[MS-FSCDFT]:
Content Distributor Fault Tolerance Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
02/19/2010	1.0	Major	Initial Availability
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction 5
1.1 Glossary 5
1.2 References 5
1.2.1 Normative References 5
1.2.2 Informative References 6
1.3 Protocol Overview 6
1.4 Relationship to Other Protocols 6
1.5 Prerequisites/Preconditions 6
1.6 Applicability Statement 7
1.7 Versioning and Capability Negotiation 7
1.8 Vendor-Extensible Fields 7
1.9 Standards Assignments 7

2 Messages 8
2.1 Transport 8
2.2 Common Data Types 8
2.2.1 cht::core::guarantee 8
2.2.2 cht::core::feeding_priority 8
2.2.3 cht::core::guarantee_set 8
2.2.4 processing::unknown_collection 9
2.2.5 core::unsupported_guarantee_set 9

3 Protocol Details 10
3.1 processing::master_dispatcher Server Details 13
3.1.1 Abstract Data Model 13
3.1.2 Timers 14
3.1.3 Initialization 14
3.1.4 Message Processing Events and Sequencing Rules 14
3.1.4.1 processing::master_dispatcher::register_dispatcher 14
3.1.5 Timer Events 15
3.1.6 Other Local Events 15
3.2 processing::master_dispatcher Client Details 15
3.2.1 Abstract Data Model 15
3.2.2 Timers 15
3.2.3 Initialization 15
3.2.4 Message Processing Events and Sequencing Rules 15
3.2.4.1 processing::master_dispatcher::register_dispatcher 15
3.2.5 Timer Events 16
3.2.6 Other Local Events 16
3.3 processing::session_factory Server Details 16
3.3.1 Abstract Data Model 16
3.3.2 Timers 16
3.3.3 Initialization 16
3.3.4 Message Processing Events and Sequencing Rules 16
3.3.5 Timer Events 16
3.3.6 Other Local Events 17
3.4 processing::session_factory Client Details 17
3.4.1 Abstract Data Model 17
3.4.2 Timers 17
3.4.3 Initialization 17

3.4.4 Message Processing Events and Sequencing Rules 17
3.4.5 Timer Events 17
3.4.6 Other Local Events 18
3.5 processing::dispatcher_node Server Details 18
3.5.1 Abstract Data Model 18
3.5.2 Timers 19
3.5.3 Initialization 19
3.5.4 Message Processing Events and Sequencing Rules 19
3.5.4.1 processing::dispatcher_node::create_session 19
3.5.4.2 processing::dispatcher_node::create 20
3.5.4.3 processing::dispatcher_node::recreate 21
3.5.4.4 processing::dispatcher_node::destroy_session 22
3.5.4.5 processing::dispatcher_node::remove_sessions 23
3.5.4.6 processing::dispatcher_node::get_sessions 23
3.5.4.7 processing::dispatcher_node::get_load 24
3.5.5 Timer Events 24
3.5.6 Other Local Events 24
3.6 processing::dispatcher_node Client Details 24
3.6.1 Abstract Data Model 24
3.6.2 Timers 24
3.6.3 Initialization 24
3.6.4 Message Processing Events and Sequencing Rules 24
3.6.5 Timer Events 24
3.6.6 Other Local Events 25

4 Protocol Examples 26
4.1 Registering a Backup Content Distributor 26
4.2 Sample Code 26
4.2.1 Protocol Server Initialization 26
4.2.2 Protocol Client Initialization 26
4.2.3 Protocol Client Message 26
4.2.4 Server Response 27

5 Security 28
5.1 Security Considerations for Implementers 28
5.2 Index of Security Parameters 28

6 Appendix A: Full FSIDL 29

7 Appendix B: Product Behavior 31

8 Change Tracking 32

9 Index 33

1 Introduction

This document specifies the protocol that multiple **content distributors** use to determine which one is the master content distributor and which ones are backup content distributors. The use of multiple content distributors increases performance and robustness; if the master content distributor becomes unavailable, one of the backup content distributors becomes the new master content distributor.

1.1 Glossary

The following terms are defined in [MS-OFCGLOS]:

abstract object reference (AOR)
base port
callback message
Cheetah
Cheetah checksum
Cheetah entity
client proxy
content client
content collection
content distributor
FAST Search Interface Definition Language (FSIDL)
host name
indexing dispatcher
indexing node
item
name server

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-FSCF] Microsoft Corporation, "Content Feeding Protocol Specification", November 2009.

[MS-FSCHT] Microsoft Corporation, "Cheetah Data Structure", November 2009.

[MS-FSDP] Microsoft Corporation, "Document Processing Protocol Specification", November 2009.

[MS-FSMW] Microsoft Corporation, "Middleware Protocol Specification", November 2009.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-FSO] Microsoft Corporation, "FAST Search System Overview", November 2009.

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary", June 2008.

1.3 Protocol Overview

One or more content distributors are part of a session-based **item** feeding chain, wherein a **content client** establishes a session with an **indexing node**. The content client uses the established session to send information about items to add, update, or remove from the index, and the indexing node sends asynchronous status **callback messages** about the items to the content client.

In a setup that has multiple content distributors, the sessions that are used in the feeding chain are distributed across all the content distributors, which improve performance and robustness. When the content client requests the master content distributor to create a new session, the master content distributor either performs the request or forwards it to one of the backup content distributors, to distribute the sessions across the available content distributors. The master content distributor returns an interface for the created session to the content client. Therefore, the content client is not aware of, nor does it have to be, whether a specific session that is used for feeding is routed through a master or backup content distributor. The concept of master and backup content distributors is therefore relevant only when the session is created.

This protocol enables the content distributors to determine which one becomes the master content distributor and which ones are backup content distributors. Each backup content distributor continues to monitor the master content distributor. If the master content distributor becomes unavailable, one of the backup content distributors takes the role of master content distributor.

For more information about the system overview, see [MS-FSO].

1.4 Relationship to Other Protocols

The protocol relies on the **Cheetah** data format to serialize data, as described in [MS-FSCHT], and on the Middleware Protocol to transport data, as described in [MS-FSMW].

The following diagram shows the underlying messaging and transport stack that this protocol uses.

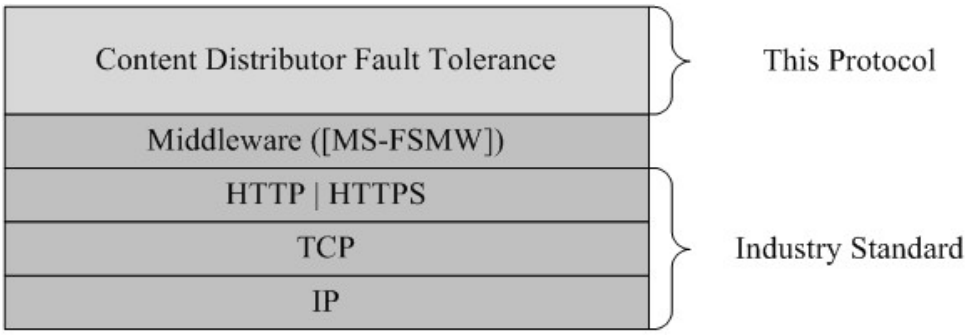

Figure 1: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

The protocol client and protocol server are expected to know the location and connection information of the shared **name server**.

1.6 Applicability Statement

This protocol enables content distributors to register as the master content distributor while others become backup content distributors. The content distributors are part of an extended session-based item feeding chain.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

Messages MUST be transported via the Middleware Protocol, as specified in [MS-FSMW]. Data serialization MUST be performed by using the Cheetah Data Format, as specified in [MS-FSCHT].

2.2 Common Data Types

FAST Search Interface Definition Language (FSIDL) data types are encoded as specified in [MS-FSMW] section 2. **Cheetah entities** are encoded as specified in [MS-FSCHT] section 2. The Cheetah type identifier and the **Cheetah checksum** for the Cheetah entities MUST be integers, as specified in the following table.

Cheetah entity	Cheetah type identifier	Cheeta checksum
cht::core::guarantee	3	-1479218033
cht::core::feeding_priority	7	-1479218033
cht::core::guarantee_set	9	-1479218033

The full FSIDL for this protocol is specified in section 6.

2.2.1 cht::core::guarantee

The **guarantee** Cheetah entity is the parent class for the **feeding_priority** Cheetah entity, as specified in section 2.2.2. Cheetah entity specification for **guarantee:**

```
entity guarantee {
};
```

2.2.2 cht::core::feeding_priority

The **feeding_priority** Cheetah entity specifies the priority for sending items to the protocol server. Cheetah entity specification for **feeding_priority**:

```
entity feeding_priority : guarantee {
  attribute int priority;
};
```

priority: An integer that MUST be 0.

2.2.3 cht::core::guarantee_set

The **guarantee_set** Cheetah entity contains a collection of **guarantee** objects, which are specified in section 2.2.1. Cheetah entity specification for **guarantee_set**:

```
entity guarantee_set {
  collection guarantee guarantees;
};
```

guarantees: A collection of **guarantee** Cheetah entities.

2.2.4 processing::unknown_collection

The **unknown_collection** exception specifies that the **content collection** is unknown. The **unknown_collection** exception is specified by the following FSIDL specification:

```
exception unknown_collection {
    string what;
};
```

what: A string that explains the cause of the exception.

2.2.5 core::unsupported_guarantee_set

The **unsupported_guarantee_set** exception specifies that the content distributor cannot create or recreate a session. The **unsupported_guarantee_set** exception is specified by the following FSIDL specification:

```
exception unsupported_guarantee_set {
    string what;
};
```

what: A string that explains the cause of the exception.

3 Protocol Details

This document specifies a protocol that is used between two or more content distributor nodes, where one is the master content distributor node and the others are backup content distributor nodes. This protocol consists of the three interfaces: **processing::session_factory**, **processing::master_dispatcher**, and **processing::dispatcher_node**.

The **processing::session_factory** interface is used to elect master and backup content distributors. The **processing::master_dispatcher** interface allows backup content distributors to register with the master.

The master content distributor uses the **processing::dispatcher_node** interface to forward incoming session requests to the master content distributor and backup content distributors.

The role as protocol client and protocol server therefore depends on the protocol that is used and the role of the content distributor. The master content distributor MUST implement and activate the **processing::session_factory** server object, as specified in section 3.3, and the **processing::master_dispatcher** server object, as specified in section 3.1. All content distributors, including the master content distributor, MUST implement and activate the **processing::dispatcher_node** server object, as specified in section 3.5.

Initially, a content distributor node does not perform either the role of master content distributor or backup content distributor. To determine its role, each content distributor MUST locate the registered master content distributor. If the master content distributor is not found, the content distributor MUST become the registered master content distributor. Otherwise, the content distributor registers with the master content distributor to become the backup content distributor. A backup content distributor monitors the master content distributor, and it takes the role of master content distributor when the master content distributor is not available.

The content distributor MUST register and activate the **processing::session_factory** server object and the **processing::master_dispatcher** server object and register the server objects with the name server to become a master content distributor. It performs this by using the **bind** method, as specified in section 3.3.3 and section 3.1.3. The master content distributor MUST register the **processing::session_factory** server object before the **processing::master_dispatcher** server object.

The content distributor MUST locate the master content distributor by using the **resolve** method, as specified in section 3.4.3, to determine whether the master content distributor registered the **processing::session_factory** interface with the name server. The content distributor uses the **__ping** method, as specified in [MS-FSMW] section 3, on the **processing::session_factory** interface to verify that a registered master content distributor is active.

The backup content distributor MUST invoke the **__ping** method of the **processing::session_factory** server object to monitor the master content distributor.

The method invocation sequence, and the number of times that the **bind**, **resolve**, and **__ping** methods are invoked, are implementation specific.

The following two-part figure specifies an example sequence.

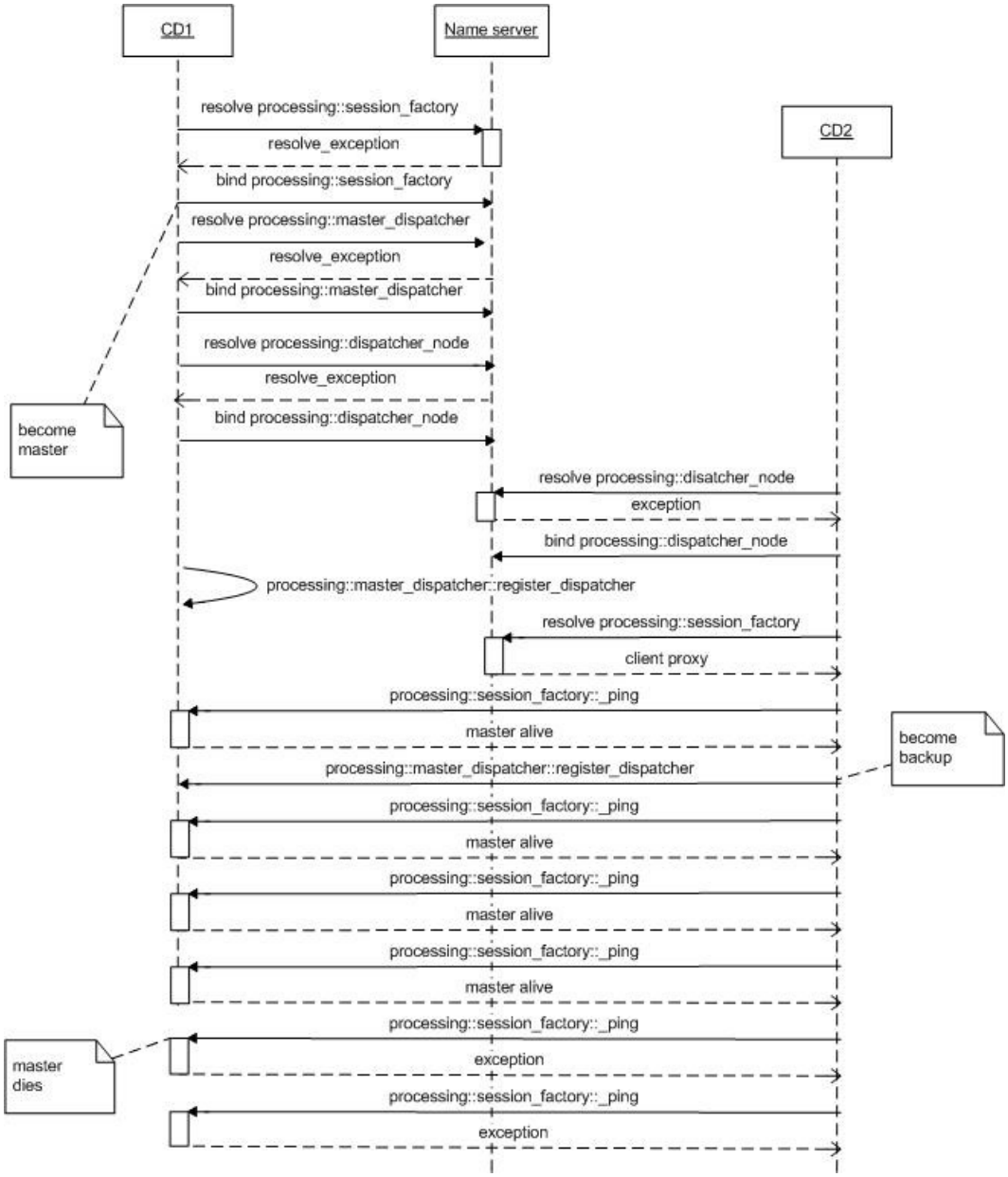

Figure 2: Determining the master node and the backup node (part 1)

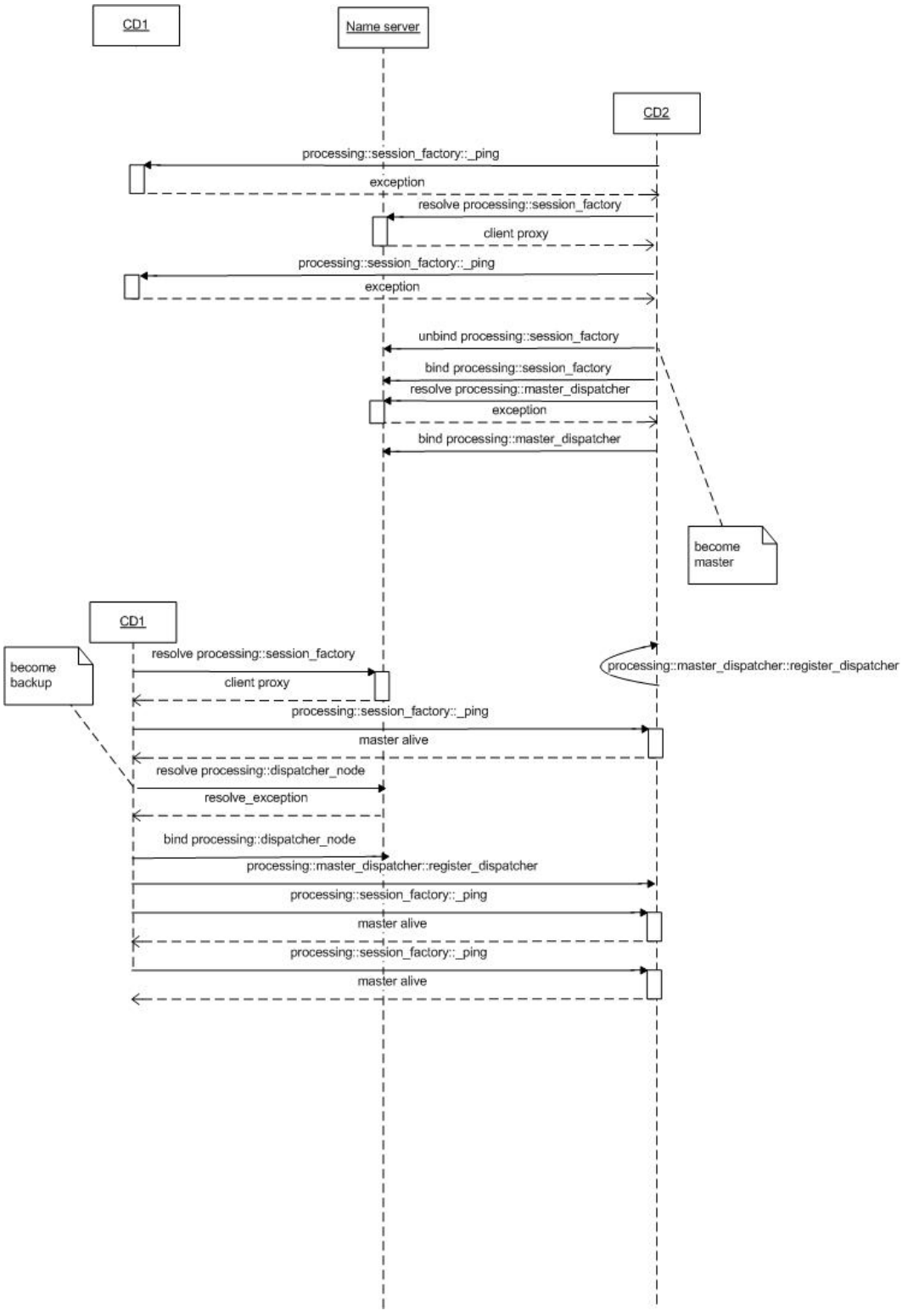

Figure 3: Determining the master node and the backup node (part 2)

During initialization, the content distributor named CD1 tries to resolve the **processing::session_factory** server object with the name server. Because no master content distributor is registered with the **processing::session_factory** interface, the **resolve** call returns a **resolve_exception**, as specified in in [MS-FSMW] section 3,to CD1. CD1 elects itself as master content distributor, and then it calls **bind** and activates the **processing::session_factory** and the **processing::master_dispatcher** server objects with the name server. Then, CD1 activates the **processing::dispatcher_node** server object and registers it by invoking the **processing::master_dispatcher::register_dispatcher** method.

When the content distributor named CD2 performs its initialization, it sends a **resolve** request for the **processing::session_factory** interface to the name server to locate the master content distributor. It receives the **client proxy** for the **processing::session_factory** server object that is registered and served by CD1. CD2 pings the client proxy; because CD1 responds, CD2 elects itself as the backup content distributor. CD2 activates the **processing::dispatcher_node** server object and calls **processing::master_dispatcher::register_dispatcher**, as specified in section 3.1.4.1, to register as the backup content distributor with CD1.

CD2 continues to invoke **__ping** at CD1 to verify that it responds. When CD1 ceases to respond, the **__ping** method returns an exception. CD2 invokes **__ping** two more times to verify that C1 does not respond. When the third **__ping** invocation returns an exception, CD2 resolves the **processing::session_factory** interface with the name server and invokes **__ping** on the returned client proxy in case another backup content distributor registered as master in the meantime. Because this fourth **__ping** invocation also returns an exception, no master content distributor is available. CD2 therefore elects itself as master content distributor. It unbinds the **processing::session_factory** server object of CD1, and it binds and activates the **processing::session_factory** server object and then the **processing::master_dispatcher** server object with the name server. CD2 activates the **processing::dispatcher_node** server object and registers it with **processing::master_dispatcher::register_dispatcher**.

When CD1 later responds again, it performs the same steps as CD2. It finds CD2 registered as master content distributor and that it responds. Therefore, CD1 registers as the backup content distributor with CD2. CD1 then starts to monitor CD2 by using **__ping** invocations to verify that it is available.

3.1 processing::master_dispatcher Server Details

The content distributor that is elected as master content distributor performs the role of protocol server for the **processing::master_dispatcher** interface. It enables backup content distributors to register with the master content distributor.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The protocol server maintains the following state:

dispatcher node: A state that contains **processing::dispatcher_node** client proxies that are associated with registered content distributors, which enables the master content distributor to use the **processing::dispatcher_node** interface to communicate with the registered content distributors.

3.1.2 Timers

None.

3.1.3 Initialization

The master content distributor MUST use the **bind** method to register a **processing::master_dispatcher** server object with the name server, as specified in [MS-FSMW] section 3.4.4.2. The parameters for the **bind** method are encapsulated in an **abstract object reference (AOR)**, as specified in [MS-FSMW] section 2.2.18.

name: A string that MUST contain the value "esp/subsystems/processing/dispatcher".

object_id: The value is implementation specific of the higher level application.

host: A string that contains the **host name** of the server object on the protocol server. The value is implementation specific—that is, determined by the higher-level application.

port: The **base port** + 390.

interface_type: A string that MUST contain the value "processing::master_dispatcher".

interface_version: A string that MUST contain the value "5.0".

3.1.4 Message Processing Events and Sequencing Rules

The **processing::master_dispatcher** interface specifies the methods that are listed in the following table.

Method	Description
register_dispatcher	Registers a content distributor with the master content distributor.

3.1.4.1 processing::master_dispatcher::register_dispatcher

The **register_dispatcher** method registers a content distributor with the master content distributor. The method is specified by the following FSIDL specification:

```
void register_dispatcher(in long node_id);
```

node_id: Identifier of the content distributor.

Return value: None.

Exceptions raised: No exceptions are raised beyond those raised by the underlying Middleware Protocol, as specified in [MS-FSMW].

The master content distributor MUST use the **node_id** input value to resolve the **processing::dispatcher_node** client proxy, by using the **resolve** method as specified in [MS-FSMW] section 3.4.4.1. The parameters for the **resolve** method are as follows:

name: A string that MUST contain the value "esp/subsystem/processing/dispatcher/*node_id*", where *node_id* is the **node_id** input parameter. For example, if the **node_id** input value is "1", the string MUST be "esp/subsystem/processing/dispatcher/1".

interface_type: A string that MUST contain the value "processing::dispatcher_node".

interface_version: A string that MUST contain the value "5.1".

The **processing::dispatcher_node** client proxy that is returned from the **resolve** method MUST be stored in the **dispatcher_node** state.

The master content distributor and the backup content distributors MUST call the **processing::dispatcher::register_dispatcher** method.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 processing::master_dispatcher Client Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

The master content distributor and the backup content distributors use the **resolve** method to retrieve the client proxy to the **master_dispatcher** server object that is bound in the name server, as specified in [MS-FSMW] section 3.4.4.1. The parameters for the **resolve** method are:

name: A string that MUST contain the value "esp/subsystems/processing/dispatcher".

interface_type: A string that MUST contain the value "processing::master_dispatcher".

interface_version: A string that MUST contain the value "5.0".

3.2.4 Message Processing Events and Sequencing Rules

The **processing::master_dispatcher** interface specifies the methods that are listed in the following table.

Method	Description
register_dispatcher	Registers a content distributor with the master content distributor.

3.2.4.1 processing::master_dispatcher::register_dispatcher

The **register_dispatcher** method is specified in section 3.1.4.1. The master content distributor and the backup content distributors invoke the **register_dispatcher** method during initialization. The **node_id** input parameter MUST be the value of the **node_id** state, as specified in section 3.5.1.

3.2.5 Timer Events

3.2.6 Other Local Events

None.

3.3 processing::session_factory Server Details

The content distributor that is elected to be the master content distributor MUST register the **processing::session_factory** interface with the name server. This registration enables the backup content distributors to monitor the master content distributor.

3.3.1 Abstract Data Model

None.

3.3.2 Timers

The **check master status** timer sends **resolve** method requests to the **processing::session_factory** server object at regular intervals to verify whether the master content distributor is still registered with the name server. The default interval is 5 seconds.

3.3.3 Initialization

The master content distributor uses the **bind** method to register a **processing::session_factory** server object with the name server, as specified in [MS-FSMW] section 3.4.4.2.

The parameters for the **bind** method are encapsulated in an AOR, as specified in [MS-FSMW] section 2.2.18.

name: A string that MUST contain the value "esp/subsystems/processing/dispatcher".

object_id: The value is implementation specific of the higher level application.

host: A string that contains the host name of the server object on the protocol server. The value is implementation specific—that is, determined by the higher level application.

port: The base port + 390.

interface_type: A string that MUST contain the value "processing::session_factory".

interface_version: A string that MUST contain the value "5.1".

3.3.4 Message Processing Events and Sequencing Rules

The protocol uses only the **___ping** method of the **processing::session_factory** interface, as specified in [MS-FSMW] section 3.2.4.2. [MS-FSCF] section 3 specifies the other methods in the **processing::session_factory** interface.

3.3.5 Timer Events

The **check master status** timer triggers the protocol server to verify whether the protocol server is still registered as the master content distributor with the name server. The protocol server MUST invoke the **resolve** method of the **processing::session_factory** interface, as specified in section 3.4.3, to locate the current master content distributor. If the **processing::session_factory** server object of the protocol server is no longer registered with the name server, the protocol server MUST

invoke the **processing::master_dispatcher::register_dispatcher** method, as specified in section 3.2.4.1, to become a backup content distributor and register with the new master content distributor.

3.3.6 Other Local Events

None.

3.4 processing::session_factory Client Details

The backup content distributor MUST use **processing::session_factory** interface to determine whether the master content distributor is active, as specified in section 3.4.4.

3.4.1 Abstract Data Model

None.

3.4.2 Timers

The **check for master** timer sends **__ping** method requests to the **processing::session_factory** server object at regular intervals to locate the current master content distributor for the backup content distributor. The interval is implementation specific; the default interval is 5 seconds.

3.4.3 Initialization

The backup content distributor MUST use the **resolve** method to retrieve the client proxy that is associated with the **processing::session_factory** server object with the name server, as specified in [MS-FSMW] section 3.4.4.1.

The parameters for the **resolve** method are:

name: A string that MUST contain the value "esp/subsystems/processing/dispatcher".

interface_type: A string that MUST contain the value "processing::session_factory".

interface_version: A string that MUST contain the value "5.1".

If the **resolve** method returns **resolve_exception**, as specified in [MS-FSMW] section 2.2.21, a master content distributor is not available. The backup content distributor MUST register as the master content distributor, as specified in section 3.2.4.1. If another backup content distributor successfully registers as a master content distributor first, this backup content distributor MUST register as backup with the new master content distributor.

3.4.4 Message Processing Events and Sequencing Rules

The backup content distributor MUST use the **__ping** method of the **processing::session_factory** interface, as specified in [MS-FSMW] section 3.2.4.2, to determine whether the master content distributor is active.

3.4.5 Timer Events

The **check for master** timer triggers the backup content distributor to verify whether an active master content distributor exists, as specified in section 3.4. The backup content distributor performs this by invoking the **__ping** method of the **processing::session_factory** interface, as specified in [MS-FSMW] section 3.2.4.2. If the **__ping** method fails three successive times, the

backup content distributor MUST register as the master content distributor. It MUST also invoke the **unbind** method, as specified in [MS-FSMW] section 3, to remove the **processing::session_factory** interface of the previous master from the name server. The parameters for the **unbind** method are:

name: A string that MUST contain the value "esp/subsystems/processing/dispatcher".

interface_type: A string that MUST contain the value "processing::session_factory".

interface_version: A string that MUST contain the value "5.1".

After calling the **unbind** method, the backup content distributor MUST register as the master content distributor, as specified in sections 3.2.3 and 3.1.3.

3.4.6 Other Local Events

None.

3.5 processing::dispatcher_node Server Details

The master content distributor and the backup content distributors perform the role of protocol server for the **processing::dispatcher_node** interface. This role enables the content distributors to receive method invocations from the master content distributor for creation, recreation, and closing of session server objects.

3.5.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

A content distributor maintains the following states:

node_id: An integer that represents a unique instance identifier for this content distributor instance.

collection holder: A collection of strings that represents content collection names. The protocol server MUST use the **collection holder** collection to maintain available content collections.

load: A field of type **long** that represents the state of the current work load of the protocol server. The protocol server MUST update this state when it receives a **processing::session::process** method invocation, as specified in [MS-FSCF] section 3.4.4.5.

session holder: A state that contains a set of **processing::session** server objects, where each server object is referenced by a session identifier. The protocol server maintains states for each **processing::session** server object in the **session holder** state, as specified in the following table.

State	Description
session id	An integer state that contains the **processing::session** server object identifier.
content collection id	A string state that contains the content collection name of the **processing::session** server object.

3.5.2 Timers

None.

3.5.3 Initialization

All content distributors MUST use the **bind** method to register a **processing::dispatcher_node** server object with the name server, as specified in [MS-FSMW] section 3.4.4.2.

The parameters for the **bind** method are encapsulated in an AOR, as specified in [MS-FSMW] section 2.2.18.

name: A string that MUST contain the value "esp/subsystems/processing/dispatcher/*node_id*", where *node_id* is the **node_id** state.

object_id: The value is implementation specific of the higher level application.

host: A string that contains the host name of the server object on the protocol server. The value is implementation specific—that is, determined by the higher-level application.

port: The base port + 390 or the base port + 391, as specified in section 3.5.4.

interface_type: A string that MUST contain the value "processing::dispatcher_node".

interface_version: A string that MUST contain the value "5.1".

All content distributors MUST invoke the **processing::master_dispatcher::register_dispatcher** method to register with the master content distributor, as specified in section 3.2.4.1.

3.5.4 Message Processing Events and Sequencing Rules

The **processing::dispatcher_node** interface specifies the methods that are listed in the following table.

Method	Description
create_session	Creates a **processing::session** server object identified by a session identifier, and returns a **processing::session** client proxy.
create	Creates a **processing::session** server object identified by a session identifier, and returns a **processing::session** client proxy.
recreate	Re-creates a session with a identifier, and returns a **processing::session** client proxy.
destroy_session	Closes and removes session with a given identifier.
remove_sessions	Removes sessions for a specified collection.
get_session	Returns all active session identifiers.
get_load	Returns work load.

3.5.4.1 processing::dispatcher_node::create_session

This method creates a feeding session and returns a new **processing::session** client proxy. The method is specified by the following FSIDL specification:

```
processing::session create_session(
        in long id,
        in string collection,
        in coreprocessing::operation_callback callback,
        in cht::core::guarantee_set guarantees,
        in long priority)
        raises (core::unsupported_guarantee_set,
                unknown_collection);
```

id: The identifier of the session.

collection: A string that contains the name of the content collection for which to create the session.

callback: A **coreprocessing::operation_callback** client proxy. The **coreprocessing::operation_callback** interface is specified in [MS-FSDP] section 3.4.

guarantees: The **guarantees** attribute of the **guarantees** input value. It MUST contain either one **cht::core::feeding_priority** Cheetah entity that specifies priority for this feeding session or an empty collection.

priority: A **long** value that MUST be 0.

Return value: A **processing::session** client proxy that is instantiated with the AOR specified in [MS-FSCF] section 3.4.

Exceptions raised:

processing::unknown_collection: MUST be raised if the specified content collection is unknown.

core::unsupported_guarantee_set: MUST be raised if the protocol server cannot create the feeding session.

When the protocol server receives a **create_session** method invocation, it MUST create and return a new **processing::session** client proxy to the protocol client. The client proxy is instantiated with the AOR specified in [MS-FSCF] section 3.4.

The protocol server MUST activate the new **processing::session** server object on base port + 390.

The protocol server MUST verify that the **collection** input value exists in the **collection holder** state. If a **collection** input value does not exist, the protocol server MUST raise a **processing::unknown_collection** exception.

When the protocol client invokes the **processing::dispatcher_node::create_session** method, the protocol server MUST invoke the **coreprocessing::session_factory::create** method in the **indexing dispatcher**, as specified in [MS-FSDP] section 3.2.4.1.

The protocol server MUST store the **processing::session** server object in a **session holder** state with the **session id** input value as the unique key.

3.5.4.2 processing::dispatcher_node::create

This method creates a feeding session and returns a new **processing::session** client proxy. The method is specified by the following FSIDL specification:

```
processing::session create(
        in long id,
```

```
        in string collection,
        in cht::core::guarantee_set guarantees,
        in long priority,
        in boolean external_client)
        raises (core::unsupported_guarantee_set,
                unknown_collection);
```

id: The identifier of the session.

collection: A string that contains the name of the content collection for which to create the session.

guarantees: The **guarantees** attribute of the **guarantees** input value. It MUST contain either one **cht::core::feeding_priority** Cheetah entity that specifies priority for this feeding session or an empty collection.

priority: A long value that MUST be 0.

external_client: A boolean value that specifies where to activate the **processing::session** server object.

Return value: A **processing::session** client proxy that is instantiated with the AOR specified in [MS-FSCF] section 3.4.

Exceptions raised:

processing::unknown_collection: MUST be raised if the specified content collection is unknown.

core::unsupported_guarantee_set: MUST be raised if the protocol server cannot create the feeding session.

When the protocol server receives a **create** method invocation, it MUST create and return a new **processing::session** client proxy to the protocol client and activate the new **processing::session** server object. The client proxy that is returned MUST be instantiated with the AOR specified in [MS-FSCF] section 3.4.

If the **external_client** input value is false, the **processing::session** server object MUST be activated on base port + 390. Otherwise, it MUST be activated on base port + 391.

The protocol server MUST verify that the **collection** input value exists in the **collection holder** state. If it does not exist, the protocol server raises a **processing::unknown_collection** exception.

When the protocol client invokes the **processing::dispatcher_node::create** method, the protocol server MUST invoke the **coreprocessing::session_factory::create** method in the indexing dispatcher, as specified in [MS-FSDP] section 3.2.4.1.

The protocol server MUST store the **processing::session** server object in a **session holder** state with the **session id** input value as the unique key.

3.5.4.3 processing::dispatcher_node::recreate

This method recreates a feeding session with a specified identifier. The method is specified by the following FSIDL specification:

```
processing::session recreate(
        in long id,
```

```
in string collection,
in cht::core::guarantee_set guarantees,
in long priority,
in boolean external_client)
raises (core::unsupported_guarantee_set,
        unknown_collection);
```

id: Identifier for the session that was already created.

collection: A string that contains the name of the content collection for which to create the session.

guarantees: The **guarantees** attribute of the **guarantees** input value. It MUST contain the **cht::core::feeding_priority** Cheetah entity that specifies priority for this feeding session, or it MUST contain an empty collection.

priority: A long value that MUST be 0.

external_client: A boolean value that specifies where to activate the **processing::session** server object.

Return value: A **processing::session** client proxy that is instantiated with the AOR specified in [MS-FSCF] section 3.4.

Exceptions raised:

processing::unknown_collection: MUST be raised if the specified content collection is unknown.

core::unsupported_guarantee_set: MUST be raised if the protocol server cannot create the feeding session.

When the protocol server receives a **recreate** method invocation, the protocol server MUST validate the **session holder** state. If the **session holder** state contains a **processing::session** server object with the specified **session id**, the protocol server returns a client proxy to the existing **processing::session** server object. If the session withthe specified **session id** does not exist, the protocol server MUST create and return a new **processing::session** client proxy to the protocol client and then activate the new **processing::session** server object. The client proxy is instantiated with the AOR specified in [MS-FSCF] section 3.4.

If the **external_client** input value is false, the **processing::session** server object MUST be activated on base port + 390. Otherwise, it MUST be activated on base port + 391.

The protocol server MUST verify whether the **collection** input value exists in the **collection holder** state. If it does not exist, the protocol server raises the **processing::unknown_collection** exception.

When the protocol client invokes the **processing::dispatcher_node::recreate** method, the protocol server MUST invoke the **coreprocessing::session_factory::recreate** method in the indexing dispatcher, as specified in [MS-FSDP] section 3.2.4.2.

The protocol server MUST store the **processing::session** server object in a **session holder** state with the **session id** input value as the unique key.

3.5.4.4 processing::dispatcher_node::destroy_session

This method closes a session with a specified identifier. The method is specified by the following FSIDL specification:

```
void destroy_session(in long id)
```

id: The identifier of the **processing::session** server object to close. It MUST be an integer that is greater than or equal to zero.

Return value: None.

Exceptions raised: No exceptions are raised beyond those raised by the underlying Middleware Protocol, as specified in [MS-FSMW].

The protocol server MUST remove the **processing::session** server object with the specified **session id** from the **session holder** state.

When the protocol client invokes the **processing::dispatcher_node::destroy_session** method, the protocol server MUST invoke the **coreprocessing::session_factory::close** method in the indexing dispatcher, as specified in [MS-FSDP] section 3.2.4.3.

3.5.4.5 processing::dispatcher_node::remove_sessions

This method closes all sessions that use the specified collection. The method is specified by the following FSIDL specification:

```
sequence<long> remove_sessions(in string coll);
```

coll: The content collection identifier for which to remove all sessions.

Return value: A sequence of **long** values that represents session identifiers of sessions that were removed.

Exceptions raised: No exceptions are raised beyond those raised by the underlying Middleware Protocol, as specified in [MS-FSMW].

The protocol server MUST delete all **processing::session** server objects that are associated with the specified content collection identifier from the **session holder** state.

3.5.4.6 processing::dispatcher_node::get_sessions

This method returns the session identifiers of the **processing::session** server objects in the protocol server. The method is specified by the following FSIDL specification:

```
sequence<long> get_sessions();
```

Return value: A sequence of **long** values that represents session identifiers.

Exceptions raised: No exceptions are raised beyond those raised by the underlying Middleware Protocol, as specified in [MS-FSMW].

The protocol server MUST return the **session id** state that is associated with each **processing::session** server object that is contained in the **session_holder** state.

3.5.4.7 processing::dispatcher_node::get_load

This method returns the load of the protocol server. The method is specified by the following FSIDL specification:

```
long get_load();
```

Return value: A long value that represents the load of the protocol server.

Exceptions raised: No exceptions are raised beyond those raised by the underlying Middleware Protocol, as specified in [MS-FSMW].

The protocol server MUST return the **load** state.

3.5.5 Timer Events

None.

3.5.6 Other Local Events

None.

3.6 processing::dispatcher_node Client Details

The master content distributor MUST forward requests to create, recreate, and close sessions to content distributors on the **processing::dispatcher_node** interface.

3.6.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The master content distributor maintains the **dispatcher node** state, as specified in section 3.1.1.

3.6.2 Timers

None.

3.6.3 Initialization

The master content distributor uses the client proxy references that are associated with the **dispatcher node** state to call the content distributor protocol servers. For more information, see section 3.1.4.1.

3.6.4 Message Processing Events and Sequencing Rules

None.

3.6.5 Timer Events

None.

3.6.6 Other Local Events

None.

4 Protocol Examples

4.1 Registering a Backup Content Distributor

This example describes how to use the **processing::master_dispatcher::register_dispatcher** method of the **processing::master_dispatcher** interface, as described in section 3.1.4.1, so that a content distributor can register as a backup content distributor with a master content distributor.

First, the master distributor, as the protocol server, creates a server object that implements the **processing::master_dispatcher** interface, and registers that server object with the name server. The backup content distributor, as the protocol client, resolves the server object with the name server to acquire the client proxy for that **processing::master_dispatcher** interface. This is possible because the protocol client and protocol server are both aware of the location of the shared name server and the symbolic name of the server object.

The protocol client is now ready to invoke the **processing::master_dispatcher::register_backup** method on the **processing::master_dispatcher** client proxy.

4.2 Sample Code

4.2.1 Protocol Server Initialization

```
SET server_object_instance TO
INSTANCE OF processing::master_dispatcher SERVER OBJECT

SET server_object_host TO "www.cohowinery.com"

SET server_object_port TO "1234"

SET server_object_interface_type TO "processing::master_dispatcher"

SET server_object_interface_version TO "5.0"

SET server_object_name TO "esp/subsystems/processing/dispatcher"

SET server_object_aor TO server_object_host, server_object_port,
server_object_interface_type, server_object_interface_version AND server_object_name

CALL nameserver.bind WITH server_object_name AND server_object_aor
```

4.2.2 Protocol Client Initialization

```
SET server_object_name TO "esp/subsystems/processing/dispatcher "

SET server_object_type TO " processing::master_dispatcher"

SET server_object_version TO "5.0"

CALL nameserver.resolve WITH server_object_name, server_object_type AND server_object_version
RETURNING master_dispatcher_client_proxy
```

4.2.3 Protocol Client Message

```
SET hostname TO "myclient.mydomain.com"
```

```
CALL master_dispatcher_client_proxy.register_backup WITH hostname AND node_id
```

4.2.4 Server Response

```
SET server_object_name TO "esp/subsystems/processing/dispatcher/" + node_id

SET server_object_type TO " processing::dispatcher_node"
SET server_object_version TO "5.1"

CALL nameserver.resolve WITH server_object_name, server_object_type AND server_object_version
RETURNING dispatcher_node_client_proxy

ADD dispatcher_node_client_proxy TO dispatcher_node_state
```

5 Security

5.1 Security Considerations for Implementers

Security is resolved in the Middleware Protocol, as described in [MS-FSMW].

5.2 Index of Security Parameters

None.

6 Appendix A: Full FSIDL

For ease of implementation, the full FSIDL is provided here.

```
  module interfaces {
  module core {

    exception unsupported_guarantee_set {
      string what;
    };
  };

  module processing {

    exception unknown_collection {
      string message;
    };

    typedef sequence<long> session_list;

    interface master_dispatcher {
#     pragma version master_dispatcher 5.0

     void register_dispatcher(in long node_id);

    };

    interface dispatcher_node {
#     pragma version dispatcher_node 5.1

       session create_session(
        in long id,
        in string collection,
        in coreprocessing::operation_callback callback,
        in cht::core::guarantee_set guarantees,
        in long priority)
        raises (core::unsupported_guarantee_set,
                unknown_collection);

      session create(
        in long id,
        in string collection,
        in cht::core::guarantee_set guarantees,
        in long priority,
        in boolean external_client)
        raises (core::unsupported_guarantee_set,
                unknown_collection);

      session recreate(
        in long id,
        in string collection,
        in cht::core::guarantee_set guarantees,
        in long priority,
        in boolean external_client)
        raises (core::unsupported_guarantee_set,
                unknown_collection);

      void destroy_session(in long id);
```

```
        sequence<long> remove_sessions(in string coll);

        sequence<long> get_sessions();

        long get_load();
      };

    interface session_factory {
      void __ping();
    }
    };
  };
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® FAST™ Search Server 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

A

Abstract data model
  client (section 3.2.1 15, section 3.4.1 17, section 3.6.1 24)
  server (section 3.1.1 13, section 3.3.1 16, section 3.5.1 18)
Applicability 7

C

Capability negotiation 7
Change tracking 32
cht
    core
        feeding_priority data type 8
        guarantee data type 8
        guarantee_set data type 8
Client
  abstract data model (section 3.2.1 15, section 3.4.1 17, section 3.6.1 24)
  initialization (section 3.2.3 15, section 3.4.3 17, section 3.6.3 24)
  local events (section 3.2.6 16, section 3.4.6 18, section 3.6.6 25)
  message processing (section 3.2.4 15, section 3.4.4 17, section 3.6.4 24)
  overview (section 3 10, section 3.4 17, section 3.6 24)
  processing::dispatcher_node interface 24
  processing::master_dispatcher::register_dispatcher method 15
  processing::session_factory interface 17
  sequencing rules (section 3.2.4 15, section 3.4.4 17, section 3.6.4 24)
  timer events (section 3.4.5 17, section 3.6.5 24)
  timers (section 3.2.2 15, section 3.4.2 17, section 3.6.2 24)
Common data types 8
core
    unsupported_guarantee_set data type 9

D

Data model - abstract
  client (section 3.2.1 15, section 3.4.1 17, section 3.6.1 24)
  server (section 3.1.1 13, section 3.3.1 16, section 3.5.1 18)
Data types
  cht
      core
          feeding_priority 8
          guarantee 8
          guarantee_set 8
  common - overview 8
  core
      unsupported_guarantee_set 9
  processing
      unknown_collection 9

E

Events
  local - client (section 3.2.6 16, section 3.4.6 18, section 3.6.6 25)
  local - server (section 3.1.6 15, section 3.3.6 17, section 3.5.6 24)
  timer - client (section 3.4.5 17, section 3.6.5 24)
  timer - server (section 3.1.5 15, section 3.3.5 16, section 3.5.5 24)
Examples
  registering a backup content distributor 26

F

Fields - vendor-extensible 7
FSIDL 29
Full FSIDL 29

G

Glossary 5

I

Implementer - security considerations 28
Index of security parameters 28
Informative references 6
Initialization
  client (section 3.2.3 15, section 3.4.3 17, section 3.6.3 24)
  server (section 3.1.3 14, section 3.3.3 16, section 3.5.3 19)
Interfaces - client
  processing::dispatcher_node 24
  processing::session_factory 17
Interfaces - server
  processing::dispatcher_node 18
  processing::master_dispatcher 13
  processing::session_factory 16
Introduction 5

L

Local events
  client (section 3.2.6 16, section 3.4.6 18, section 3.6.6 25)
  server (section 3.1.6 15, section 3.3.6 17, section 3.5.6 24)

M

Message processing
client (section 3.2.4 15, section 3.4.4 17, section 3.6.4 24)
server (section 3.1.4 14, section 3.3.4 16, section 3.5.4 19)
Messages
cht
core
feeding_priority data type 8
guarantee data type 8
guarantee_set data type 8
common data types 8
core
unsupported_guarantee_set data type 9
processing
unknown_collection data type 9
transport 8
Methods
processing::dispatcher_node::create 20
processing::dispatcher_node::create_session 19
processing::dispatcher_node::destroy_session 22
processing::dispatcher_node::get_load 24
processing::dispatcher_node::get_sessions 23
processing::dispatcher_node::recreate 21
processing::dispatcher_node::remove_sessions 23
processing::master_dispatcher::register_dispatcher (section 3.1.4.1 14, section 3.2.4.1 15)

N

Normative references 5

O

Overview (synopsis) 6

P

Parameters - security index 28
Preconditions 6
Prerequisites 6
processing
unknown_collection data type 9
processing::dispatcher_node interface (section 3.5 18, section 3.6 24)
processing::dispatcher_node::create method 20
processing::dispatcher_node::create_session method 19
processing::dispatcher_node::destroy_session method 22
processing::dispatcher_node::get_load method 24
processing::dispatcher_node::get_sessions method 23
processing::dispatcher_node::recreate method 21
processing::dispatcher_node::remove_sessions method 23
processing::master_dispatcher interface 13
processing::master_dispatcher::register_dispatcher method (section 3.1.4.1 14, section 3.2.4.1 15)
processing::session_factory interface (section 3.3 16, section 3.4 17)
Product behavior 31

R

References
informative 6
normative 5
Registering a backup content distributor example 26
Relationship to other protocols 6

S

Security
implementer considerations 28
parameter index 28
Sequencing rules
client (section 3.2.4 15, section 3.4.4 17, section 3.6.4 24)
server (section 3.1.4 14, section 3.3.4 16, section 3.5.4 19)
Server
abstract data model (section 3.1.1 13, section 3.3.1 16, section 3.5.1 18)
initialization (section 3.1.3 14, section 3.3.3 16, section 3.5.3 19)
local events (section 3.1.6 15, section 3.3.6 17, section 3.5.6 24)
message processing (section 3.1.4 14, section 3.3.4 16, section 3.5.4 19)
overview (section 3 10, section 3.1 13, section 3.3 16, section 3.5 18)
processing::dispatcher_node interface 18
processing::dispatcher_node::create method 20
processing::dispatcher_node::create_session method 19
processing::dispatcher_node::destroy_session method 22
processing::dispatcher_node::get_load method 24
processing::dispatcher_node::get_sessions method 23
processing::dispatcher_node::recreate method 21
processing::dispatcher_node::remove_sessions method 23
processing::master_dispatcher interface 13
processing::master_dispatcher::register_dispatcher method 14
processing::session_factory interface 16
sequencing rules (section 3.1.4 14, section 3.3.4 16, section 3.5.4 19)
timer events (section 3.1.5 15, section 3.3.5 16, section 3.5.5 24)

timers (section 3.1.2 14, section 3.3.2 16, section 3.5.2 19)
Standards assignments 7

T

Timer events
client (section 3.4.5 17, section 3.6.5 24)
server (section 3.1.5 15, section 3.3.5 16, section 3.5.5 24)
Timers
client (section 3.2.2 15, section 3.4.2 17, section 3.6.2 24)
server (section 3.1.2 14, section 3.3.2 16, section 3.5.2 19)
Tracking changes 32
Transport 8

V

Vendor-extensible fields 7
Versioning 7