType("application/xml");
request.addAttachmentPart(attachmentPart);
```

5. Save the changes to the SOAP message.

```
request.saveChange();
```

6. Send the SOAP request to Hermes and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, senderWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

7. Process the SOAP response and extract the identifier of the AS2 message, and print it to `System.out` if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement messageIdElement = getFirstChild(responseBody, "message_id",␣
↪nsURI);
    System.out.println(messageIdElement == null ? null : messageIdElement.
↪getValue());
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

   The method `getFirstChild` gets the first element with the name `message_id` and the namespace URI `nsURI`.

   The sender Hermes translates The SOAP request into an AS2 message stored in the file system, and then delivers the message to the receiver Hermes specified in the SOAP request parameters, of which `AS2From` and `AS2To` identify the partnership between the sender and the receiver.

List AS2 Messages

We need to create a SOAP message with the following 3 parameters.

- `as2From`: AS2 sender
- `as2To`: AS2 receiver
- `numOfMessages`: Number of messsages

1. Define a namespace URI and a prefix conforming to the WSDL.

```
private String nsURI = "http://service.as2.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL receiverListWSURL = "http://localhost:8080/corvus/httpd/as2/receiver_
↪list";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<as2_from> as2from </as2_from>
<as2_to> as2to <as2_to>
<numOfMessages> 100 </numOfMessages>
```

Compose a sample SOAP request to list AS2 messages:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("as2From" , nsPrefix, nsURI, this.
↪as2From));
soapBody.addChildElement(createElement("as2To" , nsPrefix, nsURI, this.as2To));
soapBody.addChildElement(createElement("numOfMessages", nsPrefix, nsURI, this.
↪numOfMessages + ""));
```

The method `createElement` creates a SOAP element with the namespace prefix `nsPrefix`, the namespace URL `nsURI` and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, senderWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

Here is the SOAP response:

```
<soap-body>
    <messageIds>
        <messageId> ... </messageId>
        <messageId> ... </messageId>
        <messageId> ... </messageId>
        <messageId> ... </messageId>
    </messageIds>
</soap-body>
```

Process the SOAP response and extract the identifiers of the AS2 messages to `System.out` if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement messageIdsElement = getFirstChild(responseBody, "messageIds",␣
↪nsURI);
    Iterator messageIdElementIter = getChildren(messageIdsElement, "messageId",␣
↪nsURI);

    while(messageIdElementIter.hasNext()) {
        SOAPElement messageIdElement = (SOAPElement)messageIdElementIter.next();
        System.out.println(messageIdElement.getValue());
    }
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name `messageIds` and the namespace URI `nsURI`. All children with the name `messageId` and the namespace URI `nsURI` are then extracted.

Retrieve AS2 Message

We need to create a SOAP message with the identifier of the target message and send it to Hermes as the web service request.

1. Define a namespace URI and a prefix conforming to the WSDL.

```
private String nsURI = "http://service.as2.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL receiverWSURL = "http://localhost:8080/corvus/httpd/as2/receiver";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<messageId> messageId </messageId>
```

Compose a sample SOAP request to retrieve an AS2 message:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("messageId", nsPrefix, nsURI, messageId));
```

The method `createElement` creates a SOAP element with the namespace `nsPrefix`, the namespace URL `nsURI` and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, receiverWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

6. Here is the SOAP response:

```
<soap-body>
    <hasMessage> ... </hasMessage>
</soap-body>
```

The attachment is formatted as a MIME part.

Process the SOAP response and extract the payload from the AS2 message if available.

```
if (!responseBody.hasFault()){
    SOAPElement hasMessageElement = getFirstChild(responseBody, "hasMessage",␣
↪nsURI);
    ArrayList payloadsList = new ArrayList();
    if (hasMessageElement != null){
        Iterator attachmentPartIter = response.getAttachments();
        while(attachmentPartIter.hasNext()) {
            AttachmentPart attachmentPart = (AttachmentPart) attachmentPartIter.
↪next();
            InputStream ins = attachmentPart.getDataHandler().getInputStream();
            // Do something I/O to extract the payload to physical file.
        }
    }
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name `hasMessage` and namespace URI `nsURI`. The boolean value of `hasMessage` represents the existence of a payload in this message.

The payload is extracted from the attachment part and sent to the input stream. The data can be piped to a processor or saved into a file.

Get AS2 Message Status

We need to create a SOAP message with the identifier of the target message and send it to Hermes as a web service request.

1. Define a namespace URI and a prefix conforming to the WSDL.

```
private String nsURI = "http://service.as2.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL statusQueryWSURL = "http://localhost:8080/corvus/httpd/as2/status";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<messageId> messageId </messageId>
```

Compose a sample SOAP request to get the status of an AS2 message:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("messageId", nsPrefix, nsURI, messageId));
```

The method `createElement` creates a SOAP element with the namespace prefix :code‘nsPrefix‘, the namespace URL `nsURI` and the string value of the element.

The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to get the status of an AS2 message and get the following SOAP response.

```
SOAPMessage response = soapConn.call(request, statusQueryWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

6. Here is the SOAP response:

```
<soap-body>
    <MessageInfo>
        <status> The current status of message </status>
        <statusDescription> The current status description of message </
↪statusDescription>
        <mdnMessageId> The message id of acknowledgment / receipt if any </
↪mdnMessageId>
        <mdnStatus> The status of acknowledgment / receipt if any </mdnStatus>
        <mdnStatusDescription> The status description of acknowledgment / receipt␣
↪if any </mdnStatusDescription>
    </MessageInfo>
</soap-body>
```

Process the SOAP response and extract the status of the AS2 message if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement messageInfoElement = getFirstChild(responseBody, "MessageInfo",␣
↪nsURI);
    System.out.println("Message Status : " + getFirstChild(messageInfoElement,
↪"status", nsURI);
    System.out.println("Message Status Desc : " +␣
↪getFirstChild(messageInfoElement, "statusDescription", nsURI);
    System.out.println("Ack Message Identifiers : " +␣
↪getFirstChild(messageInfoElement, "mdnMessageId", nsURI);
    System.out.println("Ack Status : " + getFirstChild(messageInfoElement,
↪"mdnStatus", nsURI);
    System.out.println("Ack Status Desc : " + getFirstChild(messageInfoElement,
↪"mdnStatusDescription", nsURI);
} else {
    throw new SOAPException(responseBody.getFault().getFaultString());
}
```

The method `getFirstChild` gets the first element with the name `MessageInfo` and the namespace URI `nsURI`.

Get AS2 Message History

We need to create a SOAP message with the following 5 parameters and send it to Hermes as the web service request.

- `messageId`: Message ID
- `messageBox`: Message box
- `as2From`: AS2 sender
- `as2To`: AS2 receiver
- `status`: Status

1. Define the namespace URI and a prefix conforming to the WSDL.

```
private String nsURI = "http://service.as2.edi.cecid.hku.hk/";
private String nsPrefix = "tns";
private URL msgHistoryWSURL = "http://localhost:8080/corvus/httpd/as2/msg_history
↪";
```

2. Create a SOAP message factory and a SOAP message object.

```
SOAPMessage request = MessageFactory.newInstance().createMessage();
```

3. Populate the SOAP body by filling in the required parameters.

```
<messageId> messageId </messageId>
<messageBox> messageBox </messageBox>
<as2From> as2From </as2From>
<as2To> as2To </as2To>
<status> status </status>
```

   Compose a sample SOAP request to get an AS2 message history:

```
SOAPBody soapBody = request.getSOAPBody();
soapBody.addChildElement(createElement("messageId", nsPrefix, nsURI, messageId));
soapBody.addChildElement(createElement("messageBox", nsPrefix, nsURI,␣
↪messageBox));
soapBody.addChildElement(createElement("as2From", nsPrefix, nsURI, cpaId));
soapBody.addChildElement(createElement("as2To", nsPrefix, nsURI, service));
soapBody.addChildElement(createElement("status", nsPrefix, nsURI, status));
```

   The method `createElement` creates a SOAP element with the namespace prefix `nsPrefix`, the namespace URL `nsURI`, and the string value of the element.

   The implementation of `createElement` is shown below:

```
SOAPElement soapElement = SOAPFactory.newInstance().createElement(localName,␣
↪nsPrefix, nsURI);
soapElement.addTextNode(value);
return soapElement;
```

4. Save the changes to the SOAP message.

```
request.saveChange();
```

5. Send the SOAP request to Hermes to get an AS2 message history and get a SOAP response.

```
SOAPMessage response = soapConn.call(request, receiverListWSURL);
SOAPBody responseBody = response.getSOAPBody();
```

Here is the SOAP response.

```
<soap-body>
    <messageList>
        <messageElement>
            <messageId> Message ID of this message </messageId>
            <messageBox> Message Box containing this message </messageBox>
        </messageElement>
        <messageElement>
            <messageId> Message ID of this message </messageId>
            <messageBox> Message Box containing this message </messageBox>
        </messageElement>
        <messageElement> ... </messageElement>
        <messageElement> ... </messageElement>
    </messageList>
</soap-body>
```

Process the SOAP response and iterate through the AS2 message history if there is no SOAP fault.

```
if (!responseBody.hasFault()){
    SOAPElement msgList = SOAPUtilities.getElement(responseBody, "messageList",␣
↪nsURI, 0);

    Iterator msgIterator = msgList.getChildElements();
    while(msgIterator.hasNext()){

        List elementList = new ArrayList();

        SOAPElement messageElement = (SOAPElement)msgIterator.next();

        Iterator elements = messageElement.getChildElements();

        // MessageId
        SOAPElement msgId = (SOAPElement)(elements.next());

        // MessageBox
        SOAPElement msgBox = (SOAPElement)(elements.next());

        System.out.println("Message ID: " + (String)msgId.get(0) + "\t" +
↪"Message Box: " + msgBox.get(0));
    }
}
```

The method `getElement` gets the element with the name `messageList` and the namespace URI `nsURI`. The `messageElement` objects are extracted from the `messageList` object. Each `messageElement` object contains the values of `messageId` and `messageBox`.

References

Reference Documentation

- *The First Step*
- *Installing Hermes*
- *Using Hermes API*
- *Setting Up ebMS 2.0 Partnerships*

- *Setting Up AS2 Partnerships*
- OASIS ebMS 2.0 Specification
- AS2 Specification

Reference Source Code

- `Hermes loopback test`

Index

E

J