

Commerce One HotFS™

Installation and Configuration Guide

Version 4.0



Hacienda Business Park, Bldg # 4
4440 Rosewood Drive
Pleasanton, CA 94588

Commerce One MarketSite™

HotFS Installation and Configuration Guide

Version 4.0

© 2000 Commerce One, Inc.

All rights reserved.

Copyright © 2001. Commerce One, Many Markets. One Source. Global Trading Web, Commerce One.net, MarketSite, XML Common Business Library, XML Development Kit, eLink, Net Market Maker, RoundTrip, and SupplyOrder are either trademarks or registered trademarks of Commerce One, Inc. Enterprise Buyer and MarketSet are trademarks of Commerce One, Inc and SAPMarkets. All other company, product, and brand names are trademarks of their respective owners.

Preface

Purpose of this Book

The *HotFS Installation and Administration Guide* explains how to install, configure and use HotFS.

Who Should Read this Book

This document is intended for CommerceOne System administrators and customers.

What's Included in this Book

The HotFS installation and configuration procedures are described in the order that you are most likely to encounter them.

Chapter 1 Introduction

Chapter 1 provides a brief overview of HotFS.

Chapter 2 HotFS Requirements

Chapter 2 describes the system requirements and the MarketSite setup requirements.

Chapter 3 Client/Service Description and Trading Partner Requirements

Chapter 3 describes the functionality of a client and service and lists the Trading Partner requirements for a HotFS Client installation.

Chapter 4 Setup Scenarios

Chapter 4 describes the four typical setup scenarios for HotFS: Simple Buyer, Supplier, Advanced Buyer, Buyer and Supplier.

Chapter 5 Configuring the XCC

Chapter 5 describes how to reconfigure the XCC server from the default setup.

Chapter 6 Installing and Uninstalling HotFS

Chapter 6 describes how to install and uninstall HotFS.

Chapter 7 Running the HotFS Client and Service

Chapter 7 describes how to run both the HotFS client and service.

Chapter 8 Changing the HotFS Configurations

Chapter 8 describes how to modify HotFS Client and Service configurations.

Chapter 9 TDF Files

Chapter 9 describes the TDF Document and provides a sample TDF instance.

Chapter 10 RDF Files

Chapter 10 describes the Reply Description Files (RDF).

Chapter 11 Result Files

Chapter 11 describes incoming files and error files.

Appendix A HotFS Configuration Schema

Appendix B HotFS Service Config Schema

Appendix C TDF Schema

Appendix D RDF Schema

Appendix E DDF Schema

Appendix F HotFS Error Schema

Appendix G Mapping of File Extensions to Mime Schema

Appendix H UUID Generation Classes

Appendix I encryptpassword Tool

Related Information

In this book, the terms below are defined as:

Enter	Refers to typing letters or numbers on the computer keyboard. If upper or lower case is mandatory, this is stated. If it is not, then either may be used.
Press	Refers to pressing one of the special keys on the computer keyboard, such as Tab, Ctrl or Alt. If it is necessary to press-and-hold a special key followed by another key, this is stated.
Click	Refers to positioning the mouse pointer (or cursor) over a screen button image and clicking the left mouse button. If pressing the right mouse button is required, this is stated as right-click.

HotFS Technical Support

Please write the names and numbers for your installation's technical support contact personnel below:

You can find details of the version in the version.txt file located in:
XCCROOT\HotFS\etc\version.txt

If you cannot resolve a problem using the HotFS manual, contact your technical support representative and ask him/her to contact Commerce One Technical Support by email at **csc@commerceone.com** or by telephone **(925-941-5959)**.

Contents

Preface	i
Purpose of this Book	i
Who Should Read this Book	i
What's Included in this Book	i
Related Information	iii
HotFS Technical Support	iv
Chapter 1: Introduction.....	1-1
In This Chapter	1-1
Introduction to HotFS	1-1
Chapter 2: HotFS Requirements	2-1
In This Chapter	2-1
System Requirements	2-1
MarketSite Registration Requirements	2-1
Chapter 3: Client/Service Description and Trading Partner Requirements	3-1
In This Chapter	3-1
Overview.....	3-1
Client Description	3-2
Trading Partner Requirements for HotFS Client	3-5
Service Description.....	3-6
Trading Partner Requirements for HotFS Service	3-10

Chapter 4: Setup Scenarios	4-1
In This Chapter	4-1
Simple Buyer	4-2
Supplier	4-3
Advanced Buyer	4-4
Buyer and Supplier	4-6
Chapter 6: Configuring the XCC Server	5-1
In This Chapter	5-1
Configuring the XCC Server	5-1
Xcc Security Setup.....	5-1
Chapter 6: Installing and Uninstalling HotFS	6-1
In This Chapter	6-1
Installing HotFS	6-1
Uninstalling HotFS.....	6-2
Chapter 7: Running the HotFS Client and Service.....	7-1
In This Chapter	7-1
Running the HotFS Client.....	7-1
Starting the Client	7-1
HotFS Client Options	7-1
HotFS Client Troubleshooting	7-3
Running the HotS Service	7-8
HotFS Service Initialization Properties	7-8
HotFS Service Troubleshooting	7-10
Chapter 8: Changing the HotFS Configurations	8-1
In This Chapter	8-1
Quick Reference for Changing HotFS Configurations	8-1
Client Configuration Quick Reference.....	8-1
Service Configuration Quick Reference	8-3
Changing the HotFS Client Configuration	8-5
Client Configurations Element	8-5
Transmission Configurations Element	8-5
Sample HotFS Configurations Instance.....	8-16

Changing the HotFS Service Configurations.....	8-18
Directory Locations Element.....	8-18
Explanation of Service Configuration.....	8-23
Sample HotFS Service Config Instance.....	8-25
Chapter 9: TDF Files	9-1
In This Chapter	9-1
Overview	9-1
TDF Element	9-1
Explanation of Transmissions Properties.....	9-2
Explanation of Root Document	9-7
Explanation of Attachment	9-9
Sample TDF Instance	9-11
Chapter 10: RDF Files	10-1
In This Chapter	10-1
Overview	10-1
RDF	10-1
Explanation of TaskID.....	10-2
Explanation of Root Document	10-2
Sample RDF Instance	10-4
Chapter 11: Result Files	11-1
In This Chapter	11-1
Overview	11-1
Incoming Files	11-1
Replies	11-1
Incoming Requests	11-2
Sample DDF Instance.....	11-3
Error File.....	11-4
Sample Error Instance	11-4

Appendix A: HotFS Configurations Schema	A-1
Appendix B: HotFS Service Config Schema.....	B-1
Appendix C: TDF Schema	C-1
Appendix D: RDF Schema	D-1
Appendix E: DDF Schema	E-1
Appendix F: HotFS Error Schema	F-1
Appendix G: Mapping of File Extensions to Mime Types	G-1
Appendix H: UUID Generation Classes.....	H-1
Appendix I: encryptpassword Tool	I-1

Chapter 1

Introduction

In This Chapter

This chapter provides a brief description of HotFS.

Introduction to HotFS

HotFS enables you to communicate with Commerce One MarketSite without integrating an existing Trading Partner system directly with the Commerce One XML Commerce Connector (XCC).

HotFS serves as a connector between the Trading Partner system and MarketSite, and communicates with the Trading Partner system via the file system. You can use HotFS to integrate legacy systems or systems unable to use the XCC java code directly.

HotFS contains the following components:

- **HotFS Client** - intended for any communication initiated by the Trading Partner
- **HotFS Service** - intended for any communication initiated by MarketSite

This document describes how to install, configure and administer the HotFS client and service.

Chapter 2

HotFS Requirements

In This Chapter

This chapter describes the system requirements and the MarketSite setup requirements.

System Requirements

Since the HotFS solution is a customization of XCC, the system that you use to run the HotFS Client and/or Service must have a fully functional installation of XCC 3.2.

For information on installing XCC see the following manuals:

- *XCC Installation and Administration Guide, v. 3.1, 2nd Edition*
- *XCC 3.2 Upgrade Installation Guide*

Note This also includes a fully functional installation of both Cygwin and Microsoft Java Virtual Machine (JVM) which has been completed as one of the prerequisites for installing XCC.

MarketSite Registration Requirements

The Trading Partner (buyer and/or supplier) must be appropriately registered with the particular MarketSite that HotFS will communicate with. This includes receiving a valid Market Participant ID (MPID) from the MarketSite administrator.

If the trading partner wants to fulfill both the buyer and supplier roles, a valid MPID must be received for each of those roles.

If the trading partner wants to fulfill a supplier role, the MarketSite administrator must set up the supplier to be an integrated supplier in order for HotFS to successfully receive the requests meant for that supplier.

Chapter 3

Client/Service Description and Trading Partner Requirements

In This Chapter

This chapter describes the functionality of a client and service and lists the Trading Partner requirements for a HotFS Client installation.

Overview

The HotFS Client and Service are installed on top of a basic installation of XCC 3.2. They perform the following functions:

- **Client** - handles any outgoing transmissions initiated by the Trading Partner, and incoming synchronous replies from the receiving MarketSite.
- **Service** - handles any incoming traffic, both asynchronous replies to previous client requests, and new incoming requests. The service also handles any replies the Trading Partner needs to send in reply to previous incoming requests.

Note In order for HotFS to function appropriately, the Trading Partner must fulfill certain requirements in its communication with HotFS. See *Trading Partner Requirements for HotFS Client* on page 3-5.

Client Description

The client sends documents from the Trading Partner to MarketSite. It determines the specific tasks to perform by polling a specified directory in the file system for Task Description Files (TDF) placed there by a Trading Partner Application by some means, for example ftp. Each TDF describes a transmission task.

When receiving a task, the client will assemble the documents to send in an envelope, and send the envelope to a MarketSite server. The client can be configured to poll for tasks only once or on a continuous basis.

The client can also accept replies to synchronous requests it sends out. The client is unable to accept replies to asynchronous requests. Asynchronous replies can be handled by the HotFS Service, if so desired.

The client operates according to a set of predictable principles that allows the Trading partner application to determine the result of any transmission:

1. When the TDF file has been read, it is immediately archived in a TDF archiving directory in the file system.
2. The client assembles the document and any attachments to be sent, and configures the transmission settings, as specified in the TDF and the client configurations, and sends the document to MarketSite.
3. If the document is successfully sent, the document, and any attachments, are immediately archived in a document archiving directory in the file system.
4. If a reply to a synchronous transmission is received, it is placed in a reply document directory in the file system. In addition, a Document Description File (DDF) is created and placed in a reply directory.
5. If any errors occur, the document and any attachments are not archived, and an error document will be placed in an error directory in the file system. The client will not attempt to re-transmit. The Trading partner is the initiator of any transmissions.

The files are routed through the file system according to pre-defined, predictable rules. Figures 3-1 and 3-2 show the flow of the documents in the HotFS Client Sending and HotFS Client Receiving.

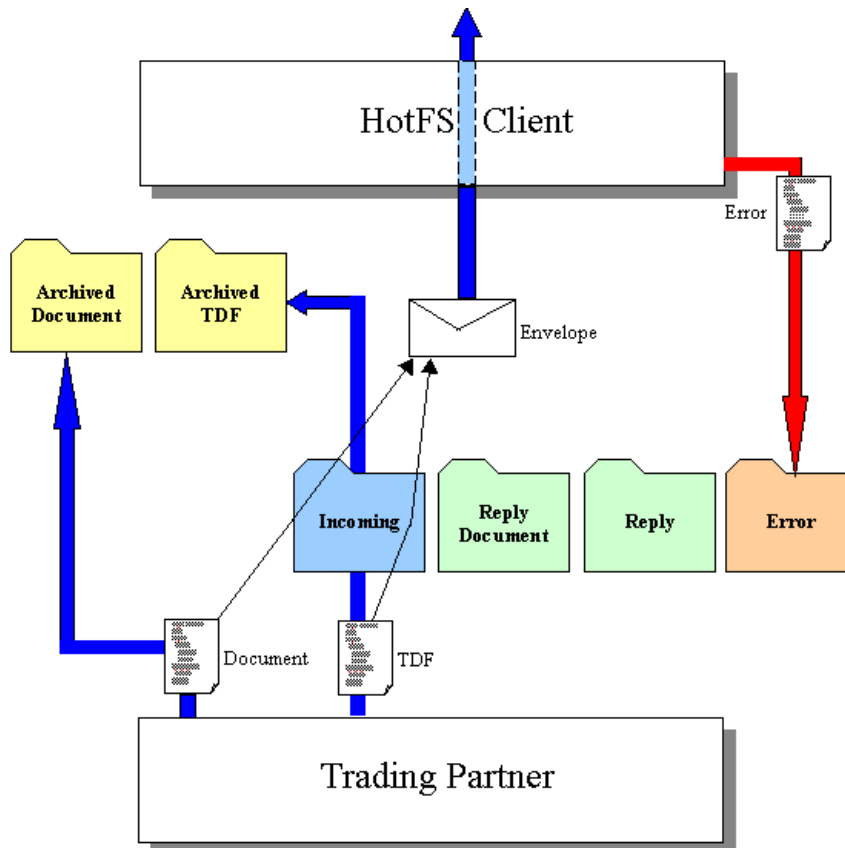


Figure 3-1 HotFS Client Sending

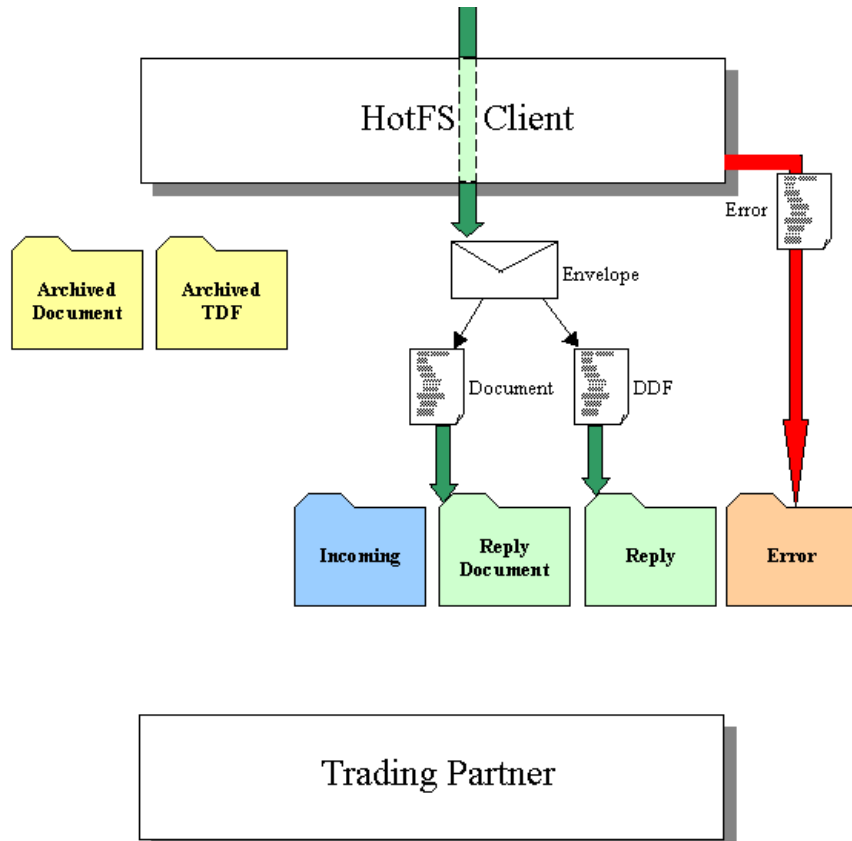


Figure 3-2 HotFS Client Receiving

Every task has a specified Task ID, and all files related to a task, will be named according to this ID. This allows the Trading Partner application to poll for error and reply files, and associate these files with the original task.

Trading Partner Requirements for HotFS Client

In order for the HotFS Client application to function properly, the Trading Partner Application (TPA) must fulfill the following requirements:

1. The TPA must notify the HotFS Client that a document should be sent, by placing a TDF in the "incoming" directory specified in the client configuration file.
2. The TDF file must be a valid XML instance of the schema TDF.sox in order for the task to be processed. (Appendix C).
3. The TDF file should have the name <TaskID>.tdf. The TaskID must be globally unique within MarketSite.
4. The documents specified in the TDF file must exist in the specified location in the file system.
5. The TDF must be placed in the file system after any files that are referenced in the TDF. The placement of the TDF is a signal that the document is ready for transmission. This is to ensure that all needed documents are present at the time the TDF is read and the envelope is prepared.
6. The document to be transmitted must be valid XML, conforming to the SOX schema it is an instance of. In addition, it must also fulfill any additional requirements the receiving service may have.
7. The trading partner is responsible for removing any archived files, error files or reply files from the file system. This can be done when a task is considered completed. This is not absolutely necessary, but would be the ideal behavior to avoid large amounts of files slowing down performance.

Service Description

The service listens for incoming transmissions. These transmissions may be incoming requests or asynchronous replies to requests previously sent out by the client. Any incoming transmissions will result in a Document Description File (DDF) placed on the file system, for the Trading Partner Application.

The service operates according to the following principles when receiving incoming transmissions:

1. When a transmission has been received, the service places the document(s) on the file system.
2. If the received document can be successfully unwrapped, the service creates a Document Description File (DDF) and places it in the file system. This is done last, to signal that a document has arrived.
3. If the document is in a response mode that will warrant a reply, the envelope is stored. The envelope will be re-used to ensure the correct delivery of a reply.
4. If any errors occur, an error document will be placed in an error directory in the file system, and an error returned to the sender.

The files will move through the file system according to pre-defined, predictable rules. Figures 3-3 shows the flow of the documents in the HotFS Service Receiving.

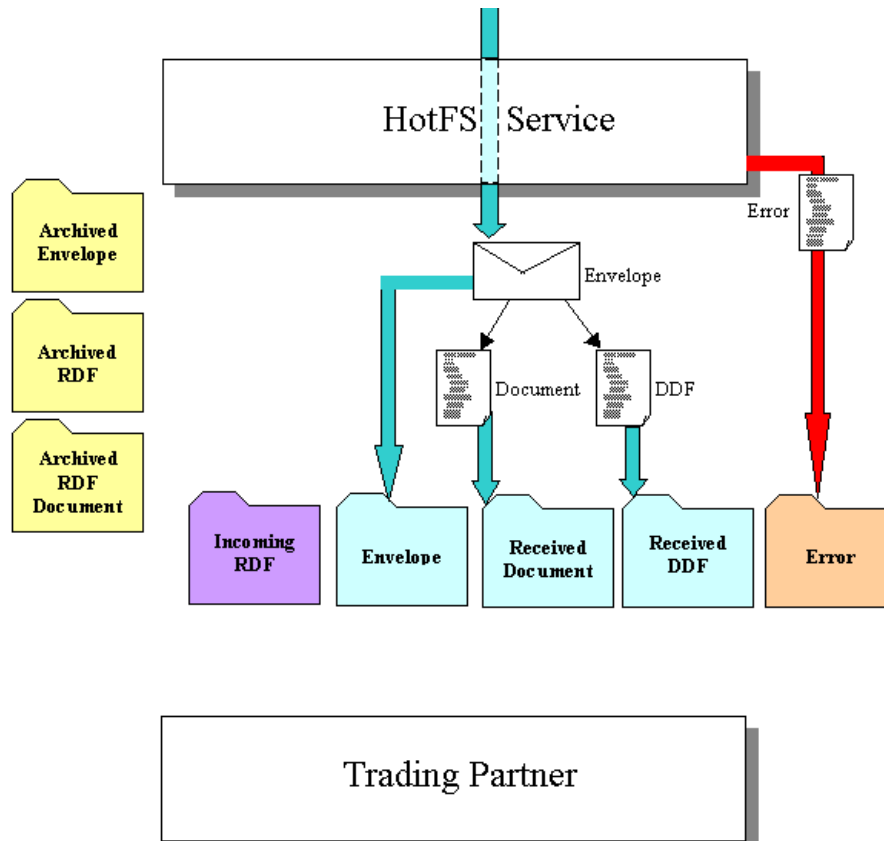


Figure 3-3 HotFS Service Receiving

The service also simultaneously polls the file system for trading partner replies to previous incoming transmissions. These replies are signaled with a Reply Description File (RDF), just as a client task was signaled with a TDF. When an RDF is encountered, the service will assemble a reply and send it to MarketSite.

The service handles replies according to the following principles:

1. When an RDF has been read, it is immediately archived in an RDF archiving directory in the file system.
2. The service gets the original request envelope stored in the file system, and assembles the reply and sends it to MarketSite.
3. If the document is successfully sent, the document and the original envelope are archived in the file system.
4. If any errors occur, the document and envelope are not archived, and an error document will be placed in an error directory in the file system. The service will not attempt to re-transmit the reply.

The files will move through the file system according to pre-defined, predictable rules. Figures 3-4 shows the flow of the documents in the HotFS Service Sending.

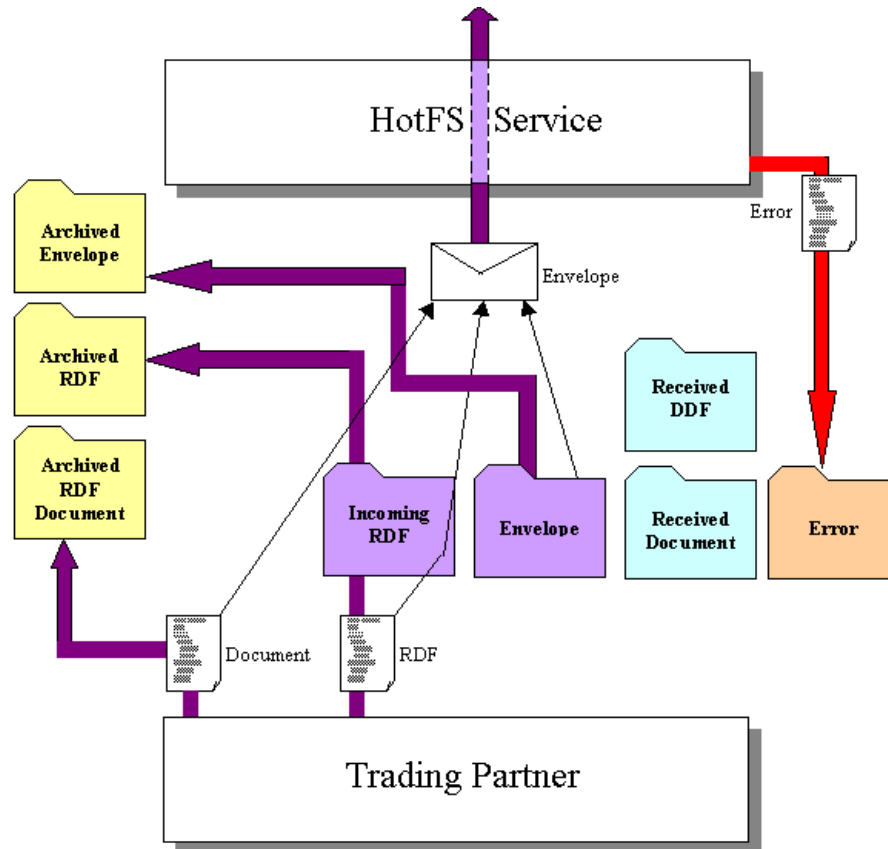


Figure 3-4 HotFS Service Sending

Every task has a specified Task ID, and all files related to a task, will be named according to this ID. This allows the Trading Partner application to poll for error and reply files, and associate these files with the original task.

Trading Partner Requirements for HotFS Service

In order for the HotFS Service to function properly, the Trading Partner Application (TPA) is required to fulfill certain requirements:

1. The TPA must reply to any incoming requests that warrants a reply from MarketSite. Requests that warrant a reply will have the transmission mode of sync or peer-peer. The synchronous requests should ideally be treated with a higher priority than the asynchronous replies, as exceeding the sender's time-out setting will result in the sender not getting a reply.
2. The TPA must notify the HotFS Client that a reply should be sent, by placing an RDF in the "reply" directory specified in the service configuration file.
3. The RDF file must be a valid XML instance of the schema RDF.sox in order for the task to be processed. (Appendix D).
4. The RDF file should have the name <TaskID>.rdf. The TaskID must be globally unique within MarketSite. This ID must be identical to the ID of a previous incoming request.
5. The document specified in the RDF file must exist in the specified location in the file system.
6. The RDF must be placed in the file system after any files that are referenced in the RDF. The placement of the RDF is a signal that the document is ready for transmission. This is to ensure that all needed documents are present at the time the RDF is read and the envelope is prepared.
7. The document to be transmitted must be valid XML, conforming to the SOX schema it is an instance of. In addition, it must also fulfill any additional requirements the receiving service may have.
8. The trading partner is responsible for removing any archived files, error files or reply files from the file system. This can be done when a task is considered completed. This is not absolutely necessary, but would be the ideal behavior to avoid large amounts of files slowing down performance.

Chapter 4

Setup Scenarios

In This Chapter

This chapter describes the four typical setup scenarios for HotFS:

- Simple buyer
- Supplier
- Advanced Buyer
- Buyer and supplier

This overview allows you to select the scenario(s) that best suits your needs.

Simple Buyer

This setup only uses the client, and is intended for the most basic buyer setup. This setup is intended for the buyer who wants to submit documents to MarketSite, and get transmission errors and synchronous replies. This setup is not able to handle asynchronous replies or content errors. The numbers in the illustration are meant to illustrate the typical flow.

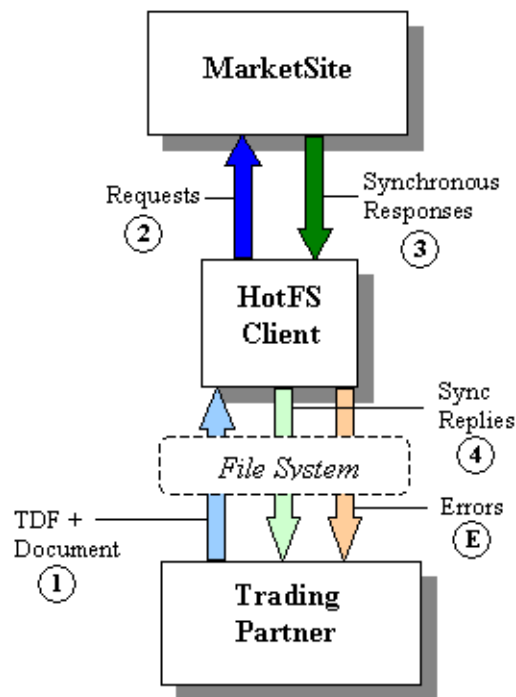


Figure 4-1 Simple Buyer

1. Trading Partner places TDF and document in the file system.
2. HotFS creates the envelope and sends it to MarketSite.
3. MarketSite may return a reply to the HotFS Client if the request was synchronous.

4. HotFS disassembles it and places it in the file system.

E. Errors can occur at any point.

If this installation is the desired option, the Trading Partner (buyer) should acquire a valid buyer Market Participant ID (MPID), and specify this MPID when performing the XCC installation. In this case the HotFS Service is not required. The Trading Partner (buyer) may wish to disable the XCC server that normally starts up on the system by default.

Supplier

The Supplier setup only uses the service, and is intended for a basic supplier setup. This setup is intended for a supplier that wants to accept requests from MarketSite, and send responses back. This setup is not able to handle transmissions initiated by the Trading Partner (supplier).

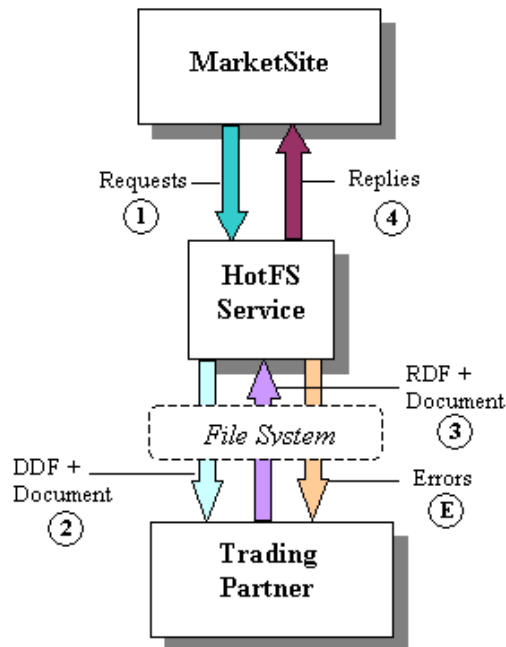


Figure 4-2 Supplier SetUp

The numbers in the illustration illustrate a typical flow.

1. MarketSite sends a request to HotFS.
 2. HotFS disassembles the envelope places the request in the file system.
 3. The Trading Partner returns a reply to HotFS if warranted.
 4. HotFS puts the reply an envelope and returns it to MarketSite.
- E. Errors can occur at any point.

If you want to use the Supplier option, the Trading Partner (supplier) should acquire a valid supplier Market Participant ID (MPID), and get registered as an integrated supplier with the receiving MarketSite. In this case the HotFS Client is not required.

Advanced Buyer

The Advanced Buyer uses both the client and the service, and is intended for the buyer who wants the complete setup on one machine.

The Advanced Buyer Setup is intended for the buyer that wants to submit documents to MarketSite, and get all errors and replies available. This setup handles both synchronous and asynchronous replies as well as transmission and content errors.

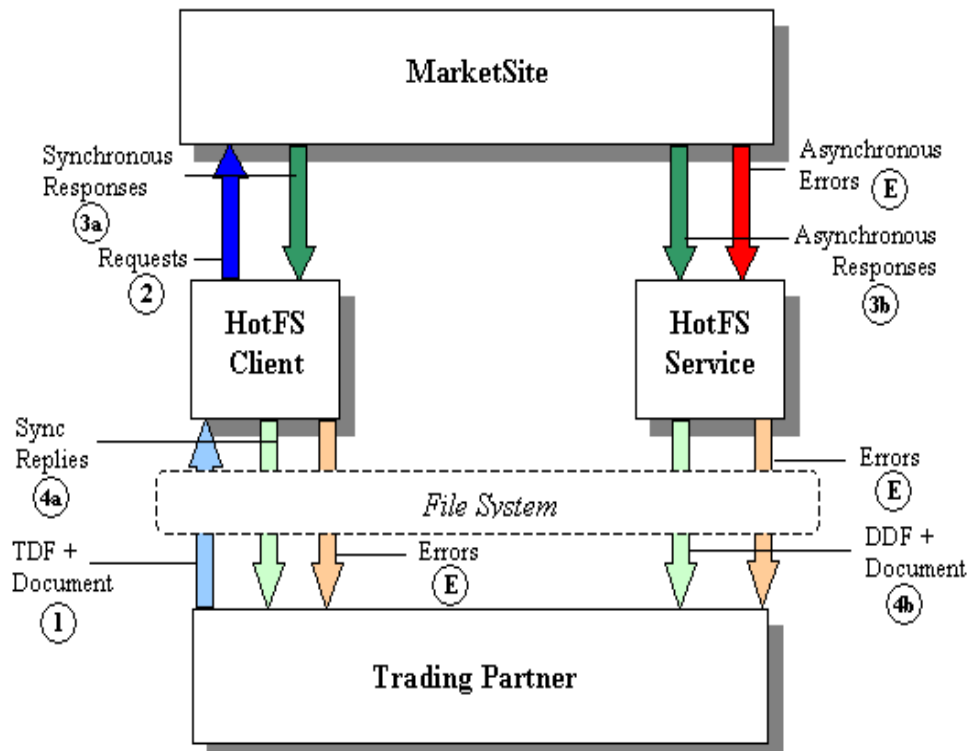


Figure 4-3 Advanced Buyer Setup

The numbers in Figure 4-3, Advanced Buyer Setup illustrate the typical flow:

1. Trading Partner places TDF and document in the file system.
2. HotFS creates the envelope and sends it to MarketSite.
3. MarketSite returns replies in the following manner:
 - a) It returns a reply to the HotFS Client if the request was synchronous.(3a).
 - b) It returns a reply to the HotFS Service if the request was asynchronous. (3b).

4. HotFS Client and HotFS Service disassembles the reply and places it in the file system (4a & 4b).

E. Errors can occur at any point (E).

If this installation is the desired option, the Trading Partner should acquire a valid buyer Market Participant ID (MPID), and specify this MPID when performing the XCC installation.

Buyer and Supplier

This setup uses both the client and the service, and is intended for the trading partner that wants to have a complete buyer and supplier setup located on the same machine. This setup is intended for the buyer and supplier that wants to submit and receive documents to and from MarketSite simultaneously. This setup handles outgoing transmissions, synchronous and asynchronous incoming and outgoing replies and transmission and content errors.

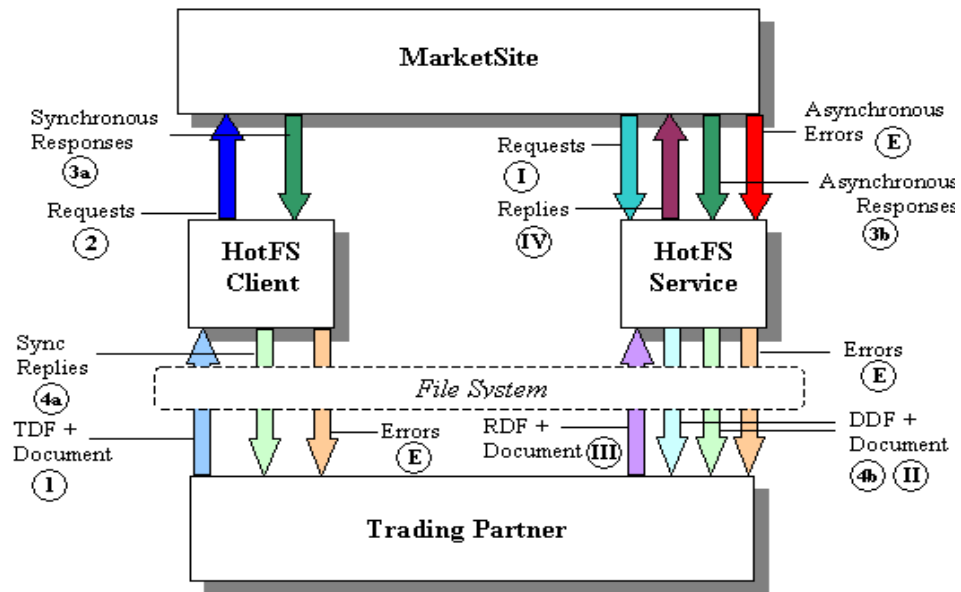


Figure 4-4 Buyer and Supplier Setup

The numbers in the illustration are meant to illustrate the typical flow:

Buyer flow:

1. Trading Partner places TDF and document in the file system.
2. HotFS creates the envelope and sends it to MarketSite.
3. MarketSite handles replies as follows:
 - a) It returns a reply to the HotFS Client if the request was synchronous (3a).
 - b) It returns a reply to the HotFS Service if the request was asynchronous (3b).
4. HotFS disassembles the reply and places it in the file system(4a & 4b).

E. Errors can occur at any point

Supplier flow:

I. MarketSite sends a request to HotFS.

II. HotFS disassembles the envelope and places the request in the file system.

III. The Trading Partner returns a reply to HotFS if warranted,

IV. HotFS puts the reply into an envelope and returns to MarketSite.

E. Errors can occur at any point (E).

If you want to use the Buyer and Supplier option, the Trading Partner should acquire a valid buyer Market Participant ID (MPID) and supplier MPID, and specify the buyer MPID when performing the XCC installation. The Trading Partner must also get registered as an integrated supplier with the receiving MarketSite.

You also have the option of putting the supplier setup on one machine and the advanced buyer setup on a different machine. This allows you to separate incoming buyer traffic from incoming supplier traffic.

Chapter 5

Configuring the XCC Server

In This Chapter

This chapter describes how to reconfigure the XCC server from the default setup.

Note XCC Configuration is handled automatically by the installation process and is transparent to the user. This chapter is provided for information purposes only.

Configuring the XCC Server

If you are using HotFS for accepting new incoming requests, and sending back replies to these requests, you must reconfigure the XCC server from the default setup. This chapter describes the steps required to complete that process.

If you are not using the HotFS Service for transmitting replies to incoming requests, you can proceed to *Chapter 6, Installing and Uninstalling HotFS*.

Any replies that HotFS creates, will be sent through the XCC server in which HotFS is running and back to the MarketSite from which the original request came.

XCC Security Setup

For XCC to successfully connect to MarketSite, and send the response, the XCC Transmitter Service must have the correct security set-up. This ensures that MarketSite will properly authorize XCC. This set-up includes the MPID, user name and password of the trading partner that is using HotFS.

For more information on how to set up the transmitter service, refer to *Chapter 4, Configuring XCC Services* in the *XCC Installation and Administration Guide, v1.1*

The following table lists the keys that you need to set in the Transmitter Service properties file and describes the value for the key setting.

Key	Description
marketparticipantid	<p>The MPID of the trading partner that will be using the HotFS service for replying to incoming requests. For a supplier-only set-up, this is the MPID of the supplier. If HotFS is used for both a buyer and supplier set-up, this is the supplier MPID of the trading partner.</p> <p>Example: If the TradingPartner is both a supplier and buyer, and the trading partner's buyer MPID is 55d8d1d6-77b1-1000-87d6-0a0000200001, and the trading partner's supplier MPID is f9651c42-77b6-1000-9535-ac1408350000, the marketparticipant value should be set as follows:</p> <pre>marketparticipantid=f9651c42-77b6-1000-9535-ac1408350000</pre>
userid	<p>User name for the trading partner specified in marketparticipantid.</p> <p>Example: If the user name of the trading partner is "user", userid is set as follows:</p> <pre>userid=user</pre>

Key	Description
authpref	<p>Type of authentication that will be performed on the receiving end of any transmission that is sent from this installation of XCC. This value should always be set to "uidpswd".</p> <p>Example: The value will always be set as follows:</p> <pre>authpref=uidpswd</pre>
password	<p>Password for the trading partner specified in marketparticipantid. This value should be the corresponding password for the user name specified in userid. The value must have a minimum of 8 characters. The password is encoded in the file and may be encrypted or unencrypted. See <i>Appendix I, encryptpassword Tool</i> for additional information.</p> <p>Example: If the password for the trading partner is "secretword" password will be set as follows:</p> <pre>password=secretword</pre>

Note When you set the key value pairs, do not use any spaces, tabs, etc. between the key, the equal sign or the value.

If you combine the above examples into a complete set of authentication settings, it appears as follows:

```
marketparticipantid=f9651c42-77b6-1000-9535-ac1408350000
userid=user
authpref=uidpswd
password=secretword
```

The Transmitter Service properties file must contain a section similar to the preceding example. However, the section must contain the appropriate values for the trading partner that will be accepting and replying to incoming requests.

Before adding the appropriate key value pairs, verify if the specified keys are present. If they are present, modify your existing values to reflect you setup. The properties file should contain these values only once.

Note If a key-value pair exists with the key "certfile", remove it from the properties file.

Note You can disregard any lines in the properties file starting with "#". These lines are comment lines.

Chapter 6

Installing and Uninstalling HotFS

In This Chapter

This chapter describes how to install and uninstall HotFS.

Note If this is not an initial install of HotFS, you must uninstall the old version of HotFS before reinstalling a new version or installing XCC. Please refer to *Uninstalling HotFS* on page 6-2 for detailed information on how to uninstall HotFS.

Installing HotFS

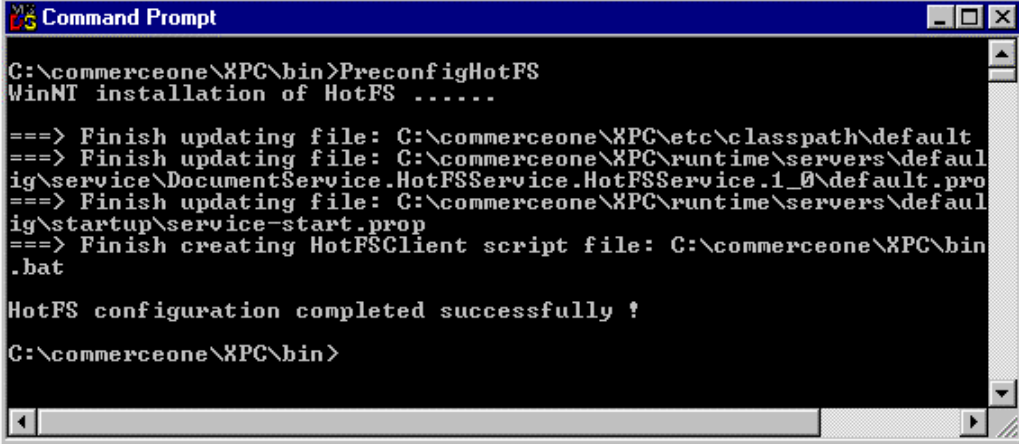
You can install HotFS client and service after XPC has been successfully installed. The following steps describe how to install HotFS

1. Open a DOS command window and change directory to `$XPCROOT\bin`.
2. Invoke the `PreconfigHotFS.bat` program by typing the following command on the command line:

```
$XPCROOT\bin> PreconfigHotFS
```

Press **<Enter>**.

3. If HotFS was successfully installed and configured, the following screen displays:



```
C:\commerceone\XPC\bin>PreconfigHotFS
WinNT installation of HotFS .....

===> Finish updating file: C:\commerceone\XPC\etc\classpath\default
===> Finish updating file: C:\commerceone\XPC\runtime\servers\default
ig\service\DocumentService.HotFSService.HotFSService.1_0\default.pro
===> Finish updating file: C:\commerceone\XPC\runtime\servers\default
ig\startup\service-start.prop
===> Finish creating HotFSClient script file: C:\commerceone\XPC\bin
.bat

HotFS configuration completed successfully !

C:\commerceone\XPC\bin>
```

Figure 6-1 HotFS Completed Successfully

4. The default HotFS client configuration file `config.xml` was copied to `$XPCROOT\bin` directory.

The default HotFS service configuration file `hotfsServiceconfig.xml` was copied to `$XPCROOT\runtime\hotFS` directory. Refer to Chapter 7, *Running the HotFS Client and Service* on how to modify these files to fit your particular setting.

Uninstalling HotFS

The following information describes how to uninstall HotFS client service.

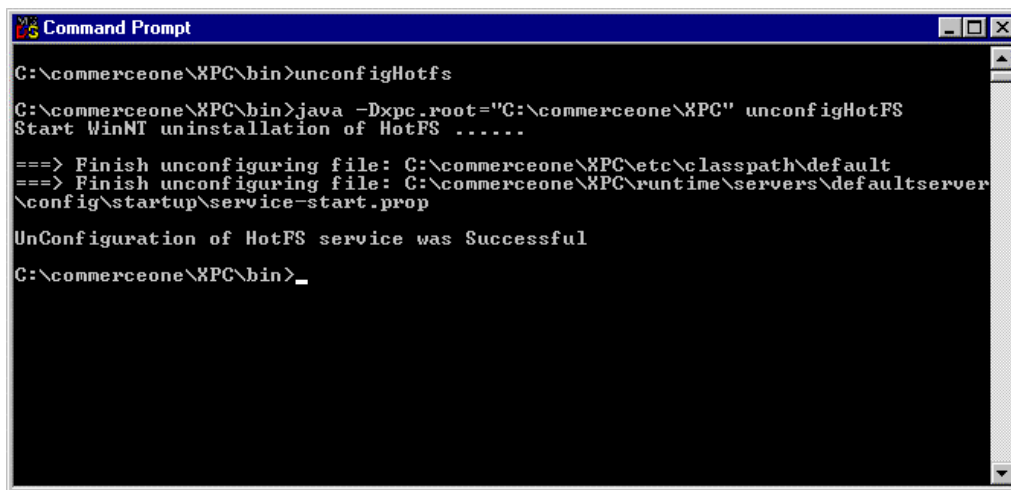
1. Before you uninstall HotFS, you must shut down the client and XPC.
 - a) To shut down the client, open a DOS window where the client is running and press **CTRL-C**.
 - b) To shut down XPC, open a Cygnus window where XPC is running and press **CTRL-C**.
2. Open a DOS command window and change directory to `$XPCROOT\bin`.

3. Invoke the unconfigHotFS.bat program by typing the following command on the command line:

```
$XPCROOT\bin> unconfigHotFS
```

Press <Enter>.

5. If HotFS was successfully uninstalled the following screen displays:

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:

```
C:\commerceone\XPC\bin>unconfigHotfs
C:\commerceone\XPC\bin>java -Dxpc.root="C:\commerceone\XPC" unconfigHotFS
Start WinNT uninstillation of HotFS .....
==> Finish unconfiguring file: C:\commerceone\XPC\etc\classpath\default
==> Finish unconfiguring file: C:\commerceone\XPC\runtime\servers\defaultserver
\config\startup\service-start.prop
UnConfiguration of HotFS service was Successful
C:\commerceone\XPC\bin>_
```

Figure 6-2 HotFS Uninstalled

Chapter 7

Running the HotFS Client and Service

In This Chapter

This chapter describes how to run both the HotFS Client and Service.

Running the HotFS Client

The Trading Partner must start the client either manually or with a script.

Starting the Client

To run the client use the client script HotFSClient located in the bin directory in the XCC installation. Use the Cygwin command line tool to run this script. Change `c:/commerceone/xcc` to the directory containing your XCC installation.

To run the HotFSClient script in Cygnus, type:

```
cd c:/commerceone/xcc/bin
sh ./HotFSClient
```

This command starts up the client with the default setting and arguments.

HotFS Client Options

The client script provides the default settings to run the client. However, if a different set of configurations, or a different schema path is desired, the HotFS Client provides two different options that you can specify on the command line:

Schema	Option
<p><code>-path <schemapath></code></p>	<p>This option provides the schema path to use when attempting to locate the HotFS schemas. The argument may contain several paths, each path separated by a semicolon ";".</p> <p>This option is specified with a default value in the script, and need not be specified unless the schemas are located elsewhere, additional schemas have been added, or customizations to the setup have been performed. The schemas that HotFS needs to use should after a successful installation of HotFS, and a default installation of XCC, be located in the directory <code>/commerceone/xcc/schema</code>.</p> <p>The argument to this option is the path to the root of the SOX schemas, not the path to the physical location of the schemas.</p> <p>Example: If we want to specify the location of the schemas to be in a different directory <code>/temp/schemas/myschemas</code>, we would run the client as follows:</p> <pre>sh ./HotFSClient -path /temp/schemas/myschemas</pre>
<p><code>-config <configfile></code></p>	<p>This option specifies the configuration file the client should use. If this option is not specified, the default behavior is for the client to look for the file "config.xml" in the current directory. This means the directory the user is currently located in.</p> <p>The argument to this option is the path of the file to use as the client configuration file.</p> <p>Example: If we want the client to use a file named <code>myconfig.cfg</code> in the directory <code>/temp</code>, we would run the client as follows:</p> <pre>sh ./HotFSClient -config /temp/myconfig.cfg</pre>

HotFS Client Troubleshooting

The following information describes some of the more common problems you may encounter when running the HotFS Client:

Errors Reported to the Screen

When the client encounters an error that can not be solved, such as incorrect configurations, an error will be output to the screen and the client will exit. Typically this will happen at the time the client is started.

The following table lists some common errors and possible solutions:

Error Message	Reason for Problem	Solution
Error reading document <document name> Configuration file could not be loaded. Fatal error occurred: Could not find file <document path>.	The specified configuration file could not be located or read.	Make sure the specified configuration file exists, and that the correct path has been specified. If no configuration file has been specified with the config option, make sure there is a configuration file named config.xml in the directory the client is run from.
Error parsing document config.xml Configuration file was not valid.	The specified configuration file was not valid.	Make sure the configuration file is a valid instance of the HotFSConfigurations schema. Use the validation error that follows the error message to correct the schema.
Configuration file did not contain HotFSConfiguration element. Fatal error occurred: <class name>	The configuration file is valid SOX but does not contain an instance of HotFSConfigurations.	Make sure the configuration file contains an instance of the HotFSConfigurations schema. Make sure the correct path for the configuration file has been provided

Error Message	Reason for Problem	Solution
<p>Error while parsing options Fatal error occurred: Invalid option <option name></p>	<p>An invalid option was provided.</p>	<p>Make sure the only options provided are -path and/or -config. No other options are supported. Make sure the options are spelled correctly.</p>
<p>Error while parsing options Fatal error occurred: No argument for the configuration file was provided</p>	<p>No argument was provided for the -config option.</p>	<p>The -config option must be used with an argument specifying the name of a valid configuration file.</p>
<p>Error while parsing options Fatal error occurred: No argument for the schema path was provided</p>	<p>No argument was provided for the -path option.</p>	<p>The -path option must be used with an argument specifying the schema path. If the -path option is not used, a default value will be provided for that option. This value should be sufficient, unless customizations to the setup have been performed.</p>
<p>Fatal error occurred: null</p>	<p>The configuration file did not start with a soxtype element</p>	<p>All valid XML instances of SOX schemas must start with a soxtype element. Make sure the document contains a valid soxtype element at the start of the document. For the client configuration file this must be: <pre><?soxtype urn:x-commerceone:document:com:commerceone:hotfs:schema:HotFSConfigurations.sox\$1.0?></pre> <p>Refer to the sample client configuration file in the configuration section for an example.</p> </p>

Errors Reported in an Error File

When the client encounters an error when executing a task, it will output an error file in the directory specified in the HotFS Client configurations. The default directory is hotfs/client/errors. Each error file is a valid instance of the schema HotFSError (Appendix F). Every error document has an occurrence code specified in the OccurredWhile element, enumerating the possible points where processing can encounter a problem. The error document also contains a detailed error message.

The following table lists some common errors and possible solutions:

OccurrenceCode	Error Message	Reason for Problem	Solution
Reading TDF	<ERROR creator="nul l">Could not find file: <file name>, <FATAL	The schema URI specified was incorrect.	Make sure the namespace specified in the soxtype element is correct.
ProcessingTDF	Stated content type was not SOX. Currently only SOX content is supported for this application.	The stated content type in the TDF was not SOX.	Only SOX content is supported at the current time. Only use valid instances of SOX schemas.
ProcessingTDF	No Local Name was specified for the document.	The LocalName element for the RootDocument was empty.	All SOX documents must have a namespace and local name specified. It is not valid to leave these fields blank.
ProcessingTDF	No Local Name was specified for attachment number 2	The LocalName element for the attachment was empty.	All SOX documents must have a namespace and local name specified. It is not valid to leave these fields blank.

OccurrenceCode	Error Message	Reason for Problem	Solution
ReadingDocument	<file name>	The specified document could not be found.	Make sure the specified document is in the stated location. Check the path of the specified document for corrections.
SendingDocument	Unable to contact target server <server name> after 3 tries.	The client did not get a response from the destination server specified in the client configurations.	Make sure the receiving server exists and is running.
SendingDocument	The host name [<server name>] couldn't be resolved. Details: "phony"; The XML processing failed with the following error: <message>	The client was unable to resolve the destination server address specified in the client configurations.	Make sure the server address is correct.
SendingDocument	The XML processing failed with the following error: <message>	The XML was not valid when received by the service.	Make sure the XML document that is transmitted is valid according to the schema it conforms to.
SendingDocument	Lost and found got sync request, sending back error	The server had no service subscribing to the type of the sent synchronous document.	Make sure the MarketSite you are sending to has a service subscribing to and processing the type of document you want to send.

OccurrenceCode	Error Message	Reason for Problem	Solution
SendingDocument	Send envelope error, code: 404	The directory specified in the server address element is incorrect.	<p>Make sure the correct server directory is specified.</p> <p>Example:</p> <p><code>https://www.marketsite.net:4433/xcc</code></p>
SendingDocument	Request timed out	The request was synchronous, and the recipient did not reply within the given time-out period.	You may increase the time-out period in the HotFS client configuration file, to ensure that synchronous replies are received.
SendingDocument	Caught unexpected exception! Message: null	The most likely reason is that the HotFSConfigurations file or a TDF file did not have a soctype element at the start of the instance.	Any instance of a SOX schema has to have a soctype element at the top of the document, specifying which schema it is an instance of.

Running the HotFS Service

The HotFS service is a service in the XCC server, and is started automatically when the XCC server is started. It can not be run independently of the XCC server.

Making Sure The HotFS Service has Started

The HotFS service starts up automatically, when the XCC server is started. For that reason it is important that the XCC server was restarted after HotFS was installed. This can be done by restarting the server manually, or by re-booting the machine.

To ensure that the HotFS Service started successfully, locate the XCC server system startup log in the /runtime/servers/defaultserver/logs directory. It will be named systemStartupLog followed by the date and time the server was started. The last modified system start-up log should be chosen. Open the file in a text viewing application.

The log file should contain an event towards the end of the log with the following text message:

```
Service DocumentService.HotFSService.HotFSService.1_0
started.
```

If the log file does not contain this message, the HotFS service did not start properly. See the HotFS troubleshooting section for more information.

HotFS Service Initialization Properties

The HotFS Service properties contain the default properties sufficient for running the service successfully. This file was set at installation, and does not need to be altered. However, if a different set of properties is desired, the properties file may be altered. If the properties are altered, the service must be restarted. This can be done manually, or simply by re-booting the machine.

The properties file, default.prop is located in the:

runtime/servers/defaultserver/config/service/
DocumentService.HotFSService.HotFSService.1_0

directory in the XCC installation. The default.prop file contains two properties conceptually identical to the client options:

Properties	Description
-path <schemapath>	<p>This property provides the schema path to use when attempting to locate the HotFS schemas. The argument may contain several paths, each path separated by a semicolon ";". This option is specified with a default value and need not be specified unless the schemas are located elsewhere, additional schemas have been added, or customizations to the setup have been performed. The schemas that HotFS needs to use should after a successful installation of HotFS, and a default installation of XCC, be located under the directory /commerceone/xcc/schema.</p> <p>The argument to this option is the path to the root of the SOX schemas, not the path to the physical location of the schemas. Example: If we want to specify the location of the schemas to be in a different directory /temp/schemas/myschemas, we would specify the property as follows:</p> <pre>schemapath=/temp/schemas/myschemas</pre>
-config <configfile>	<p>This option specifies the configuration file the service should use.</p> <p>This option is specified with a default value and need not be specified unless the configuration file is located elsewhere, or an alternate configuration file should be used. The default configuration file that the HotFS Service needs to use should after a successful installation of HotFS be located under the directory runtime/hotFS directory in the XCC installation. The argument to this option is the path of the file to use as the service configuration file.</p> <p>Example: If we want the service to use a file named myconfig.cfg in the directory /temp, we would specify the configfile property as follows:</p> <pre>configfile=/temp/myconfig.cfg</pre>

HotFS Service Troubleshooting

The following information describes some of the more common problems you may encounter when you are running HotFS Service:

Errors Reported to the System Startup Log

When the service encounters an error that can not be solved at system start-up time, such as incorrect configurations, and error will be output to the system start-up log, and the service will fail to start. The system start-up log is located in the /runtime/servers/defaultserver/logs directory. It will be named systemStartupLog followed by the date and time the server was started. The last modified system start-up log should be chosen. Open the file in a text viewing application.

Once the problem is fixed, the server needs to be restarted in order for the service to restart properly.

The following table lists some common error events, and possible solutions:

Event Message	Reason for Problem	Solution
Initialization of service HotFSService failed.	The service could not be initialized successfully by the service	This is the server's notification that the service could not be initialized properly, not the actual description of the failure. Locate the detailed initialization failure event output by the HotFS Service located before this event to determine the cause of the failure.
HotFSService could not initialize because of the following error: TIME= March 2, 2000 1:15:24 PM GMT-08:00 MILLIS= 952031724761 KEY= XDK_SIV_14 TEXT= file:///C:/commerceone/xcc/runtime/hotfs/hotfsServiceconfig.xml:3: 19: <Error Message>	The specified configuration file was not valid.	Make sure the configuration file is a valid instance of the HotFSServiceConfig schema (Appendix B)

Event Message	Reason for Problem	Solution
<p>HotFSService could not initialize because of the following error: Could not find file <file name>Make sure the specified configuration file exists, and that the correct path has been specified in the service property file.</p>	<p>The specified configuration file could not be located or read.</p>	<p>Make sure the specified configuration file exists, and that the correct path has been specified in the service property file.</p>
<p>HotFSService could not initialize because of the following error: The schema path was not specified in the HotFSService default.prop file. The schema path must be specified as &apos;schemapath=[path]&apos;.</p>	<p>The service property schemapath did not exist, or did not have a value in the service properties file.</p>	<p>Make sure the schema path property is defined as schemapath=<schema path> in the properties file, where schema path is the correct schema path to locate the HotFS schemas. The properties file is named default.prop and can be located in the runtime/servers/defaultserver/config/service directory in the XCC installation.</p>
<p>HotFSService could not initialize because of the following error: The configuration file C:/commerceone/xcc/runtime/hotfs/hotfsServiceconfig.xml specified in the HotFSService default.prop file did not contain an instance of HotFSServiceConfig</p>	<p>The service configuration file did not contain an instance of the HotFSServiceConfig schema.</p>	<p>Make sure the specified file is a valid instance of the HotFSServiceConfig schema.</p>

Event Message	Reason for Problem	Solution
HotFSService could not initialize because of the following error: <directory name> did not have a value!	A directory element in the HotFS Service configuration file did not contain a value.	All directory elements must contain a valid value.
HotFSService could not initialize because of the following error: NULL	The configuration file did not start with a soctype element.	All valid XML instances of SOX schemas must start with a soctype element. Make sure the document contains a valid soctype element at the start of the document. For the client configuration file this must be: <code><?soctype urn:x-commerceone:document:com:commerceone:hotfs:schema:HotFSServiceConfig.sox\$1.0?></code> Refer to the sample service configuration file in the configuration section for an example.

Errors Reported in Error File

When the service encounters an error when executing a task, it will output an error file in the directory specified in the HotFS Client configurations. The default directory is hotfs/service/errors.

Each error file is a valid instance of the schema HotFSError (Appendix F). Every error document has an Occurrence code specified in the OccurredWhile element, enumerating the possible points where processing can encounter a problem. The error document also contains a detailed error message.

The following table lists some common error events, and possible solutions:

Occurrence Code	Error Message	Reason for Problem	Solution
ReadingRDF	><ERROR creator="n ull">&C could not find file: <file name>, <FATAL	The schema URI specified was incorrect.	Make sure the namespace specified in the soxtype element of the RDF is correct.
ProcessingRDF	Stated content type was not SOX. Currently only SOX content is supported for this application.	The stated content type in the RDF was not SOX.	Only SOX content is supported at the current time. Only use valid instances of SOX schemas.
ProcessingRDF	No Local Name was specified in the RDF Document.	The LocalName element for the RootDocument was empty.	All SOX documents must have a namespace and local name specified. It is not valid to leave these fields blank.
ProcessingRDF	No Local Name was specified for attachment 2.	The LocalName element for the attachment was empty.	All SOX documents must have a namespace and local name specified. It is not valid to leave these fields blank.
Reading Document	<file name>	The specified document could not be found.	Make sure the specified document is in the stated location. Check the path of the specified document for correctness.

Occurrence Code	Error Message	Reason for Problem	Solution
SendingDocument	The XML processing failed with the following error: <message>	The XML was not valid when received.	Make sure the XML document that is transmitted is valid according to the schema it conforms to.
ReadingEnvelope	File \\hotfs\service\envelopes\1051be6c-77c9-1000-9ff7-0a0000a10001.e nv does not exist	The envelope file did not exist	The reply must be replying to a previous incoming transmission. Make sure the TaskID of the incoming transmission matches that of the reply.

Other Problems

Other errors may occur, that will not result in an error to the event log or an error file. These problems are covered in this section.

Problem	Reason for Problem	Solution
Incorrect document type returned to the HotFS Service. For example, a PurchaseOrder was expected, but a PurchaseOrderResponse arrived.	The supplier has not been set up properly as an integrated supplier in MarketSite.	This can only happen in a combined buyer/supplier setup scenario. Notify the MarketSite administrator of the problem.
Request did not arrive, but buyer got a PurchaseOrderResponse.	The supplier has not been set up properly as an integrated supplier in MarketSite.	Notify the MarketSite administrator of the problem.
Request did not arrive to service.	The supplier setup was not successful. Documents are not being forwarded appropriately to the HotFS service.	Notify the MarketSite administrator of the problem.

Chapter 8

Changing the HotFS Configurations

In This Chapter

The HotFS Client and HotFS Service each have their own XML configuration file. These files contain the configuration settings for each application. These files were configured when HotFS was installed, and normally do not need to be altered. However, if configurations need to be changed or added, these files should be modified.

This chapter explains how to modify HotFS Client and Service configurations. It also contains descriptions of basic transmission configurations, transmission settings, and document default settings.

Quick Reference for Changing HotFS Configurations

The configuration files are described in the following sections in detail, describing each element in the respective schemas. For configuration convenience, the most basic configuration changes and the elements that must be changed are described in this section. For more in-depth knowledge of the schemas and their elements, refer to the sections below.

Client Configuration Quick Reference

The default client configuration file is named `config.xml` and is located in the `bin` directory in the XCC installation.

Note You must restart the client for any changes to take effect.

HotFS Client Configuration Quick Reference

Change Needed	How to Configure
Change the MarketSite server to which the client transmits	In the client configuration file, change the value of the ServerAddress element to the correct MarketSite server. Note that the address must start with https:// and contain the port and directory.
Change the root directory for the client	In the client configuration file, change the value of the RootDirectory element to the desired directory path. Note that if the path is absolute, it must start with a drive letter or path separator.
Change any directory for the client	In the client configuration file, change the value of the desired directory element to the desired directory path. It is recommended that the path be relative. Note that if the path is absolute, it must start with a drive letter or path separator.
Change the client from polling continuously to polling only once	In the client configuration file, change the PollContinuously element to a PollOnce element. Note that the PollOnce element does not contain an Interval attribute.
Change the client from polling only once to polling continuously	In the client configuration file, change the PollOnce element to a PollContinuously element. Note that the PollContinuously element must contain an Interval attribute. This interval specifies the interval between pollings.
Change the MPID of the client	In the client configuration file, change the value of the SenderID element to contain the desired MPID.
Change the user name of the client	In the client configuration file, change the value of the UserName element to contain the desired user name.
Change the password of the client	In the client configuration file, change the value of the encrypted password to the new one generated by the encryptpassword tool. Refer to <i>Appendix I, encryptpassword Tool</i> for information on how to use the tool
Add default transmission settings for a document type	In the client configuration file, add a DocumentTypeDefault element in the appropriate location.

Change Needed	How to Configure
Change the default transmission modes of a document type.	In the client configuration file, modify the appropriate existing DocumentTypeDefault element.
Add proxy server support	In the client configuration file, add a ProxyServerSettings element in the appropriate location.

Service Configuration Quick Reference

The default service configuration file is named hotfsServiceconfig.xml and is located in the runtime/hotFS directory in the XCC installation.

Note You must restart the XCC for any changes to take effect. You can do this by manually by stopping and starting the server, or by re-booting the machine.

HotFSService Configuration Quick Reference

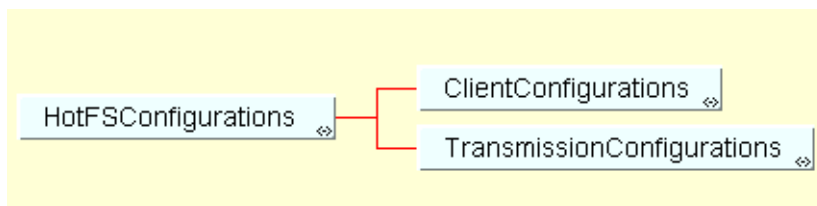
Change Needed	How to Configure
Change the root directory for the service	In the service configuration file, change the value of the RootDirectory element to the desired directory path. Note that if the path is absolute, it must start with a drive letter or path separator.
Change any directory for the service	In the service configuration file, change the value of the desired directory element to the desired directory path. It is recommended that the path be relative. Note that if the path is absolute, it must start with a drive letter or path separator.
Change the service polling interval	In the service configuration file, change the value of the SleepTime element.
Make the service subscribe to a specific document type	In the service configuration file, add a AcceptedDocument element in the appropriate location.
Make the service stop subscribing to a specific document type	In the service configuration file, remove the corresponding AcceptedDocument element.

Changing the HotFS Client Configuration

The client uses a configuration file in XML format. Assuming that the information provided when installing the XCC and HotFS components was correct, the file has already been set, and no additional configuration is necessary. You can alter the file for configuration changes.

Note The client must be restarted for any changes to take effect.

The XML configuration file must conform to the schema HotFSConfigurations listed in Appendix A in this document. The configuration file is used to set the basic client configurations, as well as configurations and default settings for a number of transmission variables.

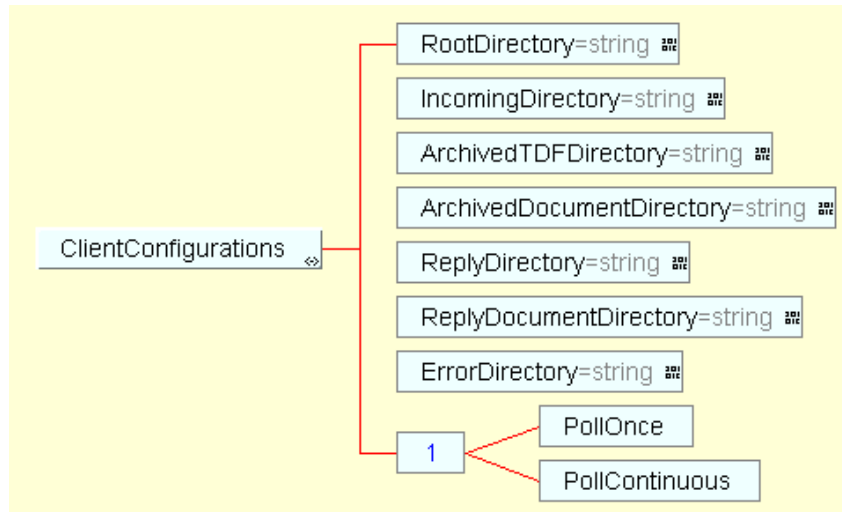


A default configuration file created at installation resides in the bin directory, and is named config.xml. This is the file that will be used by default, if no other file is specified with the client options. You can alter this file if a configuration change is desired, or you can create a new configuration file and specify it in the config option.

Client Configurations Element

The client configurations are set in the ClientConfigurations element in the instance of HotFSConfigurations. ClientConfigurations sets a number of parameters the client needs in order to function.

Explanation of ClientConfigurations



RootDirectory This is the directory that will serve as the root for all subsequent directories and files with relative paths. It will contain the subdirectories for all the files that will be used in, or result from various client transmissions.

This element is required.

Example: If we want the directory to be named hotfs/client, we specify the RootDirectory element:

```
<RootDirectory>/hotfs/client</RootDirectory>
```

IncomingDirectory This is the directory where the client will poll for TDF files that define the client tasks. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "incoming" and be relative to the root directory, it should be specified:

```
<IncomingDirectory>incoming</IncomingDirectory>
```


ArchivedTDFDirectory This is the directory where the client will place the processed TDF files. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "archived/tdf" and be relative to the root directory, it should be specified:

```
<ArchivedTDFDirectory>archived/tdf</
ArchivedTDFDirectory>
```

ArchivedDocument Directory This is the directory where the client will place the documents that were sent, if the transmission completed successfully. If the transmission did not complete successfully, the files will remain where they were originally placed. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "archived/documents" and be relative to the root directory, it should be specified:

```
<ArchivedDocumentDirectory>archived/documents</
ArchivedDocumentDirectory>
```

ReplyDirectory This is the directory where the client will place DDF files describing replies. Replies will only be returned if the transmission mode was synchronous. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "replies" and be relative to the root directory, it should be specified:

```
<ReplyDirectory>replies</ReplyDirectory>
```

ReplyDocument Directory

This is the directory where the client will place the reply documents that were contained in a reply. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "reply/documents" and be relative to the root directory, it should be specified:

```
<ReplyDocumentDirectory>reply/documents</
ReplyDocumentDirectory>
```

ErrorDirectory

This is the directory where the client will place error documents that detail any errors that were encountered during the attempt to transmit the document. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "errors" and be relative to the root directory, it should be specified:

```
<ErrorDirectory>errors</ErrorDirectory>
```

PollOnce

The client configurations can either chose this or the PollContinuous element, but not both. If this element is chosen, the client will poll the incoming directory only once, and then exit. Any TDF files that were found will be processed. This element has no content.

Example: If we want the client to poll only once, we should choose:

```
<PollOnce/>
```

PollContinuous

The client configurations can either chose this or the PollOnce element, but not both. If this element is chosen, the client will continuously poll the incoming directory, and will not exit. As long as a poll results in new files to process, the client will keep polling until no new files are found. It will then sleep for a period of time. The element has an attribute named Interval, that specifies the length of the period of sleep between polling. The interval is specified in milliseconds.

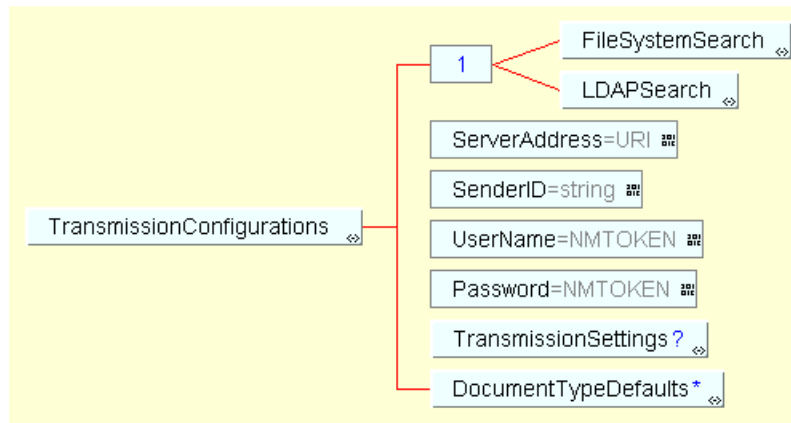
Example: If we want to specify that the client should run continuously with an interval of 1 second, we should choose PollContinuous and specify it to be:

```
<PollContinuous Interval="1000" />
```

Transmission Configurations Element

The transmission configurations are set in the Transmission Configurations element in the instance of HotFSConfigurations. TransmissionConfigurations sets a number of mandatory parameters the client needs in order to be able to transmit documents to a server. In addition, a number of default settings can be specified. These will be the settings the client will use if the individual TDF file omits any optional content.

Explanation of Basic Transmission Configuration



FileSystemSearch

The client configurations can either choose this or the LDAPSearch element, but not both. If this element is chosen, the client will attempt to locate schemas on the file system. The element contains one other element, SchemaPath. This path will be used to locate the schemas, and should point to the root of the schemas. Currently, the only schema that needs to be located, is the Error Schema. This is the recommended option. **This element is required.**

Example: If we want to be able to locate the Error Schema on the file system in an installation of XCC installed in the default location, the FileSystemSearch element should be specified:

```
<FileSystemSearch>
  <SchemaPath>/commerceone/xcc/schema</
  SchemaPath>
</FileSystemSearch>
```

LDAPSearch

The client configurations can either chose this or the FileSystemSearch element, but not both. If this element is chosen, the client will attempt to locate schemas in an LDAP server. The element contains three other elements, UserName, Password and SchemaRoot. These will be used to locate the schema in the LDAP server. Since currently, the only schema that needs to be located, is the Error Schema, it is recommended that the FileSystemSearch element is used.

This element is required.

Example: If we want to be able to locate the Error Schema in an LDAP server, the LDAPSearch element could be specified:

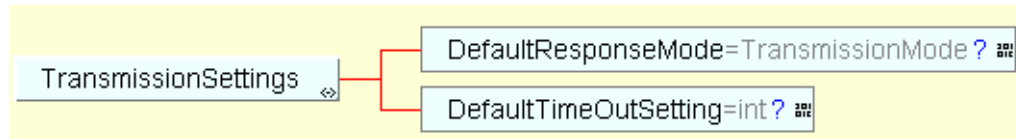
```
<LDAPSearch>
  <UserName>myUserName</UserName>
  <Password>mAJMjy/gt22FOFizplgZuA==
  </Password>
  <SchemaRoot>ldap://ldap.
  myldapserver.com:389/
  ou=SchemaRoot,o=commerceone.com
  </SchemaRoot>
</LDAPSearch>
```

Note You need to use the encryptpassword tool to generate an encrypted password from a plain text password. Refer to Appendix I for information on using the encryptpassword tool.

ServerAddress	<p>This element specifies the address to the MarketSite server the client should connect to. The address should include any ports and directories that are required to reach the MarketSite XCC server. Since the client should always connect securely, the address should start with https. This element is required.</p> <p>Example: If we want to connect to a MarketSite server, the ServerAddress element could be specified:</p> <pre><ServerAddress>https://www.marketsite.net:4433/xcc</ServerAddress></pre>
SenderID	<p>This element specifies the Market Participant ID (MPID) of the sending trading partner. An MPID is assigned to any participant in a MarketSite. This element is required.</p> <p>Example: If the MPID is 8ccb7126-77b1-1000-98c9-ac1408350001, the SenderID element should be specified:</p> <pre><SenderID>8ccb7126-77b1-1000-98c9-ac1408350001</SenderID></pre>
UserName	<p>This element specifies the user name of the sender specified in SenderID. This user name is used to authenticate the sending trading partner in the MarketSite the client is connecting to. This element is required.</p> <p>Example: If the user name for the trading partner is "user", the UserName element should be specified:</p> <pre><UserName>user</UserName></pre>
Password	<p>This element specifies the encrypted password corresponding to the user name specified in UserName. This password is used to authenticate the sending trading partner in the MarketSite the client is connecting to. This element is required.</p> <p>Refer to Appendix I for information on the encryptpassword tool.</p> <p>Example: If the password for the trading partner is "secret", the Password element should be specified with encrypted value "Y5qe7TmaNa4=" (refer to Appendix I for information on how to obtain an encrypted password from an unencrypted one):</p> <pre><Password>Y5qe7TmaNa4=</Password></pre>

Explanation of Transmission Settings

The client configuration file can define a set of default transmission settings in the TransmissionSettings element, which is optional.



DefaultResponseMode

This element defines a default response mode for any tasks that do not define the response mode. **This element is optional.**

Possible values are:

sync - This specifies synchronous transmission. The client will transmit the document, and then wait for a reply. The receiving service will be expected to post a reply.

oneway - This specifies asynchronous transmission. The client will transmit the document, but not wait for a reply. The receiving service may not post a reply to the transmission.

peer-peer - This specifies asynchronous transmission. The client will transmit the document, but not wait for a reply. The receiving service is expected to post a reply, but not immediately. The client will not receive such replies, as it is not capable of handling such replies. If a peer-peer reply is desired, the HotFS Service should be installed in addition to the client.

Example: If we want to specify the default response mode to be synchronous, the DefaultResponseMode element should be specified:

```
<DefaultResponseMode>sync</DefaultResponseMode>
```

DefaultTimeOutSetting

This element defines a default time out setting for any synchronous tasks that do not define a time out setting. The setting is specified in seconds. If the value "-1" is provided the time out setting is infinite. It is not recommended to use the infinite time out setting as it could potentially result in blocking the client. This setting has no effect for oneway or peer-peer transmissions.

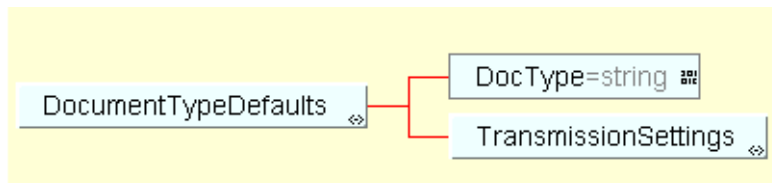
This element is optional.

Example: If we want to define the time out setting to be 10 seconds, the DefaultTimeOutSetting element should be specified:

```
<DefaultTimeOutSetting>10</DefaultTimeOutSetting>
```

Explanation of Document Type Default Settings

Since most document types have a pre-defined choreography that should be followed, the client configurations can also contain a set of default settings for specific document types. Any number of document types default settings can be defined in the DocumentTypeDefault element. **The element is optional and can be present multiple times.** Each element represents one specific document type.



DocType

This element contains the local name of the document type the default settings apply to. This will override any generic default settings in the TransmissionProperties, if the document type matches the document to transmit.

This element is optional.

Example: If we want the settings to apply to any PurchaseOrder, the DocType should be specified:

```
<DocumentType>PurchaseOrder</DocumentType>
```

DefaultResponseMode

This element defines a default response mode for any document that matches the document type defined in the DocType element, and that do not define the response mode in the task definition. **This element is optional.**

The possible values are:

sync - This specifies synchronous transmission. The client will transmit the document, and then wait for a reply. The receiving service will be expected to post a reply.

oneway - This specifies asynchronous transmission. The client will transmit the document, but not wait for a reply. The receiving service may not post a reply to the transmission.

peer-peer - This specifies asynchronous transmission. The client will transmit the document, but not wait for a reply. The receiving service is expected to post a reply, but not immediately. The client will not receive such replies, as it is not capable of handling such replies. If a peer-peer reply is desired, the HotFS Service should be installed in addition to the client.

Example: If we want to specify the default response mode for a specific document type to be peer to peer, the DefaultResponseMode element should be specified:

```
<DefaultResponseMode>peer-peer</
DefaultResponseMode>
```


DefaultTimeOutSetting

This element defines a default time out setting for any document with a synchronous response mode, that matches the document type defined in the DocType element and that do not define the time out setting in the task definition. The setting is specified in seconds. If the value "-1" is provided the time out setting is infinite. It is not recommended to use the infinite time out setting as it could potentially result in blocking the client. This setting has no effect for oneway or peer-peer transmissions.

This element is optional.

Example: If we want to define the time out setting to be 20 seconds, the DefaultTimeOutSetting element should be specified:

```
<DefaultTimeOutSetting>20</DefaultTimeOutSetting>
```

ProxyServerSettings

This element defines the physical addresses and port numbers of http and https proxy servers. **This element is optional.** To turn on the proxy server support, ProxySet must be set to true.

Example:

```
<ProxyServerSettings>
  <ProxySet>true</ProxySet>
  <ProxyHost>www-proxy.xyz.com</ProxyHost>
  <ProxyPort>80</ProxyPort>
  <HttpsProxyHost>www-proxy.xyz.com
</  HttpsProxyHost>
  <HttpsProxyPort>80</HttpsProxyPort>
</ProxyServerSettings>
```

Sample HotFSCConfigurations Instance

If we combine the individual examples above into a complete instance, it would look like this:

```
<?soxtype urn:x-
commerceone:document:com:commerceone:hotfs:schema:HotFSCConfigurations.sox$1.0?>
<HotFSCConfigurations>
  <ClientConfigurations>
    <RootDirectory>/hotfs/client</RootDirectory>
    <IncomingDirectory>incoming</IncomingDirectory>
    <ArchivedTDFDirectory>archived/tdf</ArchivedTDFDirectory>
    <ArchivedDocumentDirectory>archived/documents</
ArchivedDocumentDirectory>
    <ReplyDirectory>replies</ReplyDirectory>
    <ReplyDocumentDirectory>reply/documents</ReplyDocumentDirectory>
    <ErrorDirectory>errors</ErrorDirectory>
    <PollContinuous Interval="1000" />
  </ClientConfigurations>
  <TransmissionConfigurations>
    <FileSystemSearch>
      <SchemaPath>/commerceone/xcc/schema</SchemaPath>
    </FileSystemSearch>
    <ServerAddress>https://www.marketsite.net:4433/xcc</ServerAddress>
    <SenderID>8ccb7126-77b1-1000-98c9-ac1408350001</SenderID>
    <UserName>user</UserName>
    <Password>Y5qe7TmaNa4=</Password>
    <TransmissionSettings>
      <DefaultResponseMode>sync</DefaultResponseMode>
      <DefaultTimeOutSetting>10</DefaultTimeOutSetting>
    </TransmissionSettings>
    <DocumentTypeDefaults>
      <DocType>PurchaseOrder</DocType>
      <TransmissionSettings>
        <DefaultResponseMode>peer-peer</DefaultResponseMode>
      </TransmissionSettings>
```

```
</DocumentTypeDefaults>
<ProxyServerSettings>
  <ProxySet>true</ProxySet>
  <ProxyHost>www-proxy.xyz.com</ProxyHost>
  <ProxyPort>80</ProxyPort>
  <HttpsProxyHost>www-proxy.xyz.com</HttpsProxyHost>
  <HttpsProxyPort>80</HttpsProxyPort>
</ProxyServerSettings>
</TransmissionConfigurations>
</HotFSConfigurations>
```

Changing the HotFS Service Configurations

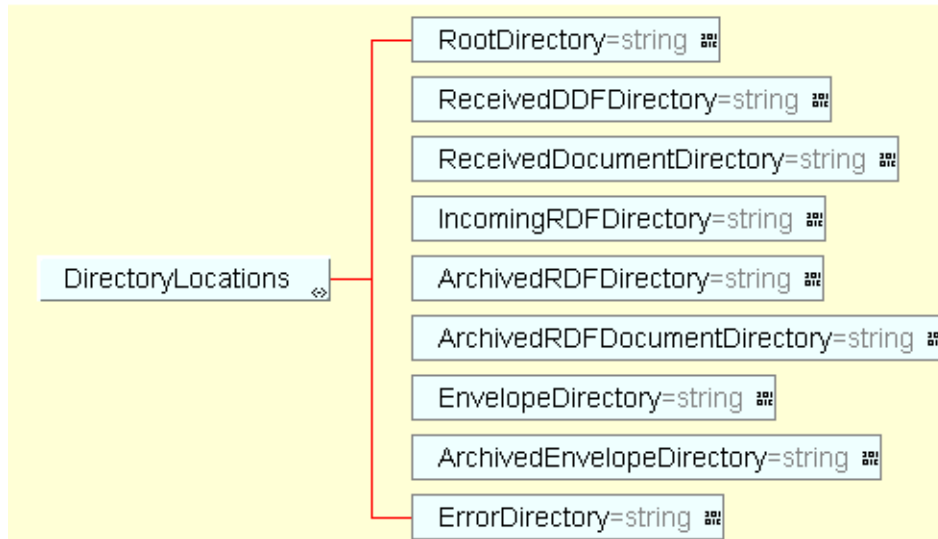
The service uses a configuration file in XML format. Assuming that the information provided when installing the XCC and HotFS components was correct, the file has already been set, and no additional configuration is necessary. The file may be altered for configuration changes however. Note that the XCC server must be restarted for any changes to take effect. This can be done by manually stopping and starting the server, or by re-booting the machine.

The XML configuration file must conform to the schema HotFSServiceConfig, which can be found in Appendix B in this document. The configuration file is used to set the basic service configurations.

A default configuration file created at installation resides in the runtime/hotFS directory in the XCC installation directory, and is named hotfsServiceconfig.xml. This is the file that will be used by default, if no other file is specified with the client options. This file may be altered if a configuration change is desired, or a new configuration file may be created and specified in the HotFS Service properties file.

DirectoryLocations Element

The service directory location configurations are set in the DirectoryLocations element in the instance of HotFSServiceConfig. DirectoryLocations sets the directories the service uses to communicate with the Trading Partner Application.



RootDirectory

This is the directory that will serve as the root for all subsequent directories and files with relative paths. It will contain the subdirectories for all the files that will be used in, or result from various client transmissions.

This element is required.

Example: If we want the directory to be named hotfs/service, we specify the RootDirectory element:

```
<RootDirectory>/hotfs/service</RootDirectory>
```

ReceivedDDFDirectory

This is the directory where the service will place DDF files describing documents from incoming transmissions. This may be a new incoming request, such as a PurchaseOrder, or replies to previous asynchronous requests sent to MarketSite by the client. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "received" and be relative to the root directory, it should be specified:

```
<ReceivedDDFDirectory>received</  
ReceivedDDFDirectory>
```

ReceivedDocument Directory

This is the directory where the service will place the documents from incoming transmissions. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "received/documents" and be relative to the root directory, it should be specified:

```
<ReceivedDocumentDirectory>received/documents  
</ReceivedDocumentDirectory>
```

IncomingRDFDirectory

This is the directory where the service will poll for RDF files that define the service reply tasks. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "incoming" and be relative to the root directory, it should be specified:

```
<IncomingRDFDirectory>incoming</  
IncomingRDFDirectory>
```

ArchivedRDFDirectory

This is the directory where the service will place the processed RDF files. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "archived/rdf" and be relative to the root directory, it should be specified:

```
<ArchivedRDFDirectory>archived/rdf</ArchivedRDFDirectory>
```

ArchivedRDFDocument Directory

This is the directory where the service will place the documents that were sent in a reply, if the transmission completed successfully. If the transmission did not complete successfully, the files will remain where they were originally placed. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "archived/documents" and be relative to the root directory, it should be specified:

```
<ArchivedRDFDocumentDirectory>archived/documents</ArchivedRDFDocumentDirectory>
```

EnvelopeDirectory

This is the directory where the client will place the envelopes from incoming transmissions. These envelopes are retained in those cases where the incoming transmission warrants a reply. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "envelopes " and be relative to the root directory, it should be specified:

```
<EnvelopeDirectory>envelopes</EnvelopeDirectory>
```

ArchivedEnvelopeDirectory

This is the directory where the service will place the envelope that were used in a reply, if the transmission completed successfully. If the transmission did not complete successfully, the envelope will remain where it was originally placed. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "archived/envelopes" and be relative to the root directory, it should be specified:

```
<ArchivedEnvelopeDirectory>archived/envelopes</ArchivedEnvelopeDirectory>
```

ErrorDirectory

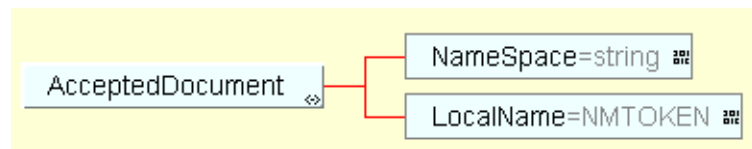
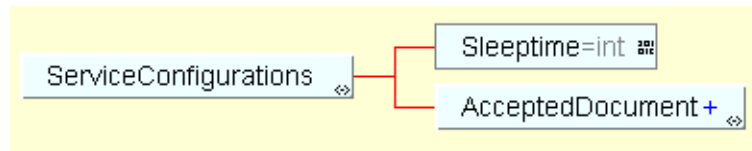
This is the directory where the service will place error documents that detail any errors that were encountered during the attempt to transmit the document. This directory should be relative to the root directory. However, if the directory starts with a path separator or a drive letter, the service will assume it is an absolute path.

This element is required.

Example: If we want the directory to be named "errors" and be relative to the root directory, it should be specified:

```
<ErrorDirectory>errors</ErrorDirectory>
```


Explanation of Service Configurations



SleepTime

The service configurations use this element to determine the length of the period of sleep between polling for RDF files. The interval is specified in milliseconds.

This element is optional.

Example: If we want to specify that the service should poll for RDF files in intervals of 1second, SleepTime is specified:

```
<SleepTime>1000</SleepTime>
```

AcceptedDocument

The service configurations can specify what document types the service should subscribe to. The document types are specified with a namespace and local name. The AcceptedDocument element may occur multiple times, but must be present at least once. It contains a Namespace and a LocalName element. The Namespace element defines the namespace of the schema the accepted document type is defined in, and the LocalName element specifies the local name of the document type.

This element is required and can be present multiple times.

Example: If we want to specify that the service should accept documents of type PurchaseOrder, we should specify an instance of AcceptedDocument as:

```
<AcceptedDocument>
  <Namespace> urn:x-
    commercione:documnet:com:commercione:
    CBL:CBL.sox$1 sox$1.0
  </Namespace><LocalName>PurchaseOrder
  </LocalName>
</AcceptedDocument>
```

Sample HotFSServiceConfig Instance

If we combine the individual examples above into a complete instance, it would look like this:

```
<?soxtype urn:x-
commerceone:document:com:commerceone:hotfs:schema:
HotFSServiceConfig.sox$1.0?>
<HotFSServiceConfig>
<DirectoryLocations>
<RootDirectory>/hotfs/service</RootDirectory>
<ReceivedDDFDDirectory>received/ddf</ReceivedDDFDDirectory>
<ReceivedDocumentDirectory>received/documents</
ReceivedDocumentDirectory>
<IncomingRDFDirectory>incoming</IncomingRDFDirectory>
<ArchivedRDFDirectory>archived/rdf</ArchivedRDFDirectory>
<ArchivedRDFDocumentDirectory>archived/document</
ArchivedRDFDocumentDirectory>
<EnvelopeDirectory>envelopes</EnvelopeDirectory>
<ArchivedEnvelopeDirectory>archived/envelopes</
ArchivedEnvelopeDirectory>
<ErrorDirectory>errors</ErrorDirectory>
</DirectoryLocations>
<ServiceConfigurations>
<SleepTime>1000</SleepTime>
<AcceptedDocument>
<Namespace>urn:x-
commerceone:document:com:commerceone:CBL:CBL.sox$1.0</
Namespace>
<LocalName>PurchaseOrder</LocalName>
</AcceptedDocument>
</ServiceConfigurations>
</HotFSServiceConfig>
```

Chapter 9

TDF Files

In This Chapter

This chapter describes the TDF Document and provides a sample TDF instance.

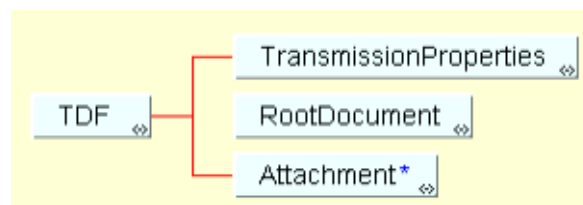
Overview

Each transmission is considered a separate task, and is described in the Task Description File (TDF). The TDF must conform to the TDF schema that can be found in Appendix C. The TDF is placed in the file system after the documents that are to be sent, and is the notification for the client that a new transmission task has arrived.

Each TDF describes a set of transmission specific parameters, that are specific to that task, and that the client can not derive from the various transmission settings in the client configuration file. In addition, the TDF can override the default setting specified for the client, or document type default settings.

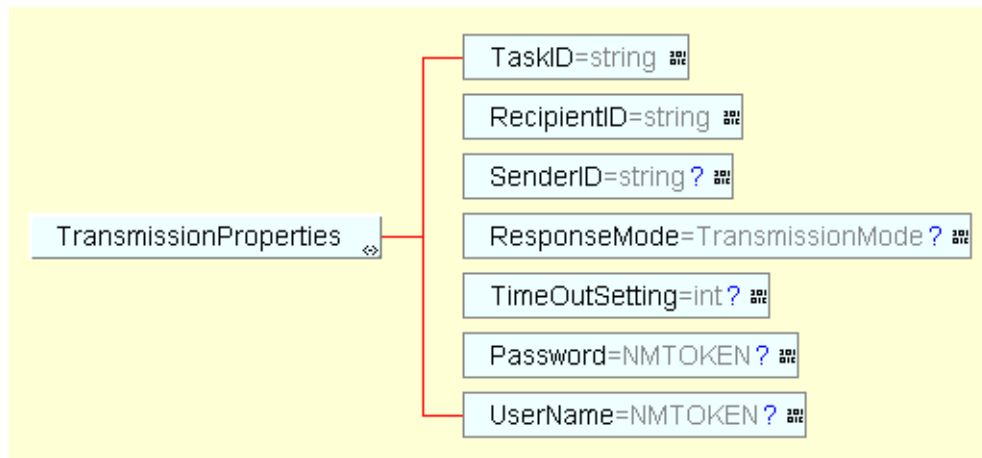
TDF Element

The TDF element contains 3 elements. TransmissionProperties, RootDocument and Attachment.



Explanation of Transmissions Properties

TransmissionProperties describes the transmission settings that can not be derived from the default settings in the client configuration file, but may also override some of those default settings if desired.



TaskID

This element contains the TaskID. This ID will be used to identify the task, and will be used to name any files resulting from this task. The TaskID must be globally unique in the receiving MarketSite. Note that the TaskID value should never start with any leading 0's, or these may be truncated in the server processing. This is important since all reply or error files resulting from the transmission are named with the TaskID, and any leading 0's may result in files with differing names.

This element is required.

Example: A valid TaskID instance could be:

```
<TaskID>1051be6c-77c9-1000-9ff7-0a0000a10001</TaskID>
```

RecipientID This element contains the MPID of the receiving trading partner.
This element is required.

Example: if the receiving trading partner's MPID is 8ccb7126-77b1-1000-98c9-ac1408350001, then the RecipientID element should be specified:

```
<RecipientID>8ccb7126-77b1-1000-98c9-  
ac1408350001</RecipientID>
```

SenderID This element specifies the MPID of the sending trading partner.
This element is optional, and should only be specified if the Sender ID for some reason is different than that specified in the client configuration. If several senders use the same client, then this should always be specified, to avoid potential identity problems.

This element is optional.

Example: If the sending trading partner's MPID is 8ccb7126-77b1-1000-98c9-ac1408350001, then the SenderID element should be specified:

```
<SenderID>8ccb7126-77b1-1000-98c9-ac1408350001</  
SenderID>
```

ResponseMode This element specifies the response mode of the transmitted document. This element is optional, and should only be specified if the response mode should be different from the default mode specified in the client configuration file.

This element is optional.

Generally, it is important that the response mode correlates to what the receiving service expects from specific document types. Therefore it is recommended that the client specify the normal document settings in the document type defaults in the client configuration file, and only use this element if the task for some reason deviates from the normal settings.

The possible values are:

sync - This specifies synchronous transmission. The client will transmit the document, and then wait for a reply. The receiving service will be expected to post a reply.

oneway - The client will transmit the document, but not wait for a reply. The receiving service may not post a reply to the transmission.

peer-peer - The client will transmit the document, but not wait for a reply. The receiving service is expected to post a reply, but not immediately. The client will not receive such replies, as it is not capable of handling such replies. If a peer-peer reply is desired, the HotFS Service should be installed in addition to the client.

Example: If we want to specify the response mode for a specific task to be synchronous, the ResponseMode element should be specified:

```
<ResponseMode>sync</ResponseMode>
```


TimeOutSetting This element defines a time out setting for the task that has a synchronous response mode. The setting is specified in seconds. If the value "-1" is provided the time out setting is infinite. It is not recommended to use the infinite time out setting as it could potentially result in blocking the client. This setting has no effect for oneway or peer-peer transmissions. This element is optional and should only be used to override the default settings in the client configuration file. This element should only be used to override the default setting, if the specific task for some reason has a different time out need than normal. **This element is optional.**

Example: If we want to define the time out setting to be 5 seconds, the TimeOutSetting element should be specified:

```
<TimeOutSetting>5</TimeOutSetting>
```

Password

This element is used to specify the encrypted password for the trading partner specified in SenderID in either the TDF or the client configuration file. This password is used to authenticate the sending trading partner in the MarketSite the client is connecting to.

This element is optional and should only be used to override the password specified in the client configuration file. This could be useful if a different SenderID is specified in the TDF, or if for some reason it is not desired to have the real password in the client configuration file.

This element is optional.

Example: If the password for the trading partner is "secret", the Password element should be specified with the encrypted value (refer to Appendix I on how to obtain an encrypted password from an unencrypted one):

```
<Password>Y5qe7TmaNa4=</Password>
```

UserName

This element is used to specify the user name for the trading partner corresponding to the password specified in Password. This user name is used to authenticate the sending trading partner in the MarketSite the client is connecting to.

This element is optional and should only be used to override the user name specified in the client configuration file. This could be useful if a different SenderID is specified in the TDF, or if for some reason it is not desired to have the real user name in the client configuration file.

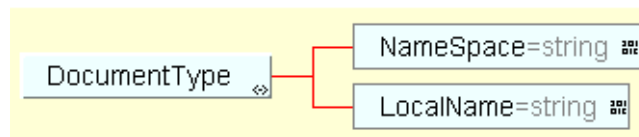
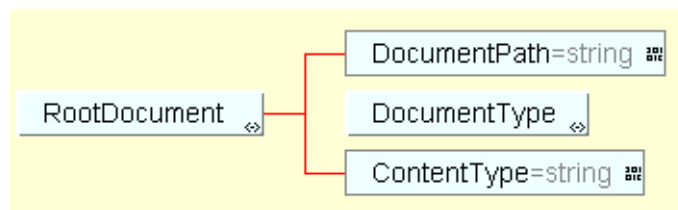
This element is optional.

Example: If the user name for the trading partner is "user", the UserName element should be specified:

```
<UserName>user</UserName>
```

Explanation of RootDocument

The RootDocument element is used to specify the document to send in the transmission.



DocumentPath

This element specifies the path of the file that will be sent as the root document. This path will be relative to the root directory specified in the client configuration file. However, if the path starts with a path separator or a drive letter, the client will assume it is an absolute path.

This element is required.

Example: If the document to be transmitted is named documents/8ccb7126-77b1-1000-98c9-ac1408350001PO123456.xml, and the documents directory is located in the root directory specified in the client configuration file, DocumentPath is specified:

```
<DocumentPath>documents/1051be6c-77c9-1000-9ff7-0a0000a10001.xml</DocumentPath>
```

DocumentType This element specifies the document type of the document specified in the DocumentPath element. It contains a Namespace and a LocalName element. The Namespace element defines the namespace of the schema the root element is defined in, and the LocalName element specifies the name of the root element of the instance.

This element is required.

Example: If the document is an instance of PurchaseOrder defined in the CBL namespace, the DocumentType element is specified:

```
<DocumentType>
  <Namespace>urn:x-
    commerceone:document:
    com:commerceone:CBL
    :CBL.sox$1.0</Namespace>
  <LocalName>PurchaseOrder</LocalName>
</DocumentType>
```

ContentType This element specifies the content type of the document. In this version, the HotFSClient only supports the content type SOX. This element is present for future support of other content than instances of SOX schemas.

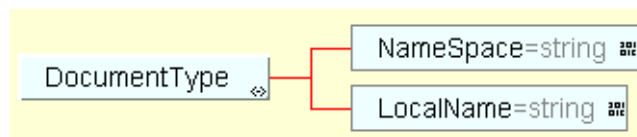
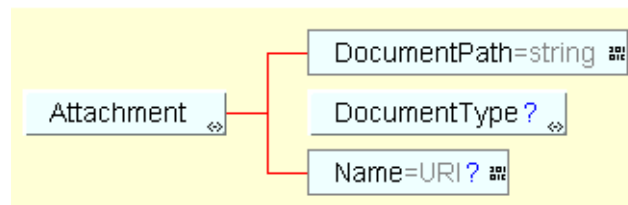
This element is required.

Example: If the content type is SOX, the ContentType element is specified:

```
<ContentType>SOX</ContentType>
```

Explanation of Attachment

Each transmission may have one or several attachments, if needed. Therefore the Attachment element may be present any number of times, or not at all, if no attachments are desired.



DocumentPath

This element specifies the path of the file that should be attached to the transmission envelope. This path will be relative to the root directory specified in the client configuration file. However, if the path starts with a path separator or a drive letter, the client will assume it is an absolute path. **This element is required.**

Example: If a document named attachments/8ccb7126-77b1-1000-98c9-ac1408350001PO123456.xml should be attached to the transmission, and the attachments directory is located in the root directory specified in the client configuration file, DocumentPath is specified:

```
<DocumentPath>attachments/1051be6c-77c9-1000-9ff7-0a0000a10001.xml</DocumentPath>
```

DocumentType This element is optional, and should only be specified when the attachment is an XML instance of a SOX schema. For all other attachment types the DocumentType element should be omitted. If the DocumentType is omitted, the client will map the document to a specific mime type. It is therefore important that the extension of the attachment is correct. See Appendix G for the current mapping between extensions and mime types. The attachment must have one of the mapped file extensions.

When used, this element specifies the document type of the XML instance of a SOX schema specified in the DocumentPath element. It contains a Namespace and a LocalName element. The Namespace element defines the namespace of the schema the attachment's root element is defined in, and the LocalName element specifies the name of the root element of the instance in the attachment.

This element is optional.

Example: If the attachment is an instance of PriceCheckResult defined in the CBL namespace, the DocumentType element is specified:

```
<DocumentType>
  <Namespace>urn:x-commerceone:document:
    com:commerceone:CBL
    :CBL.sox$1.0</Namespace>
  <LocalName>PriceCheckResult</LocalName>
</DocumentType>
```

Name This element specifies the name of the attachment, if the attachment is named.

This element is optional.

Example: If the name of the attachment is urn:myAttachment, the Name element is specified:

```
<Name>urn:myAttachment</Name>
```

Sample TDF Instance

If we combine the individual examples above into a complete instance, it would look like this:

```
<?soxtype urn:x-commerceone:document:com:commerceone:hotfs:schema:TDF.sox$1.0?>
<TDF>
  <TransmissionProperties>
    <TaskID>1051be6c-77c9-1000-9ff7-0a0000a10001</TaskID>
    <RecipientID>8ccb7126-77b1-1000-98c9-ac1408350001
    </ RecipientID>
    <SenderID>8ccb7126-77b1-1000-98c9-ac1408350001</SenderID>
    <ResponseMode>sync</ResponseMode>
    <TimeOutSetting>5</TimeOutSetting>
    <Password>Y5qe7TmaNa4=</Password>
    <UserName>user</UserName>
  </TransmissionProperties>
  <RootDocument>
    <DocumentPath>documents/1051be6c-77c9-1000-9ff7-0a0000a10001.
      xml</DocumentPath>
    <DocumentType>
      <Namespace>urn:x-commerceone:document:com:commerceone:
        CBL:CBL.sox$1.0</Namespace>
      <LocalName>PurchaseOrder</LocalName>
    </DocumentType>
    <ContentType>SOX</ContentType>
  </RootDocument>
  <Attachment>
    <DocumentPath>attachments1051be6c-77c9-1000-9ff7-
      0a0000a10001.xml</DocumentPath>
    <DocumentType>
      <Namespace>urn:x-commerceone:document:com:commerceone:
        CBL:CBL.sox$1.0</Namespace>
      <LocalName>PriceCheckResult</LocalName>
    </DocumentType>
    <Name>urn:myAttachment</Name>
  </Attachment>
</TDF>
```


Chapter 10

RDF Files

In This Chapter

This chapter describes the Reply Description Files (RDF).

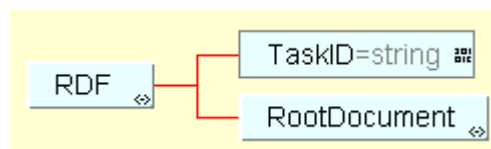
Overview

Every reply the service wants to send back in response to a previous incoming transmission must be described in a Reply Description File (RDF). The RDF must conform to the RDF schema that can be found in Appendix D. The RDF is placed in the file system after the reply document that is to be sent, and is the notification to the service that a new reply task has arrived.

Each RDF describes the TaskID for the reply, and the document to send as a reply. A reply may not contain any attachments. All transmission specific information will be picked up from the envelope of the original request.

RDF

The RDF element contains two elements. TaskID and RootDocument.



Explanation of TaskID

The TaskID element is used to specify the task ID of the reply transmission.

TaskID

This element contains the TaskID of the reply. This ID will be used to correlate the reply to the original request. Therefore it is vital that the TaskID of the reply matches the TaskID provided by the original request. This TaskID would have been defined in the DDF file of the incoming request.

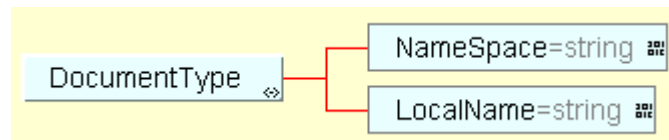
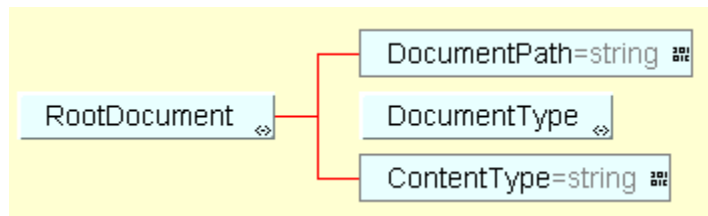
This element is required.

Example: If the TaskID of the incoming request was 1051be6c-77c9-1000-9ff7-0a0000a10001, the TaskID element must be:

```
<TaskID>1051be6c-77c9-1000-9ff7-0a0000a10001</TaskID>
```

Explanation of RootDocument

The RootDocument element is used to specify the document to send in the reply transmission.



DocumentPath This element specifies the path of the file that will be sent as the reply document. This path will be relative to the root directory specified in the service configuration file. However, if the path starts with a path separator or a drive letter, the client will assume it is an absolute path.
This element is required.

Example: If the document to be transmitted is named documents/8ccb7126-77b1-1000-98c9-ac1408350001PO123456.xml, and the documents directory is located in the root directory specified in the client configuration file, DocumentPath is specified:

```
<DocumentPath>documents/1051be6c-77c9-1000-9ff7-0a0000a10001.xml</DocumentPath>
```

DocumentType This element specifies the document type of the document specified in the DocumentPath element. It contains a Namespace and a LocalName element. The Namespace element defines the namespace of the schema the root element is defined in, and the LocalName element specifies the name of the root element of the instance.
This element is required.

Example: If the document is an instance of PurchaseOrderResponse defined in the CBL namespace, the DocumentType element is specified:

```
<DocumentType>
  <Namespace>urn:x-   commercene:document:
com:commercene:CBL
:CBL.sox$1.0</Namespace>
  <LocalName>PurchaseOrderResponse
</LocalName>
</DocumentType>
```

ContentType This element specifies the content type of the document. In this version, the HotFS Service only supports the content type SOX. This element is present for future support of other content than instances of SOX schemas.
This element is required.

Example: If the content type is SOX, the ContentType element is specified:

```
<ContentType>SOX</ContentType>
```

Sample RDF Instance

If we combine the individual examples above into a complete instance, it would look like this:

```
<?soxtype urn:x-commerceone:document:com:commerceone:hotfs:schema:RDF.sox$1.0?>
<RDF>
  <TaskID>1051be6c-77c9-1000-9ff7-0a0000a10001</TaskID>
  <RootDocument>
    <DocumentPath>documents/1051be6c-77c9-1000-9ff7-0a0000a10001.xml</DocumentPath>
    <DocumentType>
      <NameSpace>urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0</NameSpace>
      <LocalName>PurchaseOrderResponse</LocalName>
    </DocumentType>
    <ContentType>SOX</ContentType>
  </RootDocument>
</RDF>
```

Chapter 11

Result Files

In This Chapter

This chapter describes incoming files and error files.

Overview

Some transmissions will result in replies from MarketSite or errors from the transmission. These are placed as files in the file system, in directories specified in the client configurations. The Trading Partner may then poll for these files. The Trading Partner should always poll for error files, to ensure that no errors occurred during the transmission. If the Trading Partner wishes to get the replies from synchronous transmissions, it should also poll for replies.

It is important to note that only the synchronous response mode results in the client getting a reply document. A one way request will not produce a reply, and the client is unable to accept peer-peer replies. If peer-peer replies are desired, the HotFSService should also be installed in addition to the HotFS Client. The HotFS Service is capable of receiving peer-peer replies to documents the client has previously sent.

Incoming Files

The following information describes replies and incoming requests.

Replies

Each synchronous reply that the client receives, and each asynchronous reply that the service receives will result in exactly two files being placed on the file system. These documents will be the reply document, and a DDF describing the reply document. A reply will not have any attachments. The reply document received will be placed in the directory specified in the ReplyDocumentDirectory element in the client

configuration file. In addition, a Document Description File (DDF) will be written to the directory specified in the ReplyDirectory element in the client configuration file. The DDF is an instance of the DDF schema (Appendix E), and it describes the reply document. The DDF will always be written to the file system last, to ensure that the document has been properly written first. The Trading Partner Application should therefore poll for the DDF file.

The DDF File will be named <TaskID>.ddf.

The reply document will be named <TaskID>.xml.

Incoming Requests

Each new incoming request to the service will result in two or more files being placed on the file system. The incoming document received will be placed in the directory specified in the ReceivedDocumentDirectory element in the service configuration file. If the request warrants a reply, the request envelope will be stored for the future reply. If the envelope contained any attachments, they will be stored along with the document in the ReceivedDocumentDirectory. If the sender provided a file extension for the attachment, the attachment will be stored with the correct file extension. If not, the attachments will be stored with the extension "att".

In addition, a Document Description File (DDF) will be written to the directory specified in the ReceivedDDFDirectory element in the service configuration file. The DDF is an instance of the DDF schema (Appendix E), and it describes the reply document. The DDF will always be written to the file system last, to ensure that the document has been properly written first. The Trading Partner Application should therefore poll for the DDF file.

If any attachments were present in the envelope, their mime type will be described in the DDF file. The current list of mime types can be found in Appendix H. In addition to this list, an XML instance of a SOX schema will have the mime type "application/xml". This is to distinguish the XML instances of SOX schemas from regular XML documents.

The response mode of the incoming request will also be described in the DDF file. This is for the convenience of the trading partner application, to determine if a document warrants a reply, and also to be able to determine if the request was synchronous or peer-peer. Synchronous requests are more time sensitive than peer-peer requests, and this information facilitates preference for the synchronous requests.

The DDF File will be named <TaskID>.ddf.

The reply document will be named <TaskID>.xml.

Any attachments will be named <TaskID>att<N>.<Extension>, where N is the number of the attachment, (each attachment will be numbered, starting at 1, and incrementing by 1), and Extension is the extension of the attachment. If the correct extension was provided by the sender, it will be used. If not the extension will be "att".

Sample DDF Instance

```
<?soxtype urn:x-commerceone:document:com:commerceone:hotfs:schema:DDF:sox$1.0?>
<?import urn:x-commerceone:document:com:commerceone:hotfs:schema:DDF:sox$1.0?>
<DDF>
  <TransmissionInfo>
    <TaskID>1051be6c-77c9-1000-9ff7-0a0000a10001</TaskID>
    <RecipientID>536006fe-77ac-1000-9b16-ac14080d0001</RecipientID>
    <SenderID>f935078c-77b6-1000-9029-ac1408350001</SenderID>
    <ResponseMode>oneway</ResponseMode>
  </TransmissionInfo>
  <RootDocument>
    <DocumentPath>\hotfs\client\reply\documents\replydocs\1051be6c-77c9-
1000-9ff7-0a0000a10001.xml</DocumentPath>
    <DocumentType>
      <Namespace>urn:x-
commerceone:document:com:commerceone:CBL:CBL:sox$1.0</Namespace>
      <LocalName>PurchaseOrderResponse</LocalName>
    </DocumentType>
    <ContentType>SOX</ContentType>
  </RootDocument>
</DDF>
```

Error File

If an error is encountered at any point in the processing of the task, an error file will be written to the file system in the directory specified by the `ErrorDirectory` element in the client configuration file. The error is a valid instance of the `HotFSError` schema (Appendix F). The error document contains the `TaskID` of the task that produced the error, an error code signifying at what point in processing the error occurred, and an error message. The Trading Partner application can then use this document to determine what went wrong in the transmission.

The error file is named `<TaskID>.err`.

Sample HotFS Error Instance

```
<?soxtype urn:x-
commerceone:document:com:commerceone:hotfs:schema:HotFSError.sox$1.0?>
<?import urn:x-
commerceone:document:com:commerceone:hotfs:schema:HotFSError.sox$1.0?>
<HotFSError>
  <TaskID>1051be6c-77c9-1000-9ff7-0a0000a10001</TaskID>
  <OccurredWhile>ReadingTDF</OccurredWhile>
  <message>&lt;ERROR creator=&quot;Validation&quot;&gt;file:///hotfs/
client/archived/tdf/test.tdf:10:20: Specified NMTOKEN value
&amp;quot;My Name&amp;quot; is not a valid value for NMTOKEN.&lt;/
ERROR&gt;
, &lt;FATAL creator=&quot;Validation&quot;&gt;Cannot continue due to
previous errors.&lt;/FATAL&gt;,
&lt;STATUS creator=&quot;cxp&quot;&gt;Parser was stopped.&lt;/
STATUS&gt;,
  </message>
</HotFSError>
```


Appendix A

HotFS Configuration Schema

The following information is an example of a HotFS configuration schema.

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-
commerceone:document:com:commerceone:hotfs:schema:HotFSConfigurations.sox$1.0">

  <elementtype name="HotFSConfigurations">
    <explain>
      <h1>HotFSConfigurations</h1>
      <p>This element describes a set of configurations for the
      HotFS client. It contains basic configurations for the
      client, as well as default settings for any client
      transmissions.</p>

      <p><code>ClientConfigurations</code> describes the basic
      client configurations.</p>

      <p><code>TransmissionConfigurations</code> describes
      the default settings for any client transmissions.</p>
    </explain>

    <model>
      <sequence>
        <element type="ClientConfigurations"/>
        <element type="TransmissionConfigurations"/>
      </sequence>
    </model>
  </elementtype>
```

```
<elementtype name="ClientConfigurations">
  <explain>
    <h1>ClientConfigurations</h1>
    <p>This element describes the basic client configurations
    the client is to be initialized with.</p>

    <p><code>RootDirectory</code> specifies the root
    directory for the various file directories.</p>

    <p><code>IncomingDirectory</code> specifies the
    directory to poll for any incoming TDF files. This
    directory should be relative to the RootDirectory.
    If the path starts with a path separator, the path
    is assumed to be absolute on the current drive.</p>

    <p><code>ArchivedTDFDirectory</code> specifies the
    directory to place processed TDF files in. This
    directory should be relative to the RootDirectory.
    If the path starts with a path separator, the path
    is assumed to be absolute on the current drive.</p>

    <p><code>ArchivedDocumentDirectory</code> specifies
    the directory to place documents that were successfully
    transmitted in. This directory should be relative
    to the RootDirectory. If the path starts with a path
    separator, the path is assumed to be absolute on the
    current drive.</p>

    <p><code>ReplyDirectory</code> specifies the directory
    to place DDF files in. This directory should be relative
    to the RootDirectory. If the path starts with a path
    separator, the path is assumed to be absolute on the
    current drive.</p>

    <p><code>ReplyDocumentDirectory</code> specifies
    the directory to place reply documents in. This directory
    should be relative to the RootDirectory. If the path
    starts with a path separator, the path is assumed to
    be absolute on the current drive.</p>
```

```

    <p><code>ErrorDirectory</code> specifies the directory
    to place error documents in. This directory should be
    relative to the RootDirectory. If the path starts with a
    path separator, the path is assumed to be absolute on the
    current drive.</p>

    <p><code>PollOnce</code> being present signifies that
    the client should only poll the directory specified
    by IncomingDirectory once, and then exit.</p>

    <p><code>PollContinuous</code> being present
    signifies that the client should poll continuously
    in the directory specified by IncomingDirectory.</p>
  </explain>

  <model
    <sequence>
      <element name="RootDirectory" type="string"/>
      <element name="IncomingDirectory" type="string"/>
      <element name="ArchivedTDFDirectory" type="string"/>
      <element name="ArchivedDocumentDirectory" type="string"/>
      <element name="ReplyDirectory" type="string"/>
      <element name="ReplyDocumentDirectory" type="string"/>
      <element name="ErrorDirectory" type="string"/>
      <choice>
        <element type="PollOnce"/>
        <element type="PollContinuous"/>
      </choice>
    </sequence>
  </model>
</elementtype>

<elementtype name="TransmissionConfigurations">
  <explain>

    <p><code>FileSystemSearch</code> being chosen
    specifies that schemas should be located from the
    file system.</p>

    <p><code>LDAPSearch</code> being chosen specifies
    that schemas should be located from an LDAP server.</p>

```

<p><code>ServerAddress</code> is the address of the server that the client should communicate with. This should be the address of a MarketSite server, including any port and directory. Example:
<code>https://localhost:4433/xcc</code></p>

<p><code>SenderID</code> id the default MPID of the sender.</p>

<p><code>UserName</code> specifies the user name to use when connecting to the specified server. This should be the valid user name of the sending trading partner specified in SenderID.</p>

<p><code>Password</code> specifies the password to use when connecting to the specified server. This should be the valid password of the sending trading partner specified in SenderID.</p>

<p><code>TransmissionSettings</code> specifies the default tranmission settings to use.</p>

<p><code>DocumentTypeDefaults</code> specifies default transmission settings to use for specific document types.</p>

</explain>

<model>

```

<sequence>
  <choice>
    <element type="FileSystemSearch"/>
    <element type="LDAPSearch"/>
  </choice>
  <element name="ServerAddress" type="URI"/>
  <element name="SenderID" type="string"/>
  <element name="UserName" type="NMTOKEN"/>
  <element name="Password" type="string"/>
  <element type="TransmissionSettings" occurs="?" />
  <element type="DocumentTypeDefaults" occurs="*" />
  <element type="ProxyServerSettings" occurs="*" />
</sequence>
</model>

```

```
</elementtype>

<elementtype name="PollOnce">
  <explain>
    <h1>PollOnce</h1>
    <p>This element signifies that the client should only
    poll the incoming directory once.</p>
  </explain>

  <empty/>
</elementtype>

<elementtype name="PollContinuous">
  <explain>
    <h1>PollContinuous</h1>
    <p>This element signifies that the client should poll
    continuously and sleep between polls. This means the client
    will not exit unless interrupted.</p>
  </explain>

  <empty/>
  <attdef name="Interval" datatype="int">
    <explain>
      <h1>Interval</h1>
      <p>The number of milliseconds the client should sleep
      between ending one poll and starting the next.</p>
    </explain>
    <required/>
  </attdef>
</elementtype>

<elementtype name="FileSystemSearch">
  <explain>
    <h1>FileSystemSearch</h1>
    <p>This element signifies that the client should locate
    any schemas with the help of the file system.</p>

    <p><code>SchemaPath</code> specifies the path to use
    for finding schemas on the file system. This path should
    be the path to the root of the schema, not the path to
    the location of the schema. If more than one path is
```

```

        specified, they should be separated by semi-colons, (;).</p>
</explain>
<model>
    <element name="SchemaPath" type="string"/>
</model>
</elementtype>

<elementtype name="LDAPSearch">
    <explain>
        <h1>LDAPSearch</h1>
        <p>This element signifies that the client should locate
        any schemas with the help of the LDAP server.</p>

        <p><code>UserName</code> specifies the user name to
        use when logging into the LDAP server to locate schemas.</p>

        <p><code>Password</code> specifies the password to use
        when logging into the LDAP server to locate schemas.</p>

        <p><code>SchemaRoot</code> specifies the schema root to
        use when attempting to locate the schema in the
        LDAP server</p>
    </explain>
    <model>
        <sequence>
            <element name="UserName" type="NMTOKEN"/>
            <element name="Password" type="string"/>
            <element name="SchemaRoot" type="string"/>
        </sequence>
    </model>
</elementtype>

<elementtype name="DocumentTypeDefaults">
    <explain>
        <h1>DocumentTypeDefaults</h1>
        <p>This element is used to describe a set of transmission
        defaults for a specific document type.</p>
        <p><code>DocType</code> specifies the document type
        that the default transmission settings apply to.</p>
        <p><code>TransmissionSettings</code> specifies the
        default transmission settings to apply to the specified

```

```

        document type.</p>
    </explain>
</model>
    <sequence>
        <element name="DocType" type="string" />
        <element type="TransmissionSettings" />
    </sequence>
</model>
</elementtype>

<elementtype name="TransmissionSettings">
    <explain>
        <h1>TransmissionSettings</h1>
        <p>This element specifies a set of transmission settings.</p>
        <p><code>DefaultResponseMode</code> specifies the
        transmission mode to use.</p>
        <p><code>DefaultTimeOutSetting</code> specifies the
        time-out setting to use. This should be specified in
        seconds.</p>
    </explain>

    <model>
        <sequence>
            <element name="DefaultResponseMode" type="TransmissionMode"
                occurs = "?" />
            <element name="DefaultTimeOutSetting" type="int"
                occurs = "?" />
        </sequence>
    </model>
</elementtype>

<datatype name="TransmissionMode">
    <explain>
        <h1>TransmissionMode</h1>
        <p>This datatype describes the various transmission modes that the
        document can be sent in. Normally this value should only be set if
        the mode should be different for the default mode for the specified
        document type of the root document.

        <code>sync</code> =synchronous
        <code>oneway</code>= one way asynchronous
    </explain>

```

```
    <code>peer-peer</code> = two-way asynchronous</p>
</explain>

<enumeration datatype="NMTOKEN">
    <option>sync</option>
    <option>oneway</option>
    <option>peer-peer</option>
</enumeration>
</datatype>
<elementtype name="ProxyServerSettings">
    <model>
        <sequence>
            <element name="ProxySet" type="boolean" occurs="?" />
            <element name="ProxyHost" type="string" occurs="?" />
            <element name="ProxyPort" type="int" occurs="?" />
            <element name="HttpsProxyHost" type="string" occurs="?" />
            <element name="HttpsProxyPort" type="int" occurs="?" />
        </sequence>
    </model>
</elementtype>
</schema>
```


Appendix B

HotFS Service Config Schema

The following information is an example of a HotFS Service configuration schema.

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-
commerceone:document:com:commerceone:hotfs:schema:HotFSServiceConfig.sox$1.0">

  <elementtype name="HotFSServiceConfig">
    <explain>
      <hl>HotFSServiceConfig</hl>
      <p>This element describes a set of configurations for the
      HotFS service. It contains basic configurations for the
      service.</p>

      <p><code>DirectoryLocations</code> describes the locations
      of the various directories the servcic uses to
      communicate with the Trading Partner Application.</p>

      <p><code>ServiceConfigurations</code> describes
      a set of basic configurations.</p>
    </explain>

    <model>
      <sequence>
        <element type="DirectoryLocations"/>
        <element type="ServiceConfigurations"/>
      </sequence>
    </model>
  </elementtype>
```

```
<elementtype name="DirectoryLocations">
  <explain>
    <hl>DirectoryLocations</hl>
    <p>This element describes the the locations
    of the various directories the servcie uses to
    communicate with the Trading Partner Application.</p>

    <p><code>RootDirectory</code> specifies the root
    directory for the various file directories.</p>

    <p><code>ReceivedDDFDirectory</code> specifies the directory
    to place DDF files in. This directory should be relative
    to the RootDirectory. If the path starts with a path
    separator, the path is assumed to be absolute on the
    current drive.</p>

    <p><code>ReceivedDocumentDirectory</code> specifies
    the directory to place reply documents in. This directory
    should be relative to the RootDirectory. If the path
    starts with a path separator, the path is assumed to
    be absolute on the current drive.</p>

    <p><code>IncomingRDFDirectory</code> specifies the
    directory to poll for any incoming RDF files. This
    directory should be relative to the RootDirectory.
    If the path starts with a path separator, the path
    is assumed to be absolute on the current drive.</p>

    <p><code>ArchivedRDFDirectory</code> specifies the
    directory to place processed RDF files in. This
    directory should be relative to the RootDirectory.
    If the path starts with a path separator, the path
    is assumed to be absolute on the current drive.</p>

    <p><code>ArchivedRDFDocumentDirectory</code> specifies
    the directory to place documents that were successfully
    transmitted in a reply. This directory should be relative
    to the RootDirectory. If the path starts with a path
    separator, the path is assumed to be absolute on the
    current drive.</p>
```

<p><code>EnvelopeDirectory</code> specifies the directory to place envelopes from incoming requests that warrants an eventual reply. This directory should be relative to the RootDirectory. If the path starts with a path separator, the path is assumed to be absolute on the current drive.</p>

<p><code>ArchivedEnvelopeDirectory</code> specifies the directory to archive envelopes when a reply to a previous incoming request was successfully transmitted. This directory should be relative to the RootDirectory. If the path starts with a path separator, the path is assumed to be absolute on the current drive.</p>

<p><code>ErrorDirectory</code> specifies the directory to place error documents in. This directory should be relative to the RootDirectory. If the path starts with a path separator, the path is assumed to be absolute on the current drive.</p>

</explain>

<model>

<sequence>

<element name="RootDirectory" type="string"/>
<element name="ReceivedDDFDirectory" type="string"/>
<element name="ReceivedDocumentDirectory" type="string"/>
<element name="IncomingRDFDirectory" type="string"/>
<element name="ArchivedRDFDirectory" type="string"/>
<element name="ArchivedRDFDocumentDirectory" type="string"/>
<element name="EnvelopeDirectory" type="string"/>
<element name="ArchivedEnvelopeDirectory" type="string"/>
<element name="ErrorDirectory" type="string"/>

</sequence>

</model>

</elementtype>

<elementtype name="ServiceConfigurations">

<explain>

<h1>ServiceConfigurations</h1>

<p>This element describes the basic configurations for the HotFS service.</p>

```
<p><code>Sleeptime</code> states the sleep time
between the reply module polling. The value is
specified in milliseconds.</p>

<p><code>AcceptedDocument</code> may occur multiple
times, and each instance describes a document type
the service should subscribe to.</p>
</explain>

<model>
  <sequence>
    <element name="Sleeptime" type="int"/>
    <element type="AcceptedDocument" occurs="+"/>
  </sequence>
</model>
</elementtype>

<elementtype name="AcceptedDocument">
  <explain>
    <h1>AcceptedDocument</h1>
    <p>This element describes a document type that the
service should subscribe to.</p>

    <p><code>NameSpace</code> states the namespace
of the document type to subscribe to.</p>

    <p><code>LocalName</code> states the local name of
the document type to subscribe to. This </p>
  </explain>

  <model>
    <sequence>
      <element name="NameSpace" type="string"/>
      <element name="LocalName" type="NMTOKEN"/>
    </sequence>
  </model>
</elementtype>

</schema>
```

Appendix C

TDF Schema

The following information is an example of a HotFS TDF schema.

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM
"urn:x-commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-
commerceone:document:com:commerceone:hotfs:schema:TDF.sox$1.0">

  <elementtype name="TDF">
    <explain>
      <h1>TDF</h1>
      <p>This element describes the task to be assembled for an
      outgoing transmission. It describes tehe documents to
      send, as well as the properties for the transmission.</p>

      <p><code>TransmissionProperties</code> specifies the
      transmission properties to use for this specific
      transmission.</p>

      <p><code>RootDocument</code> describes the root document
      to be transmitted.</p>

      <p><code>Attachment</code> describes an attachment to
      be added to the transmission.</p>
    </explain>

    <model>
      <sequence>
        <element type="TransmissionProperties"/>
        <element type="RootDocument"/>
        <element type="Attachment" occurs="*" />
      </sequence>
    </model>
```

```
</elementtype>

<elementtype name="TransmissionProperties">
  <explain>
    <h1>TransmissionProperties</h1>
    <p>This element describes a set of properties for a
    transmission of a document. If the optional elements
    are not specified, then those properties will be
    determined from the defaults for the sending client.</p>

    <p><code>TaskID</code> specifies the unique name of
    the task. Will be used as the correlationID when the
    document is sent. It must be globally unique within the
    receiving MarketSite, and should minimally be
    sender's MPID + PO number.</p>

    <p><code>SenderID</code> is the MPID of the sender.</p>

    <p><code>RecipientID</code> is the MPID of the final
    destination.</p>

    <p><code>ResponseMode</code> describes the response
    mode of the transmission</p>

    <p><code>TimeOutSetting</code> specifies the time-out
    setting, in seconds, for the transmission. This
    is the time elapsed before the client will give up
    on getting a reply.</p>

    <p><code>Password</code> is the password to use for
    the transmission. This should be specified if a
    different password than the default password is to
    be used.</p>

    <p><code>Username</code> is the user name to use for
    the transmission. This should be specified if a
    different user name than the default user name is to
    be used.</p>
  </explain>

  <model>
```

```
<sequence>
  <element name="TaskID" type="string"/>
  <element name="RecipientID" type="string"/>
  <element name="SenderID" type="string" occurs = "?"/>
  <element name="ResponseMode" type="TransmissionMode" occurs="?"/>
  <element name="TimeOutSetting" type="int" occurs="?"/>
  <element name="Password" type="string" occurs="?"/>
  <element name="UserName" type="NMTOKEN" occurs="?"/>
</sequence>
</model>
</elementtype>

<elementtype name="RootDocument">
  <explain>
    <h1>RootDocument</h1>
    <p>This element describes the root document that should
    be transmitted.</p>

    <p><code>DocumentPath</code> describes the path to the
    document that is to be transmitted. The path is
    considered to be relative to the root directory
    specified for the client, but may be absolute if starting
    with a path separator or a drive letter.</p>

    <p><code>DocumentType</code> describes the document
    type of the document to transmitt. For sox this is the
    root element of the instance, for example PurchaseOrder.
    Routing will often be performed on the DocumentType,
    so it is important that the DocumentType is correct.</p>

    <p><code>ContentType</code> describes the source type of the
    contained document, such as SOX, XML, etc. The receiving app
    may deny a document if it is of an unuspported type.</p>
  </explain>

  <model>
    <sequence>
      <element name="DocumentPath" type="string"/>
      <element type="DocumentType"/>
      <element name="ContentType" type="string"/>
    </sequence>
```

```
</model>
</elementtype>

<elementtype name="DocumentType">
  <explain>
    <h1>DocumentType</h1>
    <p>This elementtype describes the type of the document that
    is to be transmitted.</p>

    <p><code>NameSpace</code> is the default namespace of
    the document, for example
    urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0</p>

    <p><code>LocalName</code> is the local name of the document,
    for example PurchaseOrder</p>
  </explain>

  <model>
    <sequence>
      <element name="NameSpace" type="string"/>
      <element name="LocalName" type="string"/>
    </sequence>
  </model>
</elementtype>

<elementtype name="Attachment">
  <explain>
    <h1>Attachment</h1>
    <p>This elementtype describes a document to be added
    as an attachment to the transmission. This document
    will be in addition to the RootDocument. An attachment
    may be named ur unnamed.</p>

    <p><code>DocumentPath</code> describes the path of the
    document that is to be attached to the transmission.
    The path is considered to be relative to the root directory
    specified for the client, but may be absolute if starting
    with a path separator or a drive letter.</p>

    <p><code>DocumentType</code> describes the document
    type of the attachment. For sox this is the
```



```
    root element of the instance, for example PurchaseOrder.</p>

    <p><code>Name</code> is the name of the attachment. This
    should only be used for named attachments.</p>
</explain>

<model>
  <sequence>
    <element name="DocumentPath" type="string"/>
    <element type="DocumentType"/>
    <element name="Name" type="URI" occurs="?" />
  </sequence>
</model>
</elementtype>

<datatype name="TransmissionMode">
  <explain>
    <h1>TransmissionMode</h1>
    <p>This datatype describes the various transmission modes that the
    document can be sent in. Normally this value should only be set if
    the mode should be different for the default mode for the specified
    document type of the root document.

    <code>sync</code> =synchronous
    <code>oneway</code>= one way asynchronous
    <code>peer-peer</code> = two-way asynchronous</p>
  </explain>

  <enumeration datatype="NMTOKEN">
    <option>sync</option>
    <option>oneway</option>
    <option>peer-peer</option>
  </enumeration>
</datatype>

</schema>
```

<p><code>EnvelopeDirectory</code> specifies the directory to place envelopes from incoming requests that warrants an eventual reply. This directory should be relative to the RootDirectory. If the path starts with a path separator, the path is assumed to be absolute on the current drive.</p>

<p><code>ArchivedEnvelopeDirectory</code> specifies the directory to archive envelopes when a reply to a previous incoming request was successfully transmitted. This directory should be relative to the RootDirectory. If the path starts with a path separator, the path is assumed to be absolute on the current drive.</p>

<p><code>ErrorDirectory</code> specifies the directory to place error documents in. This directory should be relative to the RootDirectory. If the path starts with a path separator, the path is assumed to be absolute on the current drive.</p>

</explain>

<model>

<sequence>

<element name="RootDirectory" type="string"/>

<element name="ReceivedDDFDirectory" type="string"/>

<element name="ReceivedDocumentDirectory" type="string"/>

<element name="IncomingRDFDirectory" type="string"/>

<element name="ArchivedRDFDirectory" type="string"/>

<element name="ArchivedRDFDocumentDirectory" type="string"/>

<element name="EnvelopeDirectory" type="string"/>

<element name="ArchivedEnvelopeDirectory" type="string"/>

<element name="ErrorDirectory" type="string"/>

</sequence>

</model>

</elementtype>

<elementtype name="ServiceConfigurations">

<explain>

<h1>ServiceConfigurations</h1>

<p>This element describes the basic configurations for the HotFS service.</p>

Appendix D

RDF Schema

The following information is an example of a HotFS RDF schema.

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-
commerceone:document:com:commerceone:hotfs:schema:RDF.sox$1.0">

  <elementtype name="RDF">
    <explain>
      <h1>RDF</h1>
      <p>This element describes the reply task to be assembled for a
reply to a previous incoming request. It describes the document
to send in the reply.</p>

      <p><code>TaskID</code> describes the task ID of the
reply. This TaskID value must match the TaskID value
of the previous incoming request.</p>

      <p><code>RootDocument</code> describes the root document
to be transmitted.</p>
    </explain>

    <model>
      <sequence>
        <element name="TaskID" type="string"/>
        <element type="RootDocument"/>
      </sequence>
    </model>
  </elementtype>
```

```
<elementtype name="RootDocument">
  <explain>
    <h1>RootDocument</h1>
    <p>This element describes the root document that should
    be sent in the reply.</p>

    <p><code>DocumentPath</code> describes the path to the
    document that should be sent in the reply. The path is
    considered to be relative to the root directory
    specified for the service, but may be absolute if starting
    with a path separator or a drive letter.</p>

    <p><code>ContentType</code> describes the source type of the
    contained document, such as SOX, XML, etc. The service
    may deny a document if it is of an unsupported type.</p>
  </explain>

  <model>
    <sequence>
      <element name="DocumentPath" type="string"/>
      <element type="DocumentType"/>
      <element name="ContentType" type="string"/>
    </sequence>
  </model>
</elementtype>

<elementtype name="DocumentType">
  <explain>
    <h1>DocumentType</h1>
    <p>This elementtype describes the type of the document that
    is to be sent in the reply.</p>

    <p><code>Namespace</code> is the default namespace of
    the document, for example
    urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0</p>

    <p><code>LocalName</code> is the local name of the document,
    for example PurchaseOrder</p>
  </explain>
```

```
<model>
  <sequence>
    <element name="Namespace" type="string"/>
    <element name="LocalName" type="string"/>
  </sequence>
</model>
</elementtype>

</schema>
```


Appendix E

DDF Schema

The following information is an example of a HotFS DDF schema.

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-
commerceone:document:com:commerceone:hotfs:schema:DDF.sox$1.0">

  <elementtype name="DDF">
    <explain>
      <h1>DDF</h1>
      <p>This element describes an incoming document. It details
some transmission information, as well as information on the
incoming document.</p>

      <p><code>TransmissionInfo</code> describes some of the
transmission variables of the incoming document.</p>

      <p><code>RootDocument</code> describes the root document
that was received.</p>
    </explain>

    <model>
      <sequence>
        <element type="TransmissionInfo"/>
        <element type="RootDocument"/>
        <element type="Attachment" occurs="*" />
      </sequence>
    </model>
  </elementtype>
```

```
<elementtype name="TransmissionInfo">
  <explain>
    <h1>TransmissionInfo</h1>
    <p>This element describes some of the transmission variables
    from the incoming transmission that the receiving application
    may find useful.</p>

    <p><code>TaskID</code> specifies the unique name of
    the task and is the correlationID of the incoming envelope.
    If this transmission was a reply to a previously outgoing
    transmission, the TaskID will match the TaskID of that
    transmission.</p>

    <p><code>RecipientID</code> is the MPID of the recipient.</p>

    <p><code>SenderID</code> is the MPID of the sender</p>

    <p><code>ResponseMode</code> describes the response
    mode of the transmission. If the mode was sync or peer-peer
    a reply is warranted. If the mode was sync, the reply
    is time sensitive.</p>
  </explain>

  <model>
    <sequence>
      <element name="TaskID" type="string"/>
      <element name="RecipientID" type="string"/>
      <element name="SenderID" type="string"/>
      <element name="ResponseMode" type="TransmissionMode"/>
    </sequence>
  </model>
</elementtype>

<elementtype name="RootDocument">
  <explain>
    <h1>RootDocument</h1>
    <p>This element describes the root document that was
    received.</p>

    <p><code>DocumentPath</code> describes the path to the
    document that was received and written to the file
```

```
system. The path is absolute.</p>

<p><code>DocumentType</code> describes the document
type of the incoming document. For sox this is the
root element of the instance, for example PurchaseOrder.</p>

<p><code>ContentType</code> describes the source type of the
incoming document, such as SOX.</p>
</explain>

<model>
  <sequence>
    <element name="DocumentPath" type="string"/>
    <element type="DocumentType"/>
    <element name="ContentType" type="string"/>
  </sequence>
</model>
</elementtype>

<elementtype name="DocumentType">
  <explain>
    <h1>DocumentType</h1>
    <p>This elementtype describes the type of the document that
was received.</p>

    <p><code>NameSpace</code> is the default namespace of
the document, for example
urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0</p>

    <p><code>LocalName</code> is the local name of the document,
for example PurchaseOrder</p>
  </explain>

  <model>
    <sequence>
      <element name="NameSpace" type="string"/>
      <element name="LocalName" type="string"/>
    </sequence>
  </model>
</elementtype>
```

```
<elementtype name="Attachment">
  <explain>
    <h1>Attachment</h1>
    <p>This elementtype describes a document that was added
    as an attachment to the transmission. An attachment
    may be named or unnamed.</p>

    <p><code>DocumentPath</code> describes the path of the
    document that was attached to the transmission. If
    the correct file extension was set by the sender,
    it will be set as the extension of the file name. If
    not the file extension will be "att", and the mime type
    will be the only informtaion of the type of the document.</p>

    <p><code>MimeType</code> describes the mime type of
    the attachment. The mime type is the type of the document,
    and should determine how the document should be processed.
    If the correct file extension can be determined for
    the file, it will be set as the extension for the
    attachment, as described in DocumentPath.</p>

    <p><code>Name</code> is the name of the attachment. This
    is only present for named attachments.</p>
  </explain>

  <model>
    <sequence>
      <element name="DocumentPath" type="string"/>
      <element name="MimeType" type="string"/>
      <element name="Name" type="URI" occurs="?"/>
    </sequence>
  </model>
</elementtype>

<datatype name="TransmissionMode">
  <explain>
    <h1>TransmissionMode</h1>
    <p>This datatype describes the various transmission modes that a
    document can arrive in. The possible values are:

    <code>sync</code> =synchronous - warrants a reply. Time
```

```
sensitive.  
<code>oneway</code>= one way asynchronous - does not warrant a reply  
<code>peer-peer</code> = two-way asynchronous - warrants a reply.</p>  
</explain>  
  
<enumeration datatype="NMTOKEN">  
  <option>sync</option>  
  <option>oneway</option>  
  <option>peer-peer</option>  
</enumeration>  
</datatype>  
  
</schema>
```


Appendix F

HOTFS Error Schema

The following information is an example of a HotFS Error schema.

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "urn:x-
commerceone:document:com:commerceone:xdk:xml:schema.dtd$1.0">

<schema uri="urn:x-
commerceone:document:com:commerceone:hotfs:schema:HotFSError.sox$1.0">

  <elementtype name="HotFSError">
    <explain>
      <h1>HotFSError</h1>
      <p>This element describes an error encountered by the
HotFS Client. It gives an error message, an approximate
point where the error occurred, and the TaskID of the
task that failed.</p>

      <p><code>TaskID</code>is the Task ID of the task
that resulted in an error.</p>

      <p><code>OccurredWhile</code> describes the point in
the processing of the task where the error occurred.</p>

      <p><code>message</code> contains an error message
describint the error that occurred.</p>
    </explain>

    <model>
      <sequence>
        <element name="TaskID" type="string"/>
        <element name="OccurredWhile" type="ErrorType"/>
        <element name="message" type="string"/>
      </sequence>
    </model>
  </elementtype>
</schema>
```

```
</sequence>
</model>
</elementtype>

<datatype name="ErrorType">
  <explain>
    <h1>ErrorType</h1>
    <p>This datatype enumerates points in the task processing
    where an error could occur.

    <code>Unknown</code> signifies that the application
    was unable to determine when the error occurred.
    <code>ReadingTDF</code> signifies that the error
    occurred when the application was reading the TDF file.
    <code>ProcessingTDF</code> signifies that the error
    occurred when the application was processing the TDF
    file.
    <code>ReadingDocument</code> signifies that the error
    occurred when the application was reading the document
    file.
    <code>ProcessingDocument</code> signifies that the error
    occurred when the application was processing the document
    file.
    <code>ReadingAttachment</code> signifies that the error
    occurred when the application was reading an attachment
    file.
    <code>SendingDocument</code> signifies that the error
    occurred when the application was sending the document.
    <code>WritingDocument</code> signifies that the error
    occurred when the application was writing a reply
    document to the file system.
    <code>WritingDDF</code> signifies that the error
    occurred when the application was writing a DDF file to
    the file system.
    <code>ReadingRDF</code> signifies that the error
    occurred when the application was reading an RDF file.
    <code>ProcessingRDF</code> signifies that the error
    occurred when the application was processing an RDF file.
    <code>WritingEnvelope</code> signifies that the error
    occurred when the application was writing an envelope to
    the file system.
```

```
<code>ReadingEnvelope</code> signifies that the error
occurred when the application was reading an envelope from
a file.
<code>RetreivingAttachment</code> signifies that the error
occurred when the application was retreiving an attachment
from an envelope.
<code>WritingAttachment</code> signifies that the error
occurred when the application was writing an attachment file
to the file system.</p>
</explain>

<enumeration datatype="NMTOKEN">
  <option>Unknown</option>
  <option>ReadingTDF</option>
  <option>ProcessingTDF</option>
  <option>ReadingDocument</option>
  <option>ProcessingDocument</option>
  <option>ReadingAttachment</option>
  <option>SendingDocument</option>
  <option>WritingDocument</option>
  <option>WritingDDF</option>
  <option>ReadingRDF</option>
  <option>ProcessingRDF</option>
  <option>WritingEnvelope</option>
  <option>ReadingEnvelope</option>
  <option>RetreivingAttachment</option>
  <option>WritingAttachment</option>
</enumeration>
</datatype>

</schema>
```


Appendix G

Mapping of File Extensions to Mime Schema

The following information is an example of a mapping of file extensions to mime schema.

```
# comments begin with a '#'
# the format is <mime type> <space separated file extensions>
# for example:
# text/plain txt text TXT xml html
# this would map file.txt, file.text, and file.TXT to
# the mime type "text/plain"

application./vbnd.ms-powerpoi      PPS POT PWZ MPP   pps pot pwz mpp
application/hta                     HTA hta
application/msword                  DOC DOT WIZ RTF  doc dot wiz rtf
application/pdf                     PDF pdf
application/pics-rules              PRF prf
application/pkcs10                  P10 p10
application/pkcs7-mime              P7C P7B SPC p7c p7b spc
application/pkcs7-signature         P75 p75
application/pkix.cert               CER CRT DER cer crt der
application/pkix-crl                CRL crl
application/vbnd.ms-pki.certst      SST sst
application/vnd                     fdf
application/vnd.fdf                 FDF
application/vnd.ms-excel             XLB XLS xlb xls
application/vnd.ms-pki.pko          PKO pko
application/vnd.ms-pki.secca        CAT cat
application/vnd.ms-pki.stl          STL stl
application/vnd.ms-powerpoi         PPA PPT ppa ppt
application/vnd.ms-project          MPD MPT mpd mpt
application/x.gzip                  HQX LZH TAR TAZ TGZ TZ UU UUE ARC ARJ
```

Mapping of File Extensions to Mime Schema

B64	BHX	GZ	XXE	Z	ZIP	hqx	lzh	tar	taz	tgz	tz	uu	uue	arc	arj	b64	bhx	gz	xxe	z	zip	
application/x-cdf														CDF	cdf							
application/x-internet-signup														INS	ISP	ins	isp					
application/x-iphone														III	iii							
application/x-msdownload														EXE	exe							
application/x-pkcs12														P12	PFX	p12	pxf					
application/x-pkcs7-certreqre														P7R	p7r							
audio/aiff														AIF	AIFC	AIFF	aif	aifc	aiff			
audio/basic														AU	SND	au	snd					
audio/mid														MID	RMI	mid	rmi					
audio/mpeg														MP3	mp3							
audio/x-mpegurl															M3U	m3u						
audio/x-pn-realaudio														RA	RAM	RM	RMM	ra	ram	rm	rmm	
audio/x-wav														WAV	wav							
image/bmp														BMP	bmp							
image/gif														JPE	JPEG	JPG	JFIF	jpe	jpeg	jpg	jfif	
image/png														PNG	png							
image/tiff														TIF	tif							
image/x.xbitmap															XBM	xbm						
image/x-icon														ICO	ico							
image/x-jg														ART	art							
java/*														CLASS	class							
message/rfc822														MHT	MHTML	EML	NWS	mht	mhtml	eml	nws	
text/asa														ASA	ASP	asp	asa					
text/css														CSS	css							
text/h323														323								
text/html														HTM	HTML	HTW	HTX	PLG	STM	htm	html	htw
														htx	plg	stm						
text/iuls														ULS	uls							
text/java														JAVA	java							
text/plain														DIC	EXC	LOG	SCP	TXT	WTX	dic	exc	log
														scp	txt	wtx						
text/scriptlet														SCT	WSC	sct	wsc					
text/webviewhtml														HTT	htt							
text/xml														XML	XSL	xml	xsl					
text/x-vcard														VCF	vcf							
video/avi														AVI	avi							
video/mpeg														M1V	MP2	MPA	MPE	MPEG	MPG	MPV2	m1v	mp2
														mpa	mpe	mpeg	mpg	mpv2				
video/quicktime														MOV	QT	mov	qt					
video/vdo														VDO	vdo							

Mapping of File Extensions to Mime Schema

video/x-ms-asf
video/x-sgi-movie

ASF LSF ASX LSX asf lsf asx lsx
BSC CAB CGM DCT EYB FN bsc cab cgm
dct eyb fn

Appendix H

UUID Generation Classes

The following information is an example of UUID generation classes.

UUID CLASSES

```
import java.net.InetAddress;
import java.util.StringTokenizer;
import java.util.Random;
import java.util.Date;

/**
 * Uuid is a IETF identity standard. 16 bytes long, "unique" id
 */

public class Uuid implements Identity
{
    // -----
    // Interface Identity

    /**
     * Identity as a printable form of the Identity.
     */
    public String toString()
    {
        StringBuffer sb = new StringBuffer(40);

        sb.append(toHex(m_bytes[0]));
        sb.append(toHex(m_bytes[1]));
        sb.append(toHex(m_bytes[2]));
        sb.append(toHex(m_bytes[3]));
        sb.append("-");
        sb.append(toHex(m_bytes[4]));
        sb.append(toHex(m_bytes[5]));
        sb.append("-");
    }
}
```

```
sb.append(toHex(m_bytes[6]));
sb.append(toHex(m_bytes[7]));
sb.append("-");
sb.append(toHex(m_bytes[8]));
sb.append(toHex(m_bytes[9]));
sb.append("-");
sb.append(toHex(m_bytes[10]));
sb.append(toHex(m_bytes[11]));
sb.append(toHex(m_bytes[12]));
sb.append(toHex(m_bytes[13]));
sb.append(toHex(m_bytes[14]));
sb.append(toHex(m_bytes[15]));

return sb.toString();
}

/**
 * Identity as a low level, space efficient
 * representation.
 */
public byte[] toBytes()
{
return m_bytes;
}

/**
 * Enforce the Identity implementations to have an equals
 * method.
 */
public boolean equals(Object obj)
{
    if (!(obj instanceof Uuid))
        return false;

for (int i=0; i<m_bytes.length; i++)
{
    if (((Uuid)obj).m_bytes[i] != m_bytes[i])
    {
        return false;
    }
}
}
```

```

return true;
}

public int hashCode()
{
    int code=0;
    for (int i=0;i<m_bytes.length;i++)
    {
        code = code+m_bytes[i];
    }
    return code;
}

/**
 * Return a Format hint. Right now we want to focus
 * on UUIDs, but understand there is a need for short
 * identities sometimes.
 */
public int formatHint()
{
    return Identity.UUID;
}

// -----
// Creation

/**
 * Create a Uuid from the string format
 */
public static Uuid internalize(String uuid_string)
{
    //04732ee2-7769-1000-9f05-0a00001c0001
    try
    {
        String uuid_trim = uuid_string.trim();

        if (uuid_trim.length() != 36 ||
            uuid_trim.charAt(8) != '-' ||
            uuid_trim.charAt(13) != '-' ||
            uuid_trim.charAt(18) != '-')
    }

```

```
        uuid_trim.charAt(23) != '-')
    {
        return null;
    }

    int index = 0;
    byte[] bytes = new byte[16];

    StringTokenizer st = new StringTokenizer(uuid_trim, "-");
    while (st.hasMoreTokens())
    {
        String hex_str = st.nextToken();

        for (int i=0; i<hex_str.length(); i+=2)
        {
            String sub_str = hex_str.substring(i, i+2);
            bytes[index++] = (Integer.valueOf(sub_str, 16)).byteValue();
        }
    }

    return new Uuid(bytes);
}
catch (Exception ex)
{
    return null;
}

public static Uuid create()
{
    return new Uuid(s_generator.newId());
}

// -----
// Private Constructor

/**
 * Create a new Unique Uuid.
 */
private Uuid(byte[] bytes)
{
```



```

m_bytes = bytes;
}

// -----
// Private Memeber Functions

/**
 * Bytes are signed 8 bits. Values between -128 and 127.
 * I want to see it as an unsigned 8 bits, between 0
 * and 255.
 */
private static String toHex(byte b)
{
int j = (int)((b>=0)?b:2*Byte.MAX_VALUE+b+2);
return s_hex_values[j];
}

// -----
// Private Static Members

private static final UuidGenerator s_generator = new UuidGenerator();

private static final String[] s_hex_values = {
"00", "01", "02", "03", "04", "05", "06", "07",
"08", "09", "0a", "0b", "0c", "0d", "0e", "0f",
"10", "11", "12", "13", "14", "15", "16", "17",
"18", "19", "1a", "1b", "1c", "1d", "1e", "1f",
"20", "21", "22", "23", "24", "25", "26", "27",
"28", "29", "2a", "2b", "2c", "2d", "2e", "2f",
"30", "31", "32", "33", "34", "35", "36", "37",
"38", "39", "3a", "3b", "3c", "3d", "3e", "3f",
"40", "41", "42", "43", "44", "45", "46", "47",
"48", "49", "4a", "4b", "4c", "4d", "4e", "4f",
"50", "51", "52", "53", "54", "55", "56", "57",
"58", "59", "5a", "5b", "5c", "5d", "5e", "5f",
"60", "61", "62", "63", "64", "65", "66", "67",
"68", "69", "6a", "6b", "6c", "6d", "6e", "6f",
"70", "71", "72", "73", "74", "75", "76", "77",
"78", "79", "7a", "7b", "7c", "7d", "7e", "7f",
"80", "81", "82", "83", "84", "85", "86", "87",
"88", "89", "8a", "8b", "8c", "8d", "8e", "8f",

```

UUID Generation Classes

```
"90", "91", "92", "93", "94", "95", "96", "97",
"98", "99", "9a", "9b", "9c", "9d", "9e", "9f",
"a0", "a1", "a2", "a3", "a4", "a5", "a6", "a7",
"a8", "a9", "aa", "ab", "ac", "ad", "ae", "af",
"b0", "b1", "b2", "b3", "b4", "b5", "b6", "b7",
"b8", "b9", "ba", "bb", "bc", "bd", "be", "bf",
"c0", "c1", "c2", "c3", "c4", "c5", "c6", "c7",
"c8", "c9", "ca", "cb", "cc", "cd", "ce", "cf",
"d0", "d1", "d2", "d3", "d4", "d5", "d6", "d7",
"d8", "d9", "da", "db", "dc", "dd", "de", "df",
"e0", "e1", "e2", "e3", "e4", "e5", "e6", "e7",
"e8", "e9", "ea", "eb", "ec", "ed", "ee", "ef",
"f0", "f1", "f2", "f3", "f4", "f5", "f6", "f7",
"f8", "f9", "fa", "fb", "fc", "fd", "fe", "ff"
};

// -----
// Private Members

private final byte[] m_bytes;
}

/**
 * Written by Kevin Hughes, stripped by Kenneth Persson
 */

class UuidGenerator
{
    // -----
    // Constructor

    public UuidGenerator()
    {
        m_serial_number = s_gen.nextLong();
    }

    public synchronized byte[] newId()
    {
        byte[] bytes = new byte[16];

        long time_stamp = ((new Date()).getTime() + TIME_DIFFERENCE) * 10;
```

```

    long clock = ++m_serial_number;

    long time_low = (long)(time_stamp & 0xFFFFFFFF);

    bytes[0] = (byte)((time_low >> 24) & 255);
    bytes[1] = (byte)((time_low >> 16) & 255);
    bytes[2] = (byte)((time_low >> 8) & 255);
    bytes[3] = (byte)(time_low & 255);

    int time_mid = (int)((time_stamp >> 32) & 0xFFFF);

    bytes[4] = (byte)((time_mid >> 8) & 255);
    bytes[5] = (byte)(time_mid & 255);

    int time_hi = ((int)((time_stamp >> 48) & 0xFFFF) | (1 << 12));

    bytes[6] = (byte)((time_hi >> 8) & 255);
    bytes[7] = (byte)(time_hi & 255);

    long clock_hi = ((clock & 0x3F00) >> 8) | 0x80;

    bytes[8] = (byte)(clock_hi & 255);

    long clock_low = (clock & 0xFF);

    bytes[9] = (byte)(clock_low & 255);

    bytes[10] = s_host_addr[0];
    bytes[11] = s_host_addr[1];
    bytes[12] = s_host_addr[2];
    bytes[13] = s_host_addr[3];

    bytes[14] = 0;
    bytes[15] = 1;

    return bytes;
  }

  private static byte[] getHostAddress()
  {

```

```
try {
    return InetAddress.getLocalHost().getAddress();
}
catch (Exception ex)
{
    return new byte[4];
}

// -----
// Private Members

// This is the time difference in milliseconds from 00:00:00:00,
// 15 October 1582 GMT to 00:00:00:00, 1 January 1970 GMT
private static final long TIME_DIFFERENCE = 12219292800000L;

private static final Random s_gen = new Random();
private static final byte[] s_host_addr = getHostAddress();

private long m_serial_number = s_gen.nextLong();
}
```

Appendix I

encryptpassword Tool

The encryptpassword tool is included with your XCC installation and is used to obtain an encrypted value for a password from an unencrypted value. The tool is located in the `${XCCROOT}/upgrade/bin` directory of XCC.

If you want to obtain the encrypted value for a password, follow these steps:

1. Make sure you have XCC 3.2 installed on your system.
2. Bring up a Cygwin session and change directories (cd) to `${XCCROOT}/upgrade/bin` directory.
3. Type: `sh ./encryptpassword <password>` where `<password>` is the **unencrypted** value of the password. For example, if you want to obtain the **encrypted** value for the password “myPassword”, type the following:

```
sh ./encryptpassword myPassword
```

The **encrypted** value “mAJMjy/gt22FOFizplgZuA==” is returned. This is the value you need to enter into any of the password fields.

