[MS-FSIDFT]:
Indexing Dispatcher 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.
03/18/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Indexing Dispatcher Fault Tolerance protocol used between multiple **indexing dispatchers**, so they can agree which one becomes the master indexing dispatcher, and which ones become backup indexing dispatchers. The use of multiple indexing dispatcher nodes increases performance and robustness; should the master indexing dispatcher become unavailable, one of the backup indexing dispatchers becomes the new master indexing dispatcher.

1.1 Glossary

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

fully qualified domain name (FQDN)
Hypertext Transfer Protocol (HTTP)

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

abstract object reference (AOR)
base port
callback message
Cheetah
Cheetah checksum
client proxy
content collection
FAST Search Interface Definition Language (FSIDL)
host name
HTTP POST
index column
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-FSCHT] Microsoft Corporation, "Cheetah Data Structure"

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

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

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

1.2.2 Informative References

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

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

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

1.3 Protocol Overview

One or more indexing dispatchers are part of an extended session-based **item** feeding chain, wherein a session is established between a feeding client and an **indexing node**. Operations containing information about items to add, update, or remove from the index are sent using the established session, and asynchronous status information **callback messages** about the items are sent from the indexing service back to the feeding client.

In a setup with multiple **index columns** and indexing dispatchers, the sessions used in the feeding chain are distributed across all the indexing dispatchers, improving performance and robustness. When the feeding client requests that a new session be created, recreated, or closed, the master indexing dispatcher will either execute the request itself, or forward the request to one of the backup indexing dispatchers, in this way distributing the sessions across the available indexing dispatchers. An interface to the created session is returned to the feeding client, and the feeding client is therefore not aware, nor does it have to be, if a specific session used for feeding is routed using a master or backup indexing dispatcher. The concept of master and backup indexing dispatcher is therefore only relevant in the task of creating a session, and not in the actual use of the session.

This protocol enables the indexing dispatchers to agree on which one becomes the master indexing dispatcher, and which ones become backup indexing dispatchers, registering themselves as backup indexing dispatchers with the master indexing dispatcher. Each backup indexing dispatcher continues to monitor the master indexing dispatcher. If the master indexing dispatcher becomes unavailable, one of the backup indexing dispatchers will take on the role as master indexing dispatcher.

For an overview of the system of which this protocol is a part, please see [MS-FSO].

1.4 Relationship to Other Protocols

This protocol uses Middleware, an **HTTP** based protocol, as described in [MS-FSMW].

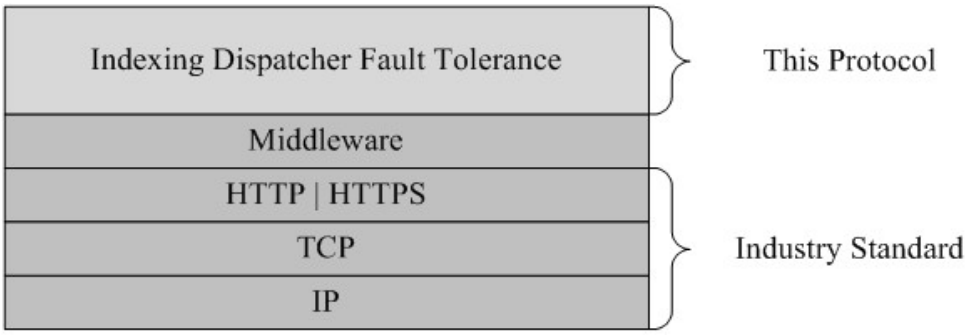

Figure 1: This protocol in relation to other protocols

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 is designed to enable indexing dispatchers to determine which one is to become master indexing dispatcher and which ones are to become backup indexing dispatchers. The indexing dispatchers are part of an extended session-based item feeding chain.

1.7 Versioning and Capability Negotiation

Capability Negotiation: The Middleware protocol is connectionless, but the correct interface version is to be specified in every message passed using the Middleware protocol. See section 3.1.3 for the specific version number.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The messages in this protocol MUST be sent as **HTTP POST** messages, as specified in [MS-FSMW], the Middleware protocol.

2.2 Common Data Types

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 checksum** MUST be an integer with the value -1479218033. The type identifier for the Cheetah entities MUST be integers as specified in the following table.

Cheetah entity	type identifier
cht::core::guarantee	3
cht::core::feeding_priority	7
cht::core::guarantee_set	9

The full FSIDL for this protocol is provided in section 6.1. The complete listing of Cheetah entities used for this protocol is provided in section 6.2.

2.2.1 coreprocessing::unknown_collection_error

```
exception unknown_collection_error {
};
```

The **unknown_collection_error** exception states that the **content collection** is unknown. It has no members.

2.2.2 core::unsupported_guarantee_set

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

The **unsupported_guarantee_set** exception states that the backup indexing dispatcher is unable to create or recreate a session. It has the following member:

what: A string holding verbose information about the cause of the exception. The content of the string is implementation specific of the higher level application.

2.2.3 cht::core::guarantee

```
entity guarantee {
};
```

The **guarantee** Cheetah entity is a parent class for the **feeding_priority** Cheetah entity, as specified in section 2.2.4.

2.2.4 cht::core::feeding_priority

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

The **feeding_priority** Cheetah entity is a subclass of the **guarantee** Cheetah entity specified in section 2.2.3. It specifies the priority when feeding items, and has the following member:

priority: An integer that specifies the feeding priority. This MUST be equal to or greater than zero, where zero represents the highest priority.

2.2.5 cht::core::guarantee_set

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

The **guarantee_set** Cheetah entity contains a collection of **guarantee** Cheetah entities, as specified in section 2.2.3. It has the following member:

guarantees: A collection of **guarantee** Cheetah entities, as specified in section 2.2.3.

3 Protocol Details

This document defines a protocol used between two or more indexing dispatcher nodes, where there is one master indexing dispatcher node and one or more backup indexing dispatcher nodes. This protocol consists of two interfaces.

The master dispatcher interface enables communication between a backup indexing dispatcher node and the master indexing dispatcher node. It is used to elect master and backup indexing dispatchers, and allows backup indexing dispatchers to register with the master.

The session factory interface enables communication between the master indexing dispatcher node and a backup indexing dispatcher node, and is used by the master indexing dispatcher to forward incoming session requests to backup indexing dispatchers.

The role as protocol client and protocol server therefore depends on the interface used and the role of the indexing dispatcher. This is shown in the following figure.

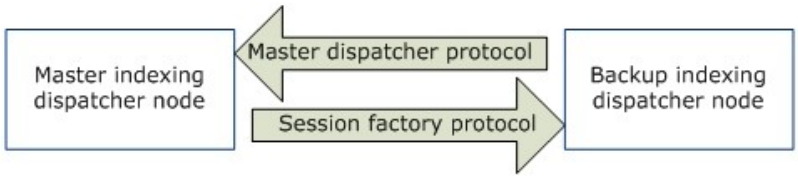

Figure 2: Interfaces between master and backup indexing dispatcher nodes

At initial startup, an indexing dispatcher node has not taken the role as either master or backup. To determine the role, each indexing dispatcher MUST look for a registered master indexing dispatcher. If no master indexing dispatcher is found to be alive, the indexing dispatcher MUST become master indexing dispatcher. If a master indexing dispatcher is found, the indexing dispatcher MUST become a backup indexing dispatcher, registering itself as a backup indexing dispatcher with the master indexing dispatcher. A backup indexing dispatcher MUST monitor the master indexing dispatcher, and attempt to take the role as master indexing dispatcher if the master indexing dispatcher is no longer available.

To become a master indexing dispatcher, the indexing dispatcher MUST use the **bind** method, as specified in section 3.1.3, to register and activate the **master_dispatcher** interface in the name server.

To look for a master indexing dispatcher, an indexing dispatcher MUST use the **resolve** method, as specified in section 3.2.3, to determine whether or not a master indexing dispatcher has registered the **master_dispatcher** interface in the name server.

To verify that a registered master indexing dispatcher is active, the indexing dispatcher MUST use the **__ping** method on the **master_dispatcher** interface, as specified in [MS-FSMW] section 3.2.

To monitor the master indexing dispatcher, the backup indexing dispatcher MUST use the **__ping** method on the **master_dispatcher** interface, as specified in [MS-FSMW] section 3.2.

The sequence and number of times the **bind**, **resolve**, and **__ping** methods are used to ensure that only one indexing dispatcher registers as master indexing dispatcher and that the backup indexing dispatchers monitor the master indexing dispatcher is implementation specific of the higher level application.

A possible sequence diagram is shown as an example in the following figure wherein the indexing dispatcher ID1 during initialization tries to resolve the **master_dispatcher** interface in the name

server to look for a master indexing dispatcher. Because no master indexing dispatcher is yet registered, the **resolve** call returns a **resolve_exception**, and ID1 elects itself as master indexing dispatcher and binds and activates the **master_dispatcher** interface in the name server.

When the indexing dispatcher ID2 starts up, it tries to **resolve** the **master_dispatcher** interface in the name server to look for a master indexing dispatcher. It receives the **client proxy** to **master_dispatcher**, registered and served by ID1, and pings it. Because ID1 is alive, ID2 then elects itself as backup indexing dispatcher, and registers itself as backup indexing dispatcher with ID1 by calling **register_backup**.

ID2 continues to ping ID1 to verify that it is alive. When ID1 dies, the ping returns an exception. ID2 pings one more time for verification. When the second ping also returns an exception, it resolves the **master_dispatcher** interface in the name server and sends the returned client proxy a new ping in case another backup indexing dispatcher has registered as master in the meantime. Because this third ping also returns an exception, there is no master indexing dispatcher alive. ID2 therefore elects itself as master indexing dispatcher, and binds and activates the **master_dispatcher** interface in the name server.

When ID1 later comes alive again, it performs the same steps as ID2. It finds ID2 has registered as master indexing dispatcher and is active; therefore ID1 registers itself as backup indexing dispatcher with ID2. It then starts to monitor ID2 with pings to verify that it remains active.

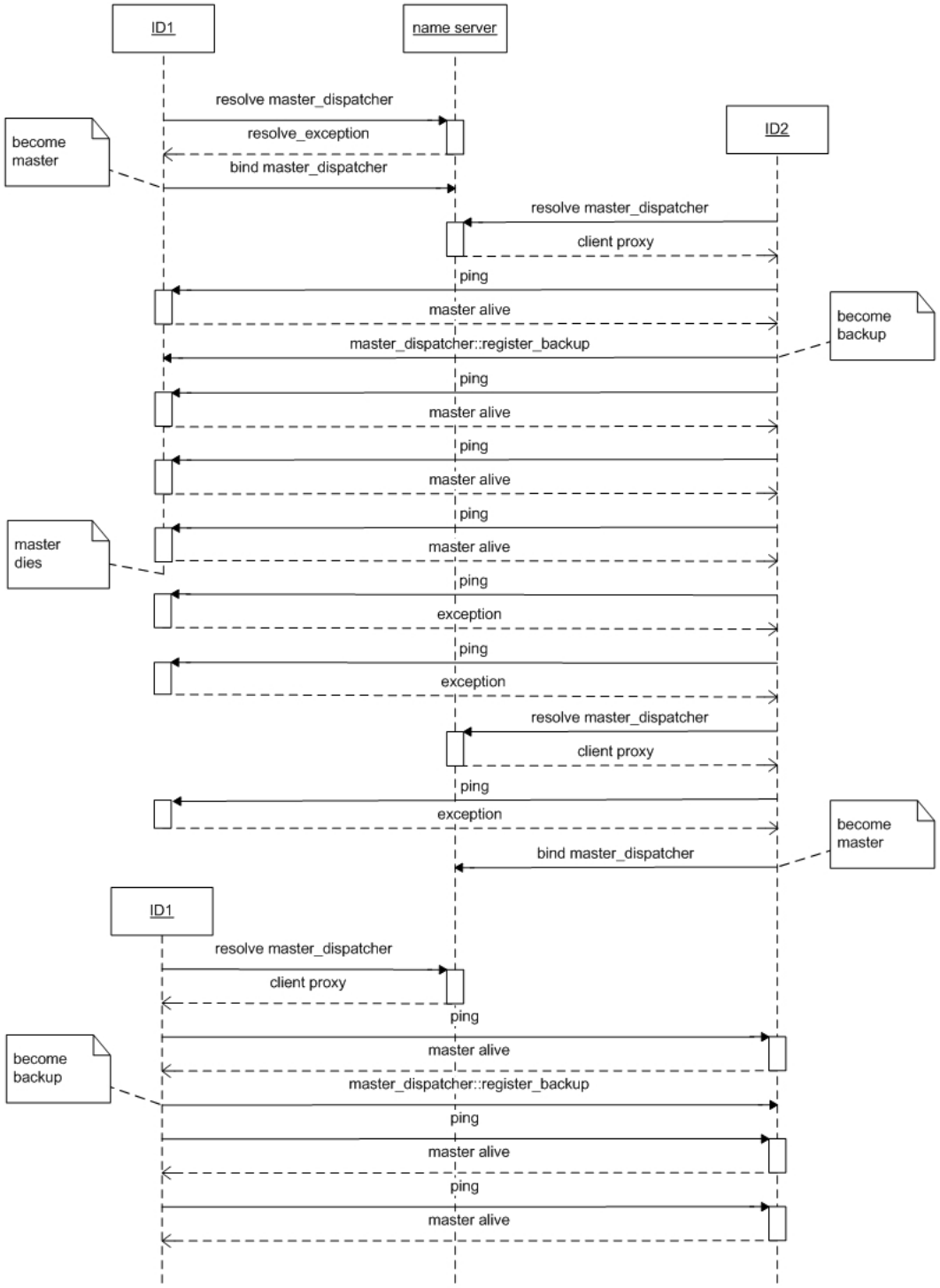

Figure 3: Determining which is the master node and which is the backup node

3.1 indexingengine::master_dispatcher Server Details

The indexing dispatcher elected as master indexing dispatcher has the role as protocol server for the **master_dispatcher** interface. It enables backup indexing dispatchers to register and monitor the master indexing dispatcher.

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 master indexing dispatcher MUST maintain the following state:

backup session factory: A state containing **session_factory** client proxies to registered backup indexing dispatchers, enabling the master indexing dispatcher to communicate with the registered backup indexing dispatchers using the session factory interface.

3.1.2 Timers

None.

3.1.3 Initialization

The master indexing dispatcher MUST use the Middleware **bind** method to register an **indexingengine::master_dispatcher** server object in 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: This MUST be a string containing the value "esp/clusters/webcluster/indexing/dispatcher".

object_id: This MUST be an integer that is unique for each server object.

host: A string specifying the **host name** of the server hosting the server object.

port: This MUST be an integer that contains the port number of the server object on the protocol server. The value is **base port** + 390.

interface_type: This MUST be a string holding the value "indexingengine::master_dispatcher".

interface_version: This MUST be a string holding the value "5.1".

The registration of the **master_dispatcher** server object in the name server enables other indexing dispatchers to detect that a master indexing dispatcher has already registered, and that the other indexing dispatchers MUST register as backup indexing dispatchers with the master indexing dispatcher.

3.1.4 Message Processing Events and Sequencing Rules

This interface includes the following method.

Method	Description
register_backup	Sends a session_factory client proxy from a backup indexing dispatcher to the master indexing dispatcher.

3.1.4.1 register_backup

The **register_backup** method sends a **session_factory** client proxy from a backup indexing dispatcher to the master indexing dispatcher.

```
void register_backup(in string hostname,
  in coreprocessing::session_factory backup_session_factory );
```

hostname: A string holding the **fully qualified domain name (FQDN)** of the backup indexing dispatcher.

backup_session_factory: A **coreprocessing::session_factory** client proxy, as specified in section 3.3.

Return Values: None.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying Middleware protocol as specified in [MS-FSMW].

The master indexing dispatcher MUST store **backup_session_factory** in the **backup session factory** state.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 indexingengine::master_dispatcher Client Details

Any backup indexing dispatcher acts as a protocol client for the **master_dispatcher** interface, used to register with the master indexing dispatcher.

3.2.1 Abstract Data Model

None.

3.2.2 Timers

The master dispatcher interface uses one timer:

Check for master: The interval between each time a backup indexing dispatcher looks for an available master indexing dispatcher. The interval is implementation specific of the higher level application; the default interval is 30 seconds.

3.2.3 Initialization

The backup indexing dispatcher MUST use the Middleware **resolve** method to find the client proxy to the **master_dispatcher** server object bound in the name server, as specified in [MS-FSMW] section 3.4.4.1.The parameters for the **resolve** method are:

name: This MUST be a string holding the value "esp/clusters/webcluster/indexing/dispatcher".

interface_type: This MUST be a string holding the value "indexingengine::master_dispatcher".

version: This MUST be a string holding the value "5.1".

If the **resolve** method returns a **resolve_exception**, as specified in [MS-FSMW] section 2.2.21, a master indexing dispatcher is not available, and the backup indexing dispatcher MUST attempt to register itself as the master indexing dispatcher, as specified in section 3.1.3. If another backup indexing dispatcher registers itself as a master indexing dispatcher before this, then the backup indexing dispatcher MUST register itself as backup with the new master indexing dispatcher.

3.2.4 Message Processing Events and Sequencing Rules

3.2.4.1 register_backup

The **register_backup** method is specified in section 3.1.4.1.

After a backup indexing dispatcher has registered as backup with the master indexing dispatcher, the backup indexing dispatcher MUST use the built in **__ping** method of the **master_dispatcher** interface, as specified in [MS-FSMW] section 3.2.4.2, to determine whether the master indexing dispatcher is active.

If the master indexing dispatcher is not active, the protocol client MUST attempt to take the role as master indexing dispatcher.

3.2.5 Timer Events

The **Check for master** timer triggers the backup indexing dispatcher to test for an active master indexing dispatcher, as specified in section 3.2.

3.2.6 Other Local Events

None.

3.3 coreprocessing::session_factory Server Details

A backup indexing dispatcher performs the role of protocol server for the **session_factory** interface. It enables the backup indexing dispatcher to receive messages from the master indexing dispatcher regarding creation, recreation, and closing of session server objects.

3.3.1 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 backup indexing dispatcher MUST maintain the following states:

session holder: A state containing a set of **session** server objects, where each server object can be referenced by a session identifier.

callback message: A state associated with a **session** server object holding a client proxy to a **callback** server object, used for sending callback messages, as specified in [MS-FSDP] section 3.

3.3.2 Timers

None.

3.3.3 Initialization

The backup indexing dispatcher MUST send a **session_factory** client proxy to the master indexing dispatcher using **master_dispatcher::register_backup**, as specified in section 3.1.4.1.

3.3.4 Message Processing Events and Sequencing Rules

This interface includes the following methods.

Method	Description
create	Creates a **session** server object identified by a session identifier, and returns a **session** client proxy.
recreate	Recreates a feeding session with a given identifier, and returns a **session** client proxy.
close	Closes a session with a given identifier.

3.3.4.1 create

The **create** method creates a **session** server object identified by a session identifier, and returns a **session** client proxy.

```
coreprocessing::session create(
  in long id,
  in string collection,
  in coreprocessing::operation_callback callback,
  in cht::core::guarantee_set guarantees)
  raises (unknown_collection_error, core::unsupported_guarantee_set);
```

id: The identifier of the new **session** server object to create, which MUST be an integer equal to or greater than zero.

collection: A string holding the name of the content collection to use.

callback: A callback message client proxy, as specified in [MS-FSDP] section 3.4.

guarantees: A **guarantee_set** Cheetah entity, as specified in section 2.2.3. It MUST contain either one **feeding_priority** Cheetah entity, as specified in section 2.2.4, which specifies the priority for the feeding session represented by the **session** client proxy, or an empty Cheetah collection.

Return Values: A **session** client proxy, as specified in [MS-FSDP] section 3.

Exceptions Thrown:

The exception **unknown_collection_error** MUST NOT be used.

The exception **unsupported_guarantee_set** MUST be thrown if unable to create a **session**.

This method MUST create, return, and activate the new **session** server object.

This method MUST store the **session** server object in the **session holder** state, with **id** as the unique key.

This method MUST store the **callback** client proxy in the **callback message** state associated with the newly created **session** server object.

3.3.4.2 recreate

The **recreate** method recreates a feeding session with a given identifier, and returns a **session** client proxy.

```
coreprocessing::session recreate(
  in long id,
  in string collection,
  in coreprocessing::operation_callback callback,
  in cht::core::guarantee_set guarantees)
  raises (unknown_collection_error, core::unsupported_guarantee_set);
```

id: The identifier of the **session** server object to recreate, which MUST be an integer equal to or greater than zero.

collection: A string holding the name of the content collection to use.

callback: A callback message client proxy, as specified in [MS-FSDP] section 3.

guarantees: A **guarantee_set** Cheetah entity, as specified in section 2.2.5. It MUST contain either one **feeding_priority** Cheetah entity, as specified in section 2.2.4, specifying the priority for the feeding session represented by the **session** client proxy, or an empty Cheetah collection.

Return Values: A **session** client proxy, as specified in [MS-FSDP] section 3.

Exceptions Thrown:

The exception unknown_collection_error MUST NOT be used.

The exception **unsupported_guarantee_set** MUST be thrown if unable to recreate the **session**.

This method MUST determine whether a **session** server object identified by **id** exists in the **session holder** state. If true, a client proxy to this existing server object MUST be returned. If false, a new session server object MUST be created and activated, and a client proxy to the created session server object MUST be returned.

This method MUST store the **session** server object in the **session holder** state, with **id** as the unique key.

This method MUST store the **callback** client proxy in the **callback message** state, associated with the newly created **session** server object.

3.3.4.3 close

The **close** method closes a session with a given identifier.

```
void close(in long id)
```

id: The identifier of the **session** to close, which MUST be an integer equal to or greater than zero.

Return Values: None.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying Middleware protocol as specified in [MS-FSMW].

This method MUST determine whether a **session** server object identified by **id** exists in the **session holder** state, and if so, close the server object and remove it from the **session holder** state.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 coreprocessing::session_factory Client Details

The master indexing dispatcher, acting as protocol client, forwards requests to create, recreate, and close sessions to backup indexing dispatchers on the session factory interface.

3.4.1 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 indexing dispatcher MUST maintain the **backup session factory** state as specified in section 3.1.1.

3.4.2 Timers

None.

3.4.3 Initialization

The master indexing dispatcher MUST use the client proxy references found in the **backup session factory** state to access the backup **indexing dispatchers**.

3.4.4 Message Processing Events and Sequencing Rules

None.

3.4.5 Timer Events

None.

3.4.6 Other Local Events

None.

4 Protocol Examples

4.1 Registering a backup indexing dispatcher

This example will describes how to use the **register_backup** method of the **master_dispatcher** interface, as specified in section 3.1.4.1, so an indexing dispatcher can register as a backup indexing dispatcher with a master indexing dispatcher and send the client proxy for a **session_factory** interface.

First the protocol server creates a server object implementing the **master_dispatcher** interface, and registers it in the name server. The protocol client then acquires a client proxy to that **master_dispatcher** interface by resolving the server object in the name server. This is possible because both the protocol client and protocol server have agreed a priori on both the location of the shared name server, and the symbolic name of the server object.

The protocol client is now ready to call the **register_backup** method on the **master_dispatcher** client proxy, and send a client proxy to its **session_factory** interface.

4.2 Sample code

4.2.1 Protocol server initialization

```
SET server_object_instance TO INSTANCE OF master_dispatcher SERVER OBJECT

SET server_object_host TO "myserver.mydomain.com"

SET server_object_port TO "1234"

SET server_object_interface_type TO "indexingengine::master_dispatcher"

SET server_object_interface_version TO "5.1"

SET server_object_name TO "esp/clusters/webcluster/indexing/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/clusters/webcluster/indexing/dispatcher"

SET server_object_type TO "indexingengine::master_dispatcher"

SET server_object_version TO "5.1"

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"
```

```
SET backup_session_factory_instance TO INSTANCE OF coreprocessing::session_factory SERVER
OBJECT

CALL master_dispatcher_client_proxy.register_backup WITH hostname AND
backup_session_factory_instance
```

4.2.4 Protocol server response

```
ADD backup_session_factory_instance TO backup_session_factory_list
```

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 and complete listing of Cheetah entities used in this protocol are provided in the following sections. The **coreprocessing::operation_callback** interface is specified in [MS-FSDP] section 3.4.

6.1 FSIDL

```
module cht {
  module core {
    typedef sequence<octet> cheetah;
    typedef cheetah guarantee_set;
  };
};

module interfaces {
  module core {
    exception unsupported_guarantee_set {
      string message;
    };
  };

  module indexingengine {

    interface master_dispatcher {
#pragma version master_dispatcher 5.1
      void register_backup(in string hostname,
                           in coreprocessing::session_factory backup_session_factory );
    };
  };

  module coreprocessing {
    exception unknown_collection_error {
    };

    interface session_factory {
      #pragma version session_factory 5.1

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

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

      void close(in long id);
    };
  };
};
```

6.2 Cheetah Entities

```
entity guarantee {
};

entity feeding_priority : guarantee {
  attribute int priority;
};

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

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

N

O

P

R

S

T

V