

cisdal
simply integration



cisdal message gateway

Installation and User Guide

Version 1.0.5
March 2009



Table of Contents

Introduction.....3
 Purpose of this Document.....3
 Installation.....5
 Deployment Architectures.....5
 Single Tomcat Instance.....5
 Single Tomcat Instance with Apache Proxy (Recommended).....6
 Multiple Instances with Apache Proxy.....7
 Installation Process.....8
 Installing Hermes 2.0.....8
 Patch Hermes 2.0.....9
 Installing Cisdal Message Gateway.....9
 Performing Database Changes.....9
 Deploying CMD to Tomcat.....10
 Database Configuration.....10
 Application Configuration.....10
 Configuring Partnerships (ebMS only).....12
 Configuring Your Security Key.....14
 Cisdal Message Gateway API.....15
 Introduction.....15
 Web Services.....15
 submitSyncMessageByValue.....15
 submitAsyncMessageByValue.....18
 findMessageByMessageId.....19
 findMessageListByConversationId.....22
 findSyncMessagesByOriginalMessageId.....23
 findSyncMessagesByInternalTransactionId.....24
 findMessageListByMessageSearch.....24
 Receiving Messages Asynchronously.....25
 Monitoring.....26
 Housekeeping.....27
 Stopping the Cisdal Message Gateway.....27
 Cisdal Message Gateway Test Harness.....28
 FAQ.....29

Introduction

The Cisdal Message Gateway is an extension to the Hermes 2.0 Message Service Handler (MSH)¹. The Hermes 2.0 MSH is an open project sponsored by the Centre for for E-Commerce Infrastructure Development providing ebMS 2.0² and AS2 messaging. It is designed to allow secure and reliable message exchange between trading partners over the Internet.

Cisdal provide full support for the Hermes 2.0 Message Gateway, along with a set of extensions designed to simplify integrating your existing internal systems with the Message Gateway.

The Cisdal Message Gateway is ideally suited to any organisation intending to use ebXML or AS2 messaging as a result of its adoption by a Trading Partner. It allows you to abstract away the complications of ebXML, and instead focus solely on the business information being sent and received. The Cisdal Message Gateway integrates you with your Trading Partners so seamlessly that it is like your Trading Partners systems become part of your enterprise systems.

The Cisdal Message Gateway offers the following features:

- A simple Web Services based interface for sending synchronous or asynchronous messages to Trading Partners.

- The ability to treat asynchronous responses to your messages from Trading Partners as synchronous responses by your internal applications.

- The ability to receive messages through a variety of interfaces (such as JMS, EJBs and Web Services) with guaranteed delivery.

- The ability to treat message payloads as simple strings of data rather than files. This is ideally suited to applications exchanging XML documents.

- Database backed logging of all messages sent and received (including their payloads), along with automatic correlation of asynchronous responses to their original requests.

- The ability to upload CPA (Collaboration Protocol Agreements) documents to automatically configure partnership definitions.

- A Web Service interface for querying messages based on a range of criteria.

- An alerting framework notifying you messages have failed and require manual intervention.

The Cisdal Message Gateway and the Hermes 2.0 Message Service Handler are both Java Web Applications that can be deployed to any JEE Servlet container supporting Java 5 or higher. Both applications support Oracle, MySQL and Postresql databases for data persistence.

Purpose of this Document

The primary purpose of this document is to explain the installation and use of the Cisdal Message Gateway.

This document provides a step by step guide for installing and testing the Cisdal Message Gateway, which includes installation of the Hermes 2.0 Gateway.

This document also explains the public APIs exposed by the Cisdal Message Gateway, and their usage. This includes Web Services exposed by the Gateway, and any extension points provided by

1 Find more information at www.freebxml.org/msh.htm

2 www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf

the gateway.

Installation

Deployment Architectures

Before installation begins, a deployment architecture must be selected that meets the needs of your business. Possible architectures range from a single host for all components, through to a host dedicated to each component. Each approach has its own advantages and disadvantages, and these will be discussed further below.

The high level components that constitute the Cisdal Message Gateway are as follows:

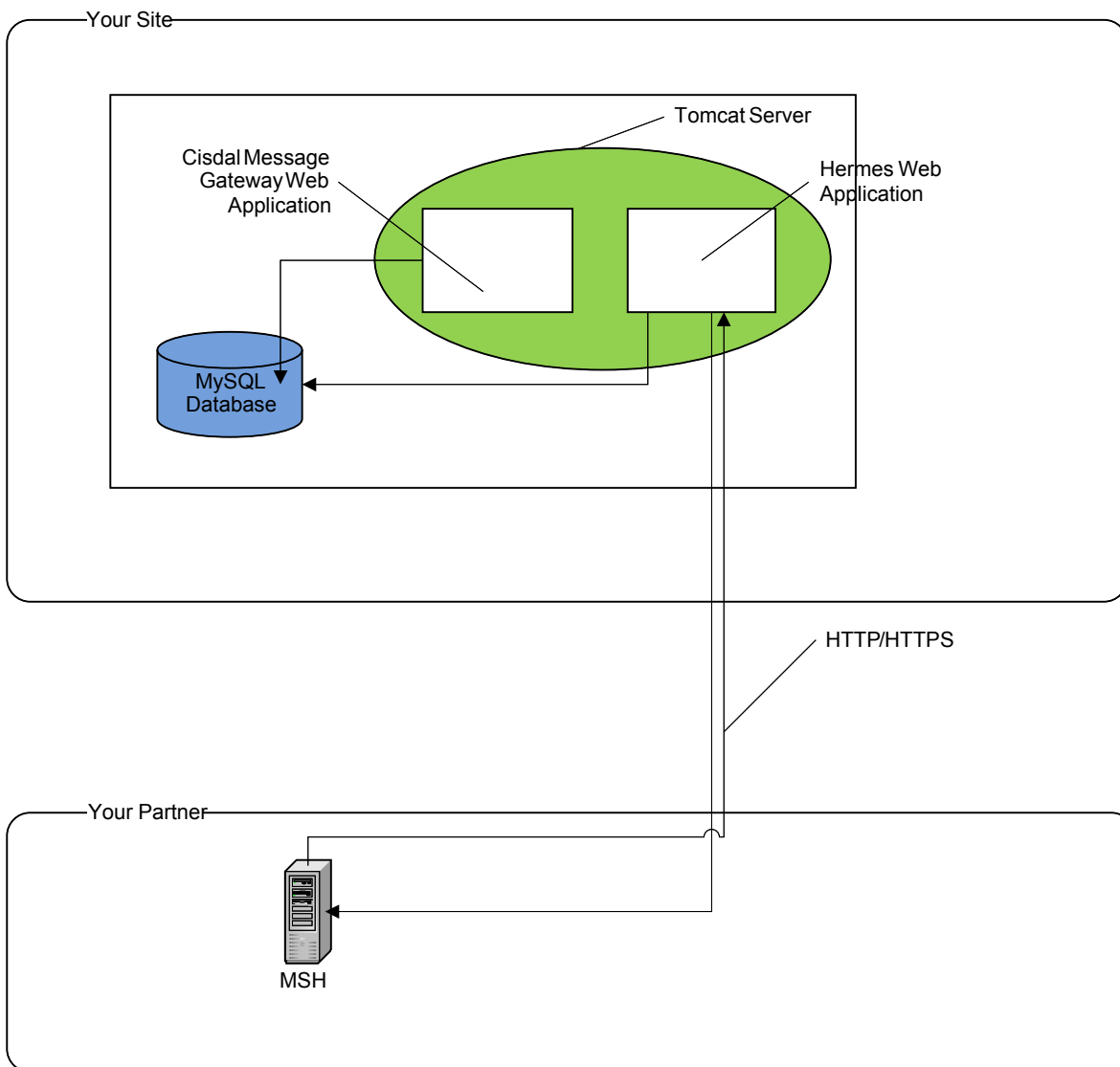
1. A single database instance. This may be MySQL, Oracle or Postresql.
2. A Java Web Application containing the Hermes Message Gateway
3. A Java Web Application containing the Cisdal Message Gateway
4. Optionally, a proxy server through which remote clients access the gateway

There are a variety of architectures that may be used for deploying Hermes 2.0 alongside the Cisdal Message Gateway. This section summarises the recommended deployment architectures:

Single Tomcat Instance

The single instance architecture consists of a single Tomcat instance communicating with your trading partners Message Service Handler directly. The database may be collocated on the same server as the Tomcat instance, or on a second server.

This deployment architecture provides a simple solution, and is a viable solution for many sites. The diagram below shows an example of this architecture. The green oval represents the Tomcat instance, which has two web applications deployed to it.



In this solution it is likely that Tomcat will need to be responsible for SSL Termination (since SSL is the primary mechanism through which most partners implement message encryption).

The disadvantage of this approach is that the Tomcat server is exposed directly to the Internet. Many businesses have policies regarding the applications that can be exposed directly to the Internet, and often this precludes Application Servers from being directly exposed.

Single Tomcat Instance with Apache Proxy (Recommended)

This architecture builds on the Single Instance architecture, but places an Apache Web Server instance in front of the Tomcat instance. This is the most popular architecture for deploying the Cisdal Message Gateway, and uses the Apache instance to provide SSL termination capabilities. The Apache instance is deployed with the mod_proxy plugin, which allows requests to be forwarded to and from remote partners.

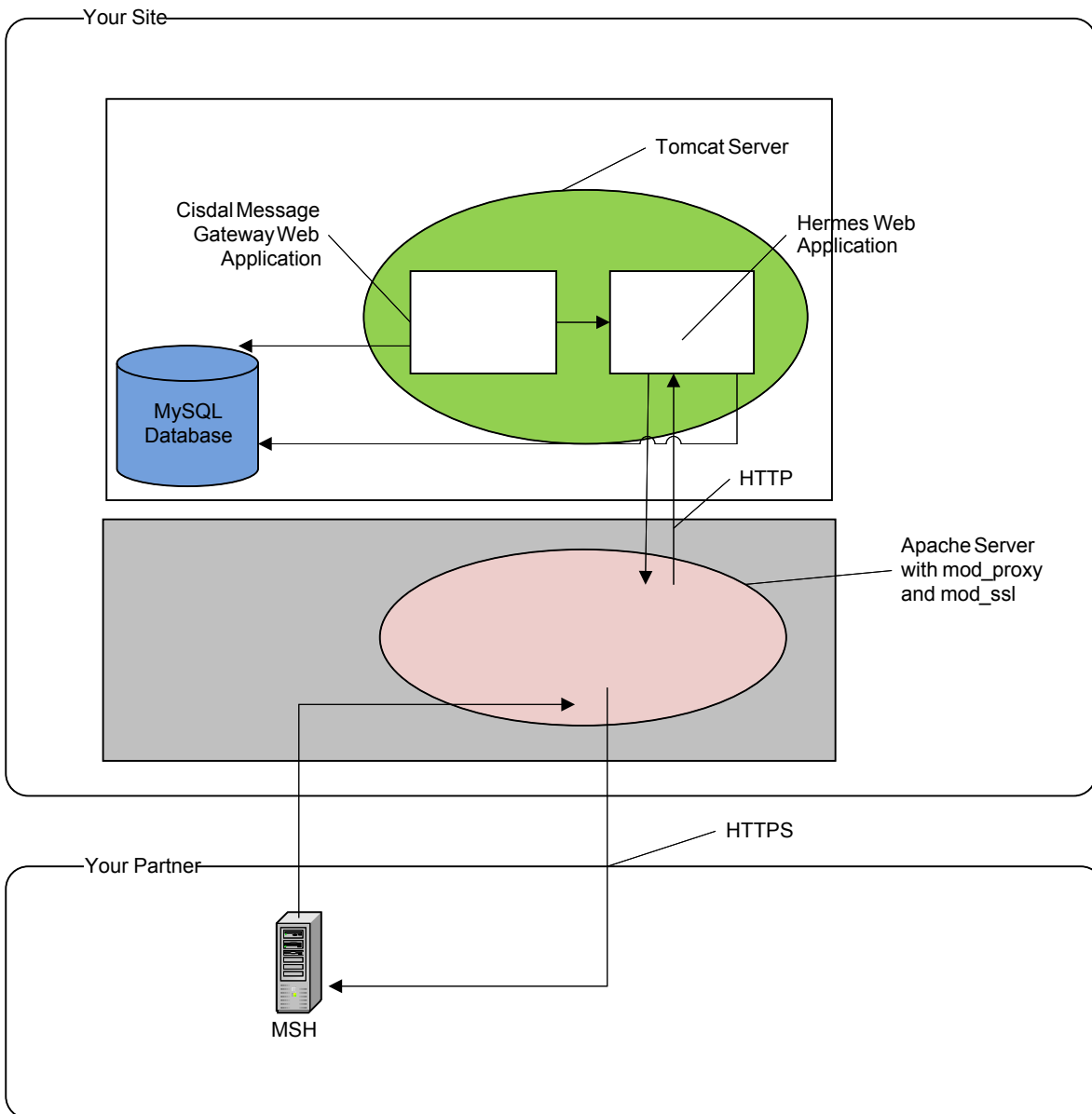
Apache can be used as a reverse proxy when receiving messages from a remote partner, and can be used as a traditional forward proxy when sending messages to partners.

This architecture ensures that the Tomcat instance is not exposed directly to the Internet, and therefore provides an extra level of flexibility, and is arguably more secure.

With this deployment architecture the database may be installed on the same server as the Tomcat instance, or on a remote server.

This architecture may also be used with alternative Application Servers such as Glassfish or Websphere. This is often a desired option for businesses with existing investment in Application Server products.

The diagram below shows an example of this architecture:



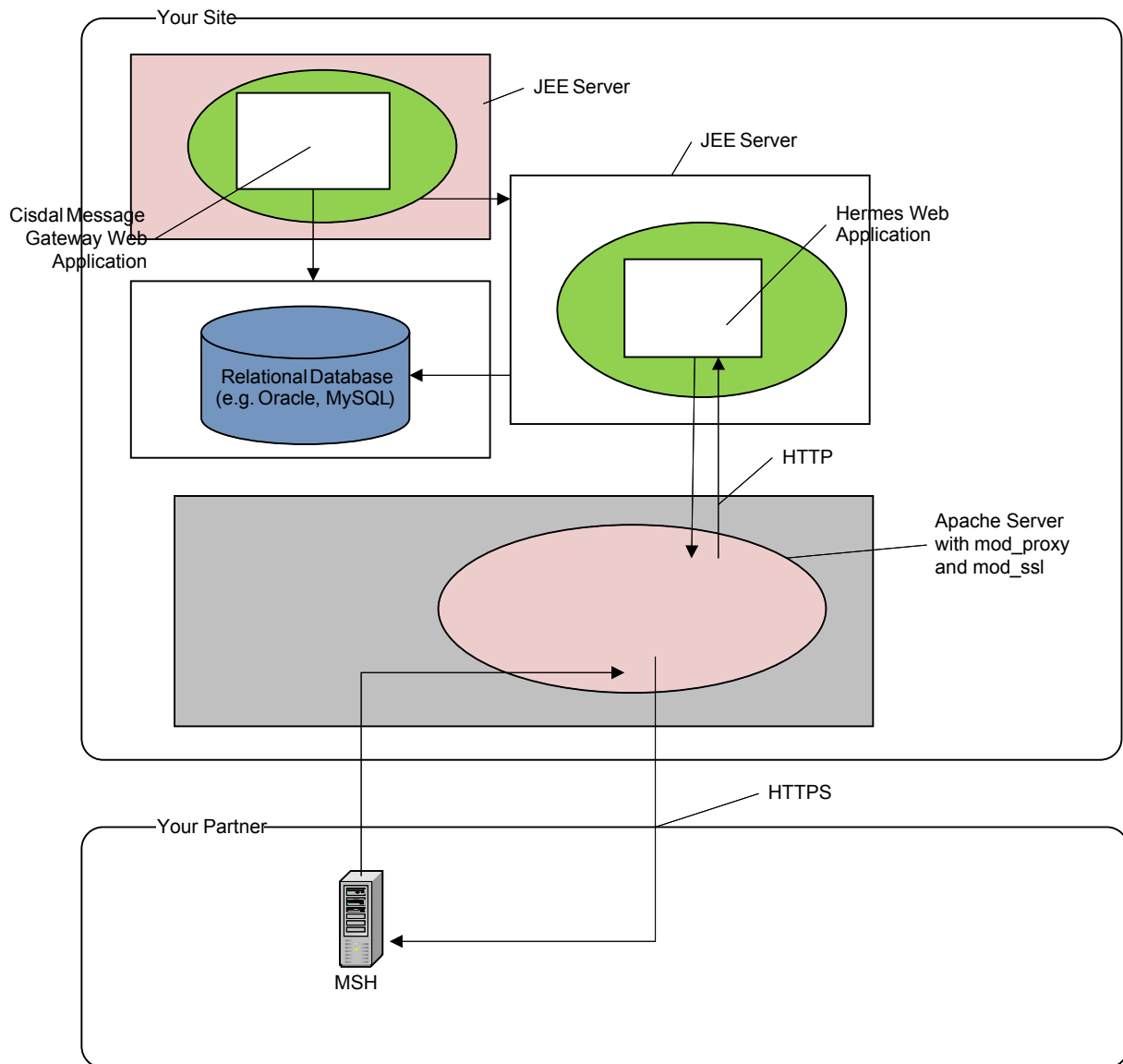
Multiple Instances with Apache Proxy

The multiple instance architecture builds on the Single Instance with Apache Proxy architecture, but splits the Hermes Web Application and the Cisdal Message Gateway Web Application across two separate Application Servers.

This allows an extra level of scalability from the previous architectures, but this is typically only

required in very high utilisation scenarios.

The diagram below provides an example of this architecture:



As a general rule, Cisdal recommend the Single Tomcat Instance with Apache Proxy approach. This provides a level of simplicity at the application layer, while still allowing client connections to be handled by the Apache Web Server.

Installation Process

The Cisdal Message Gateway installation pack provides verified versions of most of the software that will be installed. The installation pack also provides all the documentation that will be needed to support the installation process.

Installing Hermes 2.0

Before installing the Cisdal Message Gateway, the Hermes 2.0 Message Gateway should be

installed. The install process is described in the Hermes Installation Guide (hermes2_install.pdf) provided with the Cisdal Message Gateway (directly in the Hermes folder). Cisdal provide a README document alongside the Hermes Installation Guide that summaries the key steps that must be performed, and also provides guidance on some steps that may not be clear in the design.

The Cisdal Message Gateway also contains a build of Hermes that we have verified as stable.

We recommend the following configuration for Hermes:

- Tomcat version 5.5
- Sun Java 5
- MySQL 5 or Oracle 10g
- Apache 2.2 (if required)

Although Hermes supports a much wider variety of Application Servers and Databases, these are the products that we have tested, and proven to be solid in production environments.

Patch Hermes 2.0

Cisdal will provide a set of patches to Hermes. All patches can be found in the HermesPatch folder of the installation pack. This directory contains a README file explaining the patches that should be performed.

The patches provided currently relate specifically to the 2008/08/27 release of Hermes. If you have chosen to adopt a newer or older version of Hermes, please contact us for the relevant patches.

Installing Cisdal Message Gateway

The Cisdal Message Gateway consists of a single WAR (Web Archive) file that can be deployed to any JEE Servlet container supporting Java 5 or higher. Cisdal recommend co-locating the Cisdal Message Gateway Web Application in the same container as the Hermes Message Gateway Web Application.

This section contains specific information for deploying to Tomcat, but the same process will apply with most other Application Servers. Support is available from Cisdal where problems occur.

Performing Database Changes

The Cisdal Message Gateway must use the same database instance as Hermes 2.0 for persisting and retrieving necessary information. There is a database script that must be run in order to create the necessary changes in the ebms or as2 databases.

The script that should be run is located in the CMG folder of the installation pack, and is called create_objects.sql. By default, the installation pack contains a MySQL specific version, but alternative versions are available on request.

This can be executed as follows:

```
mysql ebms -u corvus -p <password> < create_objects.sql
```

The following output should be displayed:

Deploying CMD to Tomcat

Ensure that the Tomcat server is available. Access the Tomcat Manager page at the location:

<http://localhost:8080/manager/html>

This is restricted to Tomcat administrators. A Tomcat administrator by default is a user listed in the `conf/tomcat-users.xml` file as having a role of "manager". This file can be used for adding users, changing their passwords, and changing their roles.

Once the manager page is loaded, select a file in the "WAR file to deploy" section, and choose the `CisdalMessageGateway.war` file in the `CMD` folder of the installation pack. Once chosen, select "Deploy", and the application will then be listed as a deployed Application under the `/CisdalMessageGateway` path.

Once the Cisdal Message Gateway is deployed it must be configured. This will be done by editing three specific files. Two of these files reside directly in the Web Application that has been deployed, and can be found under the `<Tomcat Base>/webapps/CisdalMessageGateway` folder (if using Tomcat).

Database Configuration

Cisdal Message Gateway uses JPA for database persistence. In order to ensure that the Cisdal Message Gateway knows the location of the database, the `persistence.xml` file in `<Tomcat Base>/webapps/CisdalMessageGateway/WEB-INF/classes/META-INF` of the deployed Web Application need to be modified. The default version uses a MySQL implementation, but alternative versions are available on request.

The following properties should be updated to the correct values:

```
<property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/ebms"/>
<property name="toplink.jdbc.user" value="corvus"/>
<property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
<property name="toplink.jdbc.password" value="corvus"/>
```

Typically only the password will need to be modified.

Once modified, ensure a backup of this file is made outside Tomcat in case the application needs to be reinstalled at a future date.

Application Configuration

The Cisdal Message Gateway uses the Spring framework for configuration. This ensures that a wide variety of options can be configured without rebuilding the base application.

The configuration file used when loading the Cisdal Message Gateway Web Application is found by reading the `web.xml` file contained in `webapps/CisdalMessageGateway/WEB-INF/`.

The following portion of this file provides the reference to the configuration file:

```
<servlet>
  <servlet-name>BootstrapServlet</servlet-name>
```

```

<servlet-class>
    com.cisdal.hermesintegrator.servlets.BootstrapServlet
</servlet-class>
<init-param>
    <param-name>configFile</param-name>
    <param-value>
        C:/apps/CisdalMessageGateway/Config/MessageReceiverContext.xml
    </param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>

```

The link (c:/apps/CisdalMessageGateway/Config/MessageReceiverContext.xml in this case) must point to the MessageReceiverContext.xml file (found in the base CMG folder within the installation pack). This may be located anywhere on the local file-system.

After deploying the Cisdal Message Gateway to Hermes, modify this parameter to point to the appropriate location, and restart Tomcat.

The MessageReceiverContext.xml file contains a number of parameters. Many of these can be left as their default settings, but a number must be changed.

Sending Information

The following section is used for configuring options for sending files:

```

<bean id="sender" class="com.cisdal.messagesender.impl.EBMSsender">
    <property name="hermesLocation"
value="http://127.0.0.1:8080/corvus/httpd/ebms/" />
    <property name="outboxFileStore" value="/hermes2/outbox/" />
</bean>

```

The URL should point to the Hermes2 Web Application. If Hermes2 is installed on the same machine, and is running on the default Tomcat port, this setting will not need to change.

The outboxFileStore property needs to point to a directory where you wish to save a text based version of messages and payloads that will be sent to partners.

Receiving Information

The following section is used for configuring options for receiving files:

```

<bean id="receiver" class="com.cisdal.messagereceiver.impl.EbmsReceiver">
    <property name="hermesLocation"
value="http://127.0.0.1:8080/corvus/httpd/ebms/" />
    <property name="inboxFileStore" value="/hermes2/inbox/" />
    <property name="asyncHandler" ref="asyncHandler" />
    <property name="messageCorrelator" ref="messageCorrelator" />
    <property name="cisdalLogDirectory" value="/hermes2/logs" />
    <property name="cpaId" value="cisdal_acme" />
</bean>

```

The key items that should be changed here are:

The URL should be changed to point to the Hermes2 instance – as described above.

The `inboxFileStore` property needs to point to a directory where you wish to save a text based version of messages that are received from partners.

The `cisdalLogDirectory` property needs to point to a directory where you wish a log of received messages to be written.

The `cpaId` property can be populated if you wish to receive messages from a particular CPA ID (a CPA ID typically relates to a single partnership). This should be left as an empty string if you wish to receive messages for any CPA ID. Limiting a Cisdal Message Gateway instance to a single CPA ID is useful in cases where you wish to use multiple Cisdal Message Gateway instances for the various partners you deal with.

System Information

The following section sets other system wide information:

```
<bean id="hub" class="com.cisdal.messagereceiver.impl.DatabaseBackedMessagingHub">
  <property name="alertEngine" ref="alertEngine"/>
  <property name="retryHandler" ref="retryEngine"/>
  <property name="receiver" ref="receiver"/>
  <property name="sender" ref="sender"/>
  <property name="maxSyncTimeOut" value="60000"/>
  <property name="cleanupMilliseconds" value="30000"/>
  <property name="pollingMilliseconds" value="1000"/>
  <property name="retryIntervalSeconds" value="20"/>
  <property name="alertIntervalSeconds" value="20"/>
</bean>
```

The following properties are often altered here:

`maxSyncTimeOut`: This is the maximum amount of time that a client can wait for a synchronous response in milliseconds.

`pollingMilliseconds`: This is how often the Cisdal Message Gateway will check for new messages. This is typically set to a value of between 100 and 1000.

`retryIntervalSeconds`: This is how long to wait before retrying to deliver messages to internal applications where the previous attempts failed.

`alertIntervalSeconds`: This is how long to wait between looking to see if any new alerts have been raised.

After changing any parameters the Application Server should be restarted.

Configuring Partnerships (ebMS only)

This document assumes you have established agreed relationships with your Trading Partners and have agreed CPA (Collaboration Protocol Agreement) documents to represent these relationships. Cisdal can provide support when defining CPA agreements with Trading Partners if required.

If you are configuring relationships manually, the Hermes User Guide can be used as the basis for creating partnerships.

Partnerships are always created directly in Hermes, the Cisdal Message Gateway accesses partnership information directly from Hermes.

The patches Cisdal apply to Hermes Message Gateway during installation provide access to a CPA upload function. Access this by selecting

The CPA upload function will not import your trading partners security certificates as used for verifying digital signatures. These must be uploaded manually by selecting the partnership, and performing the following steps:

1. Navigate to http://localhost:8080/corvus/admin/ebms/agreement_upload (using the correct host and port).
2. Use your Trading Partners name (as found in `<tp:PartyInfo tp:partyName="Chorus2" />` within the CPA document) as the Party Name.
3. Select the CPA file to load
4. Select to Import the CPA File



[Administration Console]

The Universal Messaging Gateway

Module	Message H...	Partnership	Upload Ag...
Corvus Main System	Collaboration-Protocol Profile and Agreement Upload Party Name: <input type="text" value="Acme"/> File Upload: <input type="text" value="reprod\Cisdal_Acme.xml"/> <input type="button" value="Browse..."/> <input type="button" value="Import"/>		
Ebms Plugin			
AS2 Plugin			

After importing, the Partnership will be available in the "Partnership" tab. If the CPA contained multiple services and actions, each will be separated into individual partnerships, but will have the same partnership ID.

Once the partnership has been installed you will need to import your partners x.509 certificates into the partnership where you are using Digital Signatures or XML Encryption. This should be performed as follows:

1. Ask your trading partner for a copy of their x.509 certificate (if this has not been done already)
2. Navigate to the partnership tab inside Hermes (<http://localhost:8080/corvus/admin/ebms/partnership>)
3. Select the partnership you wish to associate the certificate with (there may be multiple partnerships for the same Trading Partner, therefore this may need to be done multiple times).
4. Where XML Encryption is used, select the Browse button beside "Certificate For Encryption" and select the certificate your trading partner has provided for XML Encryption (note, most configurations do not use XML Encryption, instead traffic is

secured through the use of SSL).

5. Where Digital Signatures are required (i.e. "Signing Required" is set to "Yes" in the CPA), select the Browse button beside "Certificate For Verification" and select the certificate your trading partner has provided for Digital Signatures.
6. Select to "Update" the partnership.

Configuring Your Security Key

Where you are using XML Encryption or Digital Signatures, you must configure Hermes to use your correct private key for:

1. Creating digital signatures
2. Unencrypting encrypted messages

This can be performed by modifying the ebms.module.xml file in

<Hermes_Base>\plugins\hk.hku.cecid.ebms\conf\hk\hku\cecid\ebms\spa\conf for ebMS use, or the as2.module.core.xml file in

<Hermes_Base>\plugins\hk.hku.cecid.as2\conf\hk\hku\cecid\ebms\spa\conf for AS2 use.

Where possible security keys should be in the PKCS12 format. Cisdal can provide support for creating and configuring security keys where required.

For AS2 messaging, the keystore-manager section should be updated to reflect your security key.

For ebMS messaging, the `keystore-manager-for-signature` section should be updated to the key that will be used for signing messages. The `keystore-manager-for-decryption` section should be updated to the key that will be used for unencrypting messages.

This completes the configuration that is required for your ebXML Gateway. You may now use the Cisdal Message Gateway Test Harness (described below) to test your implementation.

Cisdal Message Gateway API

Introduction

This section explains the APIs associated with the Cisdal Message Gateway. The Cisdal Message Gateway does not provide a front end application for interacting with messages, instead it provides the plumbing for your front end applications. The API exposed by the Cisdal Message Gateway consists primarily of Web Services, but additional APIs are provided for situations where Web Services are not the natural choice – such as receiving messages asynchronously.

Web Services

The primary interface through which the Cisdal Message Gateway can be accessed is via Web Services.

The following Web Services are defined for accessing the application:

- submitSyncMessageByValue
- submitAsyncMessageByValue
- findMessageByMessageId
- findMessageListByConversationId
- findMessageListByMessageSearch
- findSyncMessagesByOriginatingMessageId

Each of these services will be described in detail below:

submitSyncMessageByValue

ebXML message flows are naturally asynchronous. Despite this, many business flows implemented through ebXML gateways are more naturally modelled as synchronous flows. An example of this is placing a sales or purchase order with a trading partner. Typically a business level response is expected that confirms the order, and provides an order number, and other relevant confirmation information.

The submitSyncMessageByValue Web Service allows you to send a message to a trading partner, and will block until the asynchronous response is received, and delivered back to you as the synchronous response. This alleviates your internal applications from the responsibility of request/response correlation.

The submitSyncMessageByValue Web Service can be found at:

<http://localhost:8080/CisdalMessageGateway/submitSyncMessageByValueService?wsdl>

The request information accepted by this web service is (fields should be considered mandatory unless indicated otherwise):

Name	Type	Description
conversationId	xsd:string	Optionally: The conversation id to be linked to

		this message, and any subsequent messages related to this business level conversation.
messageId	xsd:string	Optional: the ebXML message Id for the message to which this is a direct response.
message	xsd:string	The message payload to be sent.
messageType	xsd:string	Optional: The type of the data. This defaults to text/xml.
fromPartyId	xsd:string	Your party id as defined in the CPA agreement with your trading partner.
fromPartyType	xsd:string	Your party type as defined in the CPA agreement with your trading partner. This will typically be urn:duns.
fromPartyRole	xsd:string	Optional: Your party role as defined in the CPA agreement with your trading partner.
toPartyId	xsd:string	The party id of the trading partner you wish to send the message to, as defined in the CPA agreement with the trading partner.
toPartyType	xsd:string	The party type of the trading partner you wish to send the message to, as defined in the CPA agreement with the trading partner.
toPartyRole	xsd:string	Optional: The party role of the trading partner you wish to send the message to, as defined in the CPA agreement with the trading partner.
cpaId	xsd:string	The CPA ID for the trading partner relationship as defined in the CPA agreement with the trading partner.
action	xsd:string	The action for the type of message you wish to send. These are defined in CPA documents and allow multiple types of document to be sent as part of the same trading partnership.
service	xsd:string	The service for the type of message you wish to send. These are defined in CPA documents and allow multiple types of document to be sent as part of the same trading partnership.
timeout	xsd:long	The number of milliseconds to wait for a response
internalTransactionId	xsd:string	Optional: an internal transaction id that you wish to identify with this message

The following information will be received in response:

Name	Type	Description
------	------	-------------

conversationId	xsd:string	Optionally: The conversation id to be linked to this message, and any subsequent messages related to this business level conversation.
sendingMessageId	xsd:string	The message id for the message sent to the trading partner.
receivingMessageId	xsd:string	Optionally: The message id for the message received in response. If a timeout occurred before a response was received, this will be empty.
message	xsd:string	Optionally: the message payload received in response. If a timeout occurred before a response was received, this will be empty.
action	xsd:string	Optionally: The message action for the message received in response. If a timeout occurred before a response was received, this will be empty. This may differ from the action for the message sent.
service	xsd:string	Optionally: The service for the message received in response. If a timeout occurred before a response was received, this will be empty. This may differ from the service for the message sent.
internalTransactionId	xsd:string	Optionally: The internalTransactionId provided in the request. This is provided in case it is needed for internal correlation.

If a fault occurs, the following fault information will be returned:

Name	Type	Description
faultCode	xsd:string	A code for the fault
faultMessage	xsd:string	A human readable version of the fault

The full XSD for the request, response and fault is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://cisdal.com/schema/PartnerMessages"
  xmlns:tns="http://cisdal.com/schema/PartnerMessages"
  elementFormDefault="qualified">
  <xsd:complexType name="Request">
    <xsd:sequence>
      <xsd:element name="conversationId" type="xsd:string"
minOccurs="0"></xsd:element>
      <xsd:element name="messageId" type="xsd:string" minOccurs="0" ></xsd:element>
```

```

    <xsd:element name="message" type="xsd:string"></xsd:element>
    <xsd:element name="fromPartyId" type="xsd:string"></xsd:element>
    <xsd:element name="fromPartyType" type="xsd:string"></xsd:element>
    <xsd:element name="fromPartyRole" type="xsd:string"
minOccurs="0"></xsd:element>
    <xsd:element name="toPartyId" type="xsd:string"></xsd:element>
    <xsd:element name="toPartyType" type="xsd:string"></xsd:element>
    <xsd:element name="toPartyRole" type="xsd:string"
minOccurs="0"></xsd:element>
    <xsd:element name="cpaId" type="xsd:string"></xsd:element>
    <xsd:element name="action" type="xsd:string"></xsd:element>
    <xsd:element name="service" type="xsd:string"></xsd:element>
    <xsd:element name="timeout" type="xsd:long" minOccurs="0"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="MessageRequest" type="tns:Request"></xsd:element>
  <xsd:complexType name="Response">
    <xsd:sequence>
      <xsd:element name="conversationId" type="xsd:string"
minOccurs="0"></xsd:element>
      <xsd:element name="messageId" type="xsd:string" ></xsd:element>
      <xsd:element name="message" type="xsd:string" minOccurs="0"></xsd:element>
      <xsd:element name="action" type="xsd:string"></xsd:element>
      <xsd:element name="service" type="xsd:string"></xsd:element>
    </xsd:sequence>
  </xsd:complexType>
<xsd:element name="MessageResponse" type="tns:Response"></xsd:element>
<xsd:complexType name="Fault">
  <xsd:sequence>
    <xsd:element name="faultCode" type="xsd:string"></xsd:element>
    <xsd:element name="faultMessage" type="xsd:string"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="MessageFault" type="tns:Fault"></xsd:element>
</xsd:schema>

```

submitAsyncMessageByValue

The submitAsyncMessageByValue message is similar to the submitSyncMessageByValue message, except the call does not wait for a business level response to the message sent.

The submitAsyncMessageByValue call should be used in cases where you do not expect a response to a business message (such as a notification message), or where the response may follow a significant period of time later and you do not want the caller to block waiting for a response.

Although the submitAsyncMessageByValue does not wait for a business response, the ebXML

handshake does ensure that the message is delivered to the trading partner, and that an ebXML acknowledgement is received.

The submitAsyncMessageByValue Web Service can be found at:

<http://localhost:8080/CisdalMessageGateway/submitAsyncMessageByValueService?wsdl>

The request information accepted by this web service is identical to the information provided to submitSyncMessageByValue, except the timeout value will be ignored if provided.

The following information will be received in response:

Name	Type	Description
conversationId	xsd:string	Optionally: The conversation id to be linked to this message, and any subsequent messages related to this business level conversation.
sendingMessageId	xsd:string	The message id for the message sent to the trading partner. This can be used for manual correlation if an asynchronous response is ever received for this message.
internalTransactionId	xsd:string	Optionally: The internalTransactionId provided in the request. This is provided in cases where a response has been received asynchronously for a request you initiated.

The fault information and xsd for this message are identical to submitSyncMessageByValue.

findMessageByMessageId

This Web Service is designed to return a single message as identified by its unique ebXML message id. If no message is returned, this Web Service delivers a fault to the caller. The message may be either a message that has been sent or received.

The findMessageByMessageId service is located at the following location:

<http://localhost:8080/CisdalMessageGateway/findMessageByMessageIdService?wsdl>

This service accepts the following parameters:

Name	Type	Description
messageId	xsd:string	The ebXML message id

This service returns the following information if a matching message is found:

Name	Type	Description
messageId	xsd:string	The ebXML message id
message	xsd:string	The message payload associated with the message

conversationId	xsd:string	Optionally: The message conversation id
service	xsd:string	The ebXML service that the message was sent to or received at. Along with action, this provides an indication of the type of the message.
action	xsd:string	The ebXML action that the message was sent to or received at.
internalPartyId	xsd:string	The party associated with the message at your end.
partnerPartyId	xsd:string	The trading partner associated with the message.
direction	xsd:string	One of the following values: Inbound Outbound Indicating whether this is a message you sent or received.
status	xsd:string	The ebXML status of the message. This will be one of: Pending Processed Delivered Delivery Failure Processed Processed Error
isSyncResponseExpected	xsd:boolean	Was this message sent expecting a synchronous response? This is only applicable for outbound messages.
isSyncResponseReceived	xsd:boolean	Was a synchronous response received? This is only applicable for outbound messages.
date	xsd:dateTime	The date/time the message was sent (in the case of outbound messages), or received (in the case of inbound messages).
internalTransactionId	xsd:string	Optionally: The internal transaction id you associated with the message.

This response definition is identical for all other “find” Web Services, except all other searches potentially return multiple results.

If a fault occurs, including if no message is found, the following is returned:

Name	Type	Description
faultCode	xsd:string	The code associated with the fault
faultMessage	xsd:string	A human readable message associated with the fault

If no message is found, the fault code will be:

MESSAGE_NOT_FOUND

The XSD supporting this Web Service is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://cisdal.com/schema/Message"
  xmlns:tns="http://cisdal.com/schema/Message"
  elementFormDefault="qualified">

  <xsd:simpleType name="direction">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="inbound"/>
      <xsd:enumeration value="outbound"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="status">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pending"/>
      <xsd:enumeration value="Processed"/>
      <xsd:enumeration value="Delivered"/>
      <xsd:enumeration value="Delivery Failure"/>
      <xsd:enumeration value="Processed"/>
      <xsd:enumeration value="Processed Error"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="MessageDefinition">
    <xsd:sequence>
      <xsd:element name="messageId" type="tns:messageId"/>
      <xsd:element name="message" type="xsd:string"/>
      <xsd:element name="conversationId" type="tns:conversationId"/>
      <xsd:element name="service" type="xsd:string"/>
      <xsd:element name="action" type="xsd:string"/>
      <xsd:element name="internalPartyId" type="xsd:string"/>
      <xsd:element name="partnerPartyId" type="xsd:string"/>
      <xsd:element name="direction" type="tns:direction"/>
      <xsd:element name="status" type="tns:status"/>
      <xsd:element name="isSyncResponseExpected" type="xsd:boolean"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```

        <xsd:element name="isSyncResponseReceived" type="xsd:boolean"></xsd:element>
        <xsd:element name="date" type="xsd:dateTime"></xsd:element>
        <xsd:element name="internalTransactionId" type="tns:internalTransactionId"
minOccurs="0"></xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="Message" type="tns:MessageDefinition"
nillable="true"></xsd:element>

    <xsd:simpleType name="messageId">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="internalTransactionId">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="conversationId">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:element name="messageId" type="tns:messageId" nillable="true"></xsd:element>
    <xsd:element name="conversationId" type="tns:conversationId"
nillable="true"></xsd:element>
    <xsd:element name="internalTransactionId" type="tns:internalTransactionId"
nillable="true"></xsd:element>
    <xsd:complexType name="FaultDefinition">
        <xsd:sequence>
            <xsd:element name="faultCode" type="xsd:string"></xsd:element>
            <xsd:element name="faultMessage" type="xsd:string"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="faultContents" type="tns:FaultDefinition"
nillable="true"></xsd:element>
</xsd:schema>

```

findMessageListByConversationId

This Web Service is designed to return all messages relating to a conversation id. Conversation Ids are ebXML conventions for grouping messages that combined form a business transaction. For instance, all messages relating to a single Sales Order or Purchase Order. It is entirely up to the trading partners to decide what constitutes a business transaction, and should be covered by the same conversation id.

The findMessageListByConversationId service is located at the following location:

<http://localhost:8080/CisdalMessageGateway/findMessageListByConversationIdService?wsdl>

This service accepts the following parameters:

Name	Type	Description
conversationId	xsd:string	The ebXML conversation id

The findMessageListByConversationId returns a MessageListDefinition which contains zero or more MessageDefinition elements. MessageDefinition are identical in structure to the definition defined in the findMessageByMessageId section above.

Likewise, the fault definition is identical to that defined for findMessageByMessageId, however this Web Service does not return a fault if no records are found.

The xsd for MessageListDefinition is as follows:

```
<xsd:complexType name="MessageListDefinition">
  <xsd:sequence>
    <xsd:element name="message" type="tns:MessageDefinition"
maxOccurs="unbounded" minOccurs="0"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="MessageList" type="tns:MessageListDefinition"
nillable="true"></xsd:element>
```

All other find operations defined below contain this same definition for returning zero to many results.

findSyncMessagesByOriginalMessageId

This Web Service is designed to return a message originated by you, along with the response (if any) received to this message. The parameter expected by this Web Service is the message id relating to your original request.

ebXML is asynchronous in nature, however, it is very common to still perform conventional request/response messaging over this asynchronous channel. For instance, a submitted Sales Order will often be followed by a Sales Order confirmation.

This Web Service is intended to return zero, one or two messages:

zero messages are returned if the message id does not correspond to a message you have sent

one message is returned if no response was received to your message

two messages are returned if a response was delivered to your message

The findSyncMessagesByOriginalMessageId service is located at the following location:

<http://localhost:8080/CisdalMessageGateway/findSyncMessagesByOriginalMessageIdService?wsdl>

This service accepts the following parameters:

Name	Type	Description
messageId	xsd:string	The ebXML message id

This service returns identical results and fault information as the findMessageListByConversationId

message.

findSyncMessagesByInternalTransactionId

This Web Service is designed to return a message originated by you, along with the response (if any) received to this message. The parameter expected by this Web Service is the internal transaction id provided by your internal application.

This Web Service is intended to return zero, one or two messages:

zero messages are returned if the message id does not correspond to a message you have sent

one message is returned if no response was received to your message

two messages are returned if a response was delivered to your message

The findSyncMessagesByInternalTransactionId service is located at the following location:

<http://localhost:8080/CisdalMessageGateway/findSyncMessagesByInternalTransactionIdService?wsdl>

This service accepts the following parameters:

Name	Type	Description
internalTransactionId	xsd:string	Your internal transaction id

This service returns identical results and fault information as the findMessageListByConversationId message.

findMessageListByMessageSearch

The search services listed above all solve a specific scenario. A generic search mechanism is also provided that can be used in all other scenarios.

The findMessageListByMessageSearch service takes a multitude of criteria as its input. Any combination of criteria may be provided, since all criteria are optional.

The following are the criteria that can be provided:

Name	Type	Description
conversationId	xsd:string	An ebXML conversation id
messageContents	xsd:string	A portion of text to search for from the message payload. Wildcards are added to the front and end of the search string. This can be used for looking for messages with particular business meaning such as order ids or customer names.
service	xsd:string	The ebXML service the message relates to
action	xsd:string	The ebXML action the message relates to
internalPartyId	xsd:string	The party id on your side involved in the message flow (either receiver or sender)
partnerPartyId	xsd:string	The party id on your partner's side involved in the message flow (either receiver or sender)

direction	Enumeration	One of: inbound outbound This specifies whether the message was sent or received by you.
status	Enumeration	One of: Pending Processed Delivered Delivery Failure Processing Error This specified the current status of the message.
isSyncResponseExpected	xsd:Boolean	Was a response expected to the message. This only applies to messages originated by you.
isSyncResponseReceived	xsd:Boolean	Was a response received to this message. This only applies to messages originated by you.
fromDate	xsd:dateTime	A date/time that the message was created after
toDate	xsd:dateTime	A date/time that the message was created before
maximumResults	xsd:int	The maximum number of results to return. This is the only parameter that has a default value – the default being 500.

This service returns identical results and fault information as the findMessageListByConversationId message.

Receiving Messages Asynchronously

A handler is required for providing messages to your internal applications when messages are not received as synchronous responses to messages you have originated. This includes the following scenarios:

1. Unsolicited messages from Trading Partners
2. Responses to synchronous messages that are received after your session has timed out
3. Responses to messages you have sent to Trading Partners asynchronously

In order to receive asynchronous messages, the following interface should be implemented:

```
public interface AsyncReceiverHandler {
    public void handleMessage(MessageHolder mh) throws UndeliverableMessageException;
    public void handleUndeliverableMessage(MessageHolder mh);
}
```

Cisdal implement this interface for all customers as part of their service, but this can be extended by customers if required.

If you implement this directly, the implementation should be placed in either the classes or lib folder under WEB-INF within the Cisdal Message Gateway web application.

Cisdal can provide implementation of this interface to deliver messages to your internal applications via interfaces including:

- SOAP Web Service

- JMS

- EJB

- RMI

For SOAP and JMS Based implementations, we have XML formats defined for providing message information to your applications.

All Cisdal implementations provide guaranteed messaging. If an error occurs delivering a message, the message will be retried until it succeeds (or until a configurable maximum number of retries is exceeded).

Please consult with us regarding the interfaces exposed by your internal applications, and we will provide the required implementation.

When we deliver messages to your internal application, the following information can be provided:

Message Id: The unique ebXML message Id

Originating Message Id: If this is a response to a message you originated, the ebXML message Id of that message

Originating Internal Transaction Id: If this is a response to a message you originated, the internal transaction Id you provided for the message

Message: The message received

Conversation Id: The conversation id corresponding to the message

Service: The ebXML service that received this message, along with the action this can be used for determining the message type

Action: The ebXML service that received this message

Cisdal recommend using the JMS implementation where possible. Because JMS provides its own level of persistence, this implementation guarantees that messages won't be lost if messages are received while your internal applications are unavailable.

Monitoring

There are two forms of monitoring that can be performed:

Log files should be monitored. The key log files are the application server log files and the Hermes2 log files. Hermes2 log files can be found in the hermes2/logs directory. The most important of these is ebms.log. This should be monitored for errors on an on-going basis.

The Cisdal Message Gateway also provides an Alerting framework. This is used for notifying you that significant events have occurred regarding messages. The key events that you will be notified of are:

- The Delivery of an ebXML message failed

- The Processing of an ebXML message failed

- An ebXML message received a negative acknowledgement

These alerts are provided through a variety of interfaces (e.g. email, log files), and we can work with you to incorporate these alerts with any existing alerting infrastructure that you might have.

Alerts represent events where human intervention is required. Alerts are not raised if the system can automatically perform retries. It is critical that alerts are monitored on a regular basis since they should not occur in a correctly functioning system.

Housekeeping

The following house keeping rules should be observed:

- Ensure that your database is backed up on a regular basis

- Ensure that your database has its statistics kept up to date on at least a daily basis

- Ensure the message inbox and message outbox folders are kept to a manageable size. We recommend archiving these after every 5000-10000 messages. Cisdal can provide a utility for performing this archiving.

Stopping the Cisdal Message Gateway

The Cisdal Message Gateway contains a web page that allows the Gateway to be stopped without stopping the Application Server. This will prevent your internal applications sending messages, and will mean no messages are delivered to your internal applications.

This web page can be found at the following address.

<http://localhost:8080/CisdalMessageGateway/>

Users can only execute the start and stop functionality if they have the role of admin (in Tomcat this is defined in the tomcat-users.xml file mentioned above).

The same page can be used for starting the gateway again.

Cisdal Message Gateway Test Harness

Cisdal offer a Test Harness along with the Cisdal Message Gateway that can be used for sending and receiving messages from a partner.

The Cisdal Test Harness is a Java Application that can be loaded via Web Start from

<http://www.cisdal.com/xmlrunner/xmlrunner.jnlp>

The Cisdal Test Harness allows XML Messages to be sent through the Cisdal Message Gateway to a remote client. The Harness will wait for a synchronous response to arrive if required, and correlate requests to responses.

The Test Harness can be used for both functional testing and performance/stress testing.

The Test Harness is also typically used for testing a Cisdal Message Gateway/Hermes2 configuration before it is integrated with internal applications

FAQ

Does the Cisdal Message Gateway support clustering?

We do not currently support clustering if you wish to receive messages synchronously. This is a planned enhancement for Q4 2009.

How do I request extensions to the Cisdal Message Gateway?

Email us at info@cisdal.com

How do I raise defects against the Cisdal Message Gateway?

Email us at defects@cisdal.com

What are the typical throughput rates handled by the Cisdal Message Gateway?

The Cisdal Message Gateway is typically used in scenarios with a throughput rate less than 1000 messages per hour. The Cisdal Message Gateway is not currently recommended for throughputs of greater than 2000 messages per hour due to the fact it does not support clustering.

What situations do you/don't you recommend the Cisdal Message Gateway for?

We recommend the Cisdal Message Gateway for situations where you need to use ebXML to perform functionality required by your primary applications, not for situations where the gateway is a primary application.

Has Cisdal Message Gateway/Hermes2 been tested against Websphere Partner Gateway?

Yes, we have tested Cisdal Message Gateway/Hermes2 against Websphere Partner Gateway and proven compliance. Our patch sets resolve any issues that will occur when communicating with Websphere Partner Gateway.

How should we implement encryption when sending/receiving messages to remote partners?

We recommend using SSL rather than XML Encryption for securing traffic. We have found SSL to provide superior performance.