[MS-FSIFT]:
Indexer Fault Tolerance Protocol Specification

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	1.05	Minor	Clarified the meaning of the technical content.

Table of Contents

1 Introduction

This document specifies the Indexer Fault Tolerance Protocol, which is used for distributing and synchronizing data structures between indexing nodes, through which functionality the index column achieves fault-tolerant behavior.

1.1 Glossary

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

fault-tolerant
fully qualified domain name (FQDN)
marshal
unmarshal

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

abstract object reference (AOR)
base port
Boolean
Cheetah
Cheetah checksum
Cheetah entity
client proxy
content collection
document identifier
exclusion list
FAST Index Markup Language (FIXML)
FAST Search Interface Definition Language (FSIDL)
index column
index partition
indexer row
indexing node
inverted index
item
master indexer node
name server
query matching node

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", November 2009.

[MS-FSID] Microsoft Corporation, "Indexing Distribution Protocol Specification", November 2009.

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

[MS-FSRFCO] Microsoft Corporation, "Remote File Copy Orchestration Protocol Specification", February 2010.

[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-FSFIXML] Microsoft Corporation, "FIXML Data Structure", November 2009.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

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

1.3 Protocol Overview (Synopsis)

On the highest level, the index is partitioned across several **index columns**. On the level of each **indexing node**, the index is partitioned into a disjointed set of **index partitions**. These index partitions are denoted by integers from 0 to n-1, where n is the number of index partitions on the indexing node. A full set of disjoint index partitions is called an index set, and the set of index partitions currently used by **query matching nodes** to facilitate search queries is called the active index set.

In a **fault-tolerant** system setup there are several indexing nodes in the same index column, with indexing nodes assuming different column roles. In every index column one of the indexing nodes assumes the role of the **master indexer node**, while the rest are referred to as backup indexing nodes. The different indexing nodes are identified by their **indexer row** identifier, as shown in the following figure.

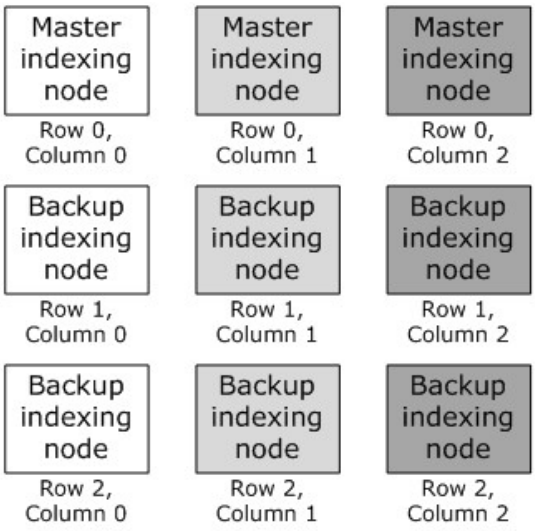

Figure 1: Matrix of indexing nodes in a fault-tolerant topology

The backup indexing nodes have all of the **item** data needed to build indexes, but it is only the master indexer node that performs the actual indexing.

Item operations, such as adding new items or removing old items, result from Middleware calls, as described in [MS-FSMW]. Supported item operations include the **update_operation**, which adds a new item to the **inverted index**, and the **remove_operation**, which removes an item from the inverted index. All supported item operations are described in [MS-FSID] section 2.2.

The high-level item operations are converted by the master indexer node into low-level **sequence operations**, as described in section 2.2.2. The set of indexable items are stored in the intermediate **FAST Index Markup Language (FIXML)** format, as described in [MS-FSFIXML], and the sequence operations manipulate both the FIXML files and the inverted index structures. FIXML files are identified by using a **file identifier**, an increasing integer sequence number. A single FIXML file potentially contains several items.

The master indexer node also keeps a fault-tolerant storage, a data structure that contains a backlog of processed sequence operations. The sequence operations are numbered with an increasing integer sequence identifier. The fault-tolerant storage enables a restarted backup indexing node to quickly become fully synchronized, as the master indexer node only has to resend the sequence operations that were not delivered during the backup indexing node's downtime.

Using the protocol described in this document, the backup indexing nodes register a file receiver with the master indexer node. The file receiver is a server object implementing the file_receiver interface, as described in [MS-FSRFCO]. The file receiver interface is used for transferring the inverted index from the master indexer node to the backup indexing nodes.

The Indexer Fault Tolerance Protocol uses four interrelated **FAST Search Interface Definition Language (FSIDL)** interfaces, with both the master indexer node and the backup indexing nodes taking on the role of protocol client in some transactions, and protocol server in others. The interfaces are described in the following sections.

1.3.1 Column Backup

The Column Backup interface is used by the master indexer node in its communication with the backup indexing nodes. The provided functionality includes the following:

- Retrieving the indexer row identifier of the backup indexing node.
- Retrieving the **fully qualified domain name (FQDN)** of the backup indexing node.
- Submitting a set of sequence operations to the backup indexing node.
- Forcing the backup indexing node to re-resolve the master indexer node.

1.3.2 Column Master

The Column Master interface is used by the backup indexing nodes in their communication with the master indexer node. The provided functionality includes the following:

- Retrieving the indexer row identifier of the master indexer node.
- Registering the existence of a backup indexing node.
- Unregistering unresponsive backup indexing nodes.
- Forcing the master indexer node to relinquish status as master indexer node.

- Registering a subscribing file receiver.

1.3.3 Content Operation Sequence Store

The Content Operation Sequence Store interface is used by the backup indexing nodes to request missing item data from other indexing nodes in the index column. The provided functionality includes the following:

- Retrieving the indexer row identifier of the remote indexing node.
- Confirming whether the remote indexing node is the master indexer node.
- Querying whether the remote indexing node has a specific sequence operation available.
- Querying the range of sequence operations available from the remote indexing node.
- Requesting an asynchronous transfer of sequence operations.
- Retrieving the fully qualified domain name (FQDN) of the remote indexing node.
- Retrieving the sequence identifier of the newest sequence operation stored in the remote indexing node's fault-tolerant storage.
- Retrieving the sequence identifier of the oldest sequence operation stored in the remote indexing node's fault-tolerant storage.

1.3.4 Sequence Receptor

The Sequence Receptor interface is used by indexing nodes to respond to requests issued using the Content Operation Sequence Store interface. The provided functionality includes the following:

- Retrieving the fully qualified domain name (FQDN) of the remote indexing node.
- Submitting a set of sequence operations as a response to requests issued using the Content Operation Sequence Store interface.

1.4 Relationship to Other Protocols

This 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 used by this protocol:

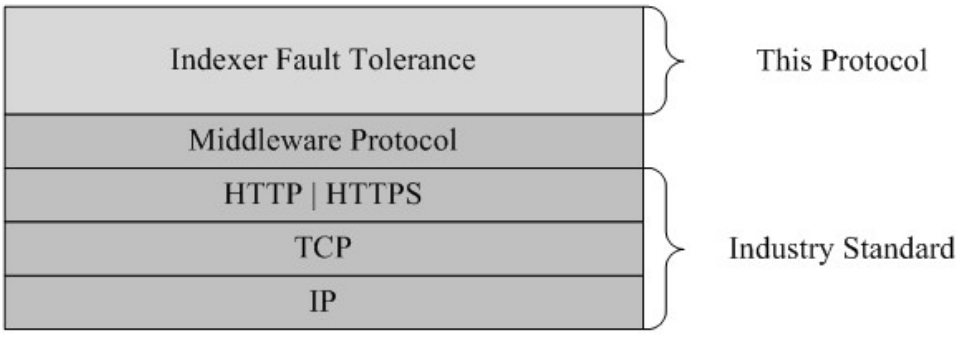

Figure 2: This protocol in relation to other protocols

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 applicable where there is a need for distributing and synchronizing data structures between indexing nodes.

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 sections 3.1.3 and 3.3.3 for the specific version numbers.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The messages supported by the interfaces specified in sections 3.1.4 and 3.3.4 MUST be sent as HTTP POST messages, as specified in [MS-FSMW].

2.2 Common Data Types

The allowed FSIDL data types are specified in [MS-FSMW]. This protocol also uses custom Cheetah data types that are **marshaled** and embedded in a generic collection of bytes.

Cheetah entities MUST be encoded as specified in [MS-FSCHT] section 2.2. The **Cheetah checksum** of the Cheetah messages MUST be -2127454238. The Cheetah type identifier for the Cheetah entities MUST be as specified in the following table.

Cheetah entity	Cheetah type identifier
sequence_log_info	5
sequence_operation	6
empty_operation	7
fixml_invalidation	8
Remdoclist	9
Exclusionlist	10
remove_collection	11
fixml_append	12
document_error	15
content_operation_sequence	16

In addition to the preceding Cheetah entities, the protocol also makes use of data types that are aliases for standard FSIDL data types. The aliased data types are not custom data types, but rather standard FSIDL data types that have been given more convenient or verbose names to increase code readability.

2.2.1 sequence_log_info

The **sequence_log_info** data type is used to encapsulate information regarding the content of the fault-tolerant storage. The structure of this data type, as specified in section 6.2, is as follows:

```
entity sequence_log_info {
  attribute longint low_sequence_id;
  attribute longint high_sequence_id;
  attribute longint processed_sequence_id;
};
```

low_sequence_id: An integer value containing the sequence identifier of the oldest sequence operation stored in the fault-tolerant storage. The value MUST be greater than or equal to zero, and the value MUST be less than or equal to **high_sequence_id**.

high_sequence_id: An integer value containing the sequence identifier of the newest sequence operation stored in the fault-tolerant storage. The value MUST be greater than or equal to zero, and the value MUST be greater than or equal to **low_sequence_id**.

processed_sequence_id: An integer value containing the sequence identifier of the newest operation the indexing node has executed. The value MUST be greater than or equal to zero, and the value MUST be less than or equal to **high_sequence_id**.

2.2.2 sequence_operation

The **sequence_operation** data type is the base type for operations manipulating the set of indexed items. The structure of this data type, as specified in section 6.2, is as follows:

```
entity sequence_operation {
  attribute longint sequence_number;
  attribute longint operation_id;
};
```

sequence_number: An integer value containing the sequence identifier of the sequence operation. The value MUST be greater than or equal to zero.

operation_id: The identifier of the item operation from which the sequence operation originated.

2.2.3 empty_operation

The **empty_operation** data type is a subtype of **sequence_operation** specified in section 2.2.2. The **empty_operation** MUST NOT have any effect. The structure of this data type, as specified in section 6.2, is as follows:

```
entity empty_operation : sequence_operation {
};
```

2.2.4 fixml_invalidation

The **fixml_invalidation** data type is a sub-type of **sequence_operation** specified in section 2.2.2. The **fixml_invalidation** data type is used to invalidate an item. The structure of this data type, as specified in section 6.2, is as follows:

```
entity fixml_invalidation : sequence_operation {
  attribute string document_id;
  attribute int file_id;
  attribute int magic_idx;
  attribute bool is_update;
};
```

document_id: The **document identifier (3)**.

file_id: The file identifier of the FIXML file containing the item to be removed.

magic_idx: An integer value from 0 to n-1, where n is the number of items in the FIXML file, describing the position of the item in the FIXML file.

is_update: MUST be **true** if the sequence operation was derived from a high-level **update_operation** as specified in [MS-FSID] section 2.2.36; otherwise **false**.

2.2.5 remdoclist

The **remdoclist** data type is a subtype of **sequence_operation** specified in section 2.2.2. The **remdoclist** data type is used to add an item to the **remove list**, a list of items that are to be removed from the index the next time it is re-indexed. Instead of removing the item from the index directly, it is removed from query results by being added to the query matching nodes' **exclusion lists**. The structure of this data type, as specified in section 6.2, is as follows:

```
entity remdoclist : sequence_operation {
  attribute string  document_id;
  attribute int     old_file_id;
  attribute int     new_file_id;
};
```

document_id: The document identifier (3) of the item.

old_file_id: The file identifier of the FIXML file containing the item to remove.

new_file_id: If the sequence operation was derived from a high-level **update_operation,** as specified in [MS-FSID] section 2.2.36, the value MUST be the file identifier of the FIXML file containing the updated item. If the sequence operation was not derived from a high-level **update_operation**, the value MUST be the same as **old_file_id**.

2.2.6 exclusionlist

The **exclusionlist** data type is a subtype of **sequence_operation** specified in section 2.2.2. The **exclusionlist** data type is used to add an item to the exclusion list. An item that has been added to the exclusion list will be removed from query results, even if it is present in the index. The structure of this data type, as specified in section 6.2, is as follows:

```
entity exclusionlist : sequence_operation {
  attribute string  document_id;
  attribute int     old_file_id;
};
```

document_id: The document identifier (3) of the item.

old_file_id: The file identifier of the FIXML file containing the item to exclusion list.

2.2.7 remove_collection

The **remove_collection** data type is a subtype of **sequence_operation** specified in section 2.2.2. The **remove_collection** data type is used to remove a **content collection** and all of its indexed items. The content collection concerned is implicitly derived from the higher level session set up before item operations are fed, as specified in [MS-FSID] section 3.1.4.1. The structure of this data type, as specified in section 6.2, is as follows:

```
entity remove_collection : sequence_operation {
```

```
};
```

2.2.8 fixml_append

The **fixml_append** data type is a subtype of **sequence_operation** specified in section 2.2.2. The **fixml_append** data type is used to add an item to a FIXML file. The structure of this data type, as specified in section 6.2, is as follows:

```
entity fixml_append : sequence_operation {
  attribute string document_id;
  attribute string document_content;
  attribute int file_id;
  attribute int magic_idx;
  attribute bool is_update;
};
```

document_id: The document identifier (3) of the item.

document_content: The content of the item to append.

file_id: The file identifier of the FIXML file in which to place the item.

magic_idx: An integer describing the position of the item in the FIXML file. This MUST be between 0 and n-1, where n is the number of items in the FIXML file.

is_update: A **Boolean** value that MUST be **true**.

2.2.9 document_error

The **document_error** data type is a subtype of **sequence_operation** specified in section 2.2.2. The **document_error** data type contains errors relating to an item. The structure of this data type, as specified in section 6.2, is as follows:

```
entity document_error : sequence_operation {
  attribute string document_id;
  attribute int error_code;
  attribute int action;
  attribute string subsystem;
  attribute string error_message;
}
```

document_id: The document identifier (3) of the item.

error_code: An integer value containing an error code. The value MUST be one of the values described in the following table.

Error code	Description
1	Missing attribute.
2	Generic error.
3	Unknown item.

Error code	Description
4	Indexer suspended.
5	FIXML write error. Failed to persist item operation.
6	Unknown content collection.
7	Partial update error.

action: An integer value containing a suggested action code. The value MUST be one of the values described in the following table.

Action code	Description
1	Resubmit.
2	Limited resubmit.
3	Drop operation.
4	Terminate.

subsystem: A string value that MUST be "indexing".

error_message: A string that contains the error message.

2.2.10 content_operation_sequence

The **content_operation_sequence** data type is a collection of sequence operations. The structure of this data type, as specified in section 6.2, is as follows:

```
entity content_operation_sequence {
  attribute int session_id;
  attribute string document_collection;
  attribute longint low_sequence_id;
  attribute longint high_sequence_id;
  collection sequence_operation operations;
};
```

session_id: The session identifier of the session used to feed the high level operations from which the sequence operations were deduced. The sessions are created using the **create_session** method, as specified in [MS-FSID] section 3.2.4.1.

document_collection: The content collection associated with the sequence operations.

low_sequence_id: The lowest sequence identifier of the operations contained in the structure.

high_sequence_id: The highest sequence identifier of the operations contained in the structure.

operations: A collection of **sequence_operations**, as specified in section 2.2.2.

2.2.11 cheetah

The **cheetah** data type is an alias for a collection of bytes. It is used for all custom data types that are marshaled using Cheetah. The structure of this data type, as specified in section 6.2, is as follows:

```
typedef sequence<octet> cheetah;
```

3 Protocol Details

This protocol consists of four different protocol interfaces: **column_master**, **column_backup**, **content_operation_sequence_store**, and **sequence_receptor**. For **column_master** and **content_operation_sequence_store** the master indexer node acts as a protocol server, and the backup indexing nodes act as the protocol client. For **column_backup** and **sequence_receptor**, the roles are reversed, and the master indexer node is the protocol client, while the backup indexing nodes are the protocol servers.

3.1 column_master and content_operation_sequence_store Server Details

A master indexer node implementing the **column_master** and **content_operation_sequence_store** interfaces receives messages from the backup indexing nodes.

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 following data structures are needed by the master indexer node in the role of protocol server:

fault_tolerance_storage: a data structure containing a backlog of sequence operations.

lowest_sequence_id: an integer value containing the sequence identifier of the oldest sequence operation stored in the **fault_tolerance_storage**.

highest_sequence_id: an integer value containing the sequence identifier of the newest sequence operation stored in the **fault_tolerance_storage**.

highest_processed_id: an integer value containing the sequence identifier of the newest sequence operation that the indexing node has processed.

column_role: the current column role of the indexing node in the index column, either "BACKUP," "MASTER," or "UNKNOWN." See section 3.1.3 for details on establishing the column role.

backups: a set of client proxies implementing the **column_backup** interface. The client proxies are not resolved using the **name server**, rather they are registered using the **register_backup_node** method of the **column_master** interface.

file_receivers: a set of client proxies implementing the **file_receiver** interface, specified in [MS-FSRFCO]. The file receivers are registered and deregistered by using the Indexer Fault Tolerance protocol; however, the actual use of the client proxies are in relation with the protocol specified in [MS-FSRFCO].

3.1.2 Timers

None.

3.1.3 Initialization

Unless the column role of an indexing node is decided through static and implementation-specific configuration options, the decision about which column role to assume is established through the following steps, performed at system initialization:

The column role is set to "UNKNOWN".

The indexing node tries to resolve the **column_master** interface in the name server, using the **resolve** method, as specified in [MS-FSMW]. There are two possible outcomes:

The interface is successfully resolved, and the indexing node tries to ascertain the availability of the current master indexer node using the standard Middleware **__ping** method, as specified in [MS-FSMW] section 4.2. There are two possible outcomes:

The remote indexing node responds to the **__ping**, and the local indexing node assumes the role of "BACKUP".

The remote indexing node is unavailable, and the local indexing node tries to bind the **column_master** interface itself. If it succeeds in binding to the **column_master** interface, it assumes the role of "MASTER;" otherwise, the local indexing node repeats step 2.

The interface cannot be resolved, and the local indexing node tries to bind the **column_master** interface itself. If it succeeds in binding to the **column_master** interface, it assumes the role of "MASTER;" otherwise, the local indexing node repeats step 2.

When the backup indexing nodes are established, they will periodically ascertain the availability of the master indexer node, and if the master indexer node fails to respond to a **__ping**, the backup indexing nodes will try to bind to the **column_master** interface and assume the role of master indexer node. A typical scenario is described in the following figure.

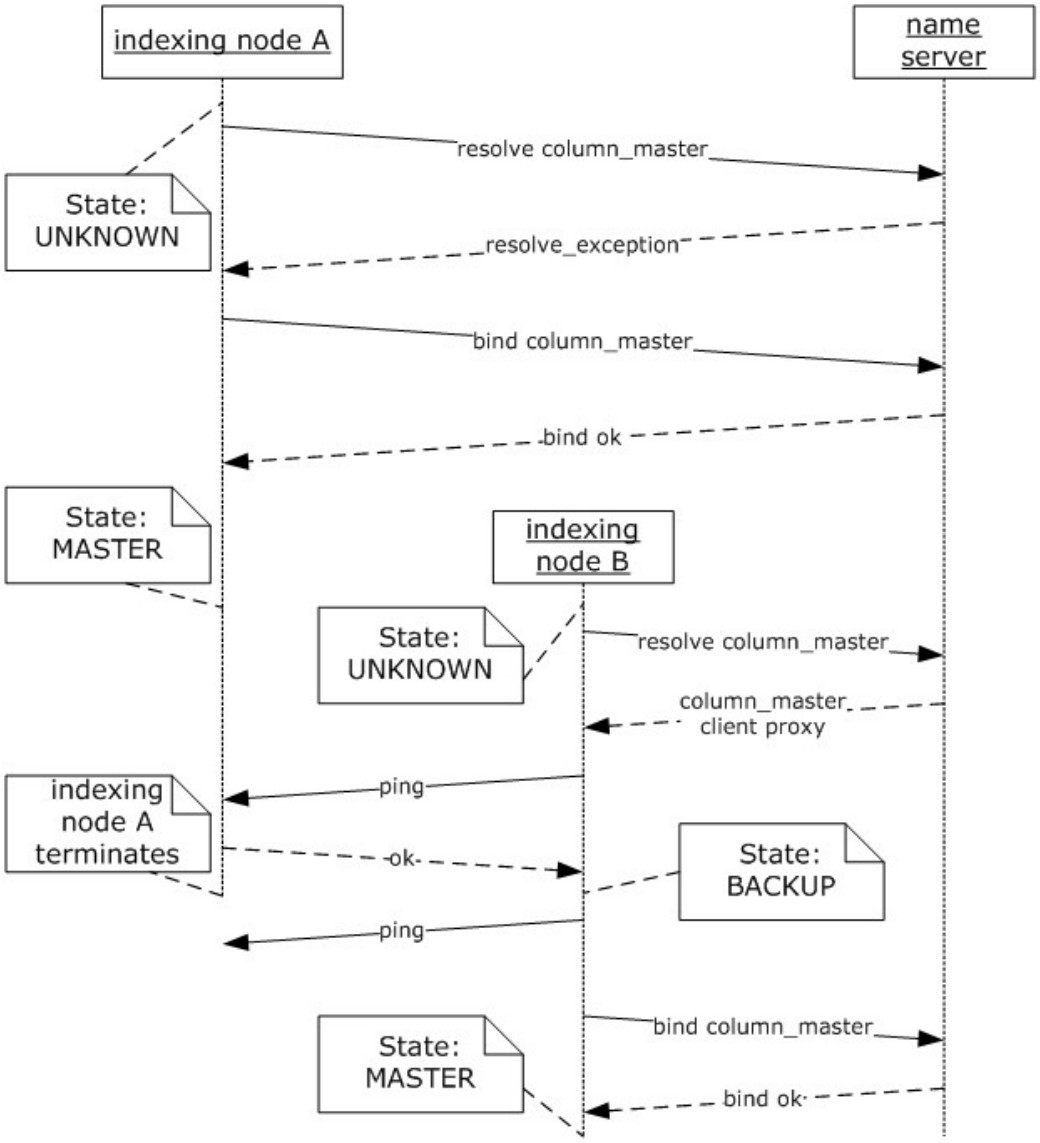

Figure 3: Establishing column role

If the indexing nodes have predefined roles, a backup indexing node will not try to register the **column_master** interface in the absence of a master indexer node, rather it will remain a backup indexing node and continually try to resolve the predefined master indexer node.

The master indexer node MUST use the Middleware **bind** method to register an **rtsearch::column_master** server object in the name server, as specified in [MS-FSMW] section 2.2. The parameters for the **bind** method are encapsulated in an **abstract object reference (AOR)**, as specified in [MS-FSMW] section 3.4.4.2:

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

port: An integer specifying the port number of the server object on the protocol server. The value is **base port** plus 390.

interface_type: A string that MUST be "rtsearch::column_master".

interface_version: A string that MUST be "5.9".

object_id: An integer value that MUST be unique for each server object.

name: A string that MUST be "esp/clusters/webcluster/indexing/indexer-C/columnmaster", where *C* is the index column identifier.

If fault-tolerant behavior is enabled, all indexing nodes MUST use the Middleware **bind** method to register an **rtsearch::content_operation_sequence_store** server object in the name server, as specified in [MS-FSMW] section 2.2. The parameters for the **bind** method are encapsulated in an abstract object reference (AOR), as specified in [MS-FSMW] section 3.4.4.2:

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

port: An integer specifying the port number of the server object on the protocol server. The value is base port plus 390.

interface_type: A string that MUST be "rtsearch::content_operation_sequence_store".

interface_version: A string that MUST be "5.6".

object_id: An integer value that MUST be unique for each server object.

name: A string that MUST be "esp/clusters/webcluster/indexing/indexer-C-R/opr_seq_store", where *C* is the index column identifier and *R* is the indexer row identifier.

3.1.4 Message Processing Events and Sequencing Rules

The message type is determined at the Middleware level. The Middleware MUST call the correct method of a server object implementing an interface. If custom data types are present in the signature of the method being called, the Middleware MUST **unmarshal (1)** the Cheetah data before passing the arguments to the server object.

In accordance with the Middleware specification, the generic Middleware exceptions may be thrown from any method, and are thus not defined in the FSIDL method signatures.

The available methods are specified in the following table.

Method	Description
column_master::get_row_id	Returns the indexer row of the indexing node.
column_master::register_backup_node	Registers a backup indexing node.
column_master::has_backup_node	Returns whether a specific indexing node is registered with the master indexer node.
column_master::check_backup_nodes	Unregisters unresponsive backup indexing nodes.
column_master::abdicate	Relinquishes status as master indexer node.
column_master::connect_receiver	Adds a file receiver to the

Method	Description
	file_receivers state variable.
column_master::disconnect_receiver	Removes a file receiver from the **file_receivers** state variable.
content_operation_sequence_store::is_master	Returns whether the indexing node is the master indexer node of the index column.
content_operation_sequence_store::get_stored_sequences	Returns the range of sequence operations stored in the fault-tolerant storage.
content_operation_sequence_store::has_sequence_id	Returns whether the fault-tolerant storage contains a specified sequence operation.
content_operation_sequence_store::request_sequences	Requests an asynchronous transfer of sequence operations.
content_operation_sequence_store::get_row_id	Returns the indexer row of the indexing node.
content_operation_sequence_store::get_hostname	Returns the host name of the indexing node.
content_operation_sequence_store::get_highest_sequence_id	Returns the value of the state variable **highest_sequence_id**.
content_operation_sequence_store::get_lowest_sequence_id	Returns the value of the state variable **lowest_sequence_id**.

3.1.4.1 column_master::get_row_id

The **get_row_id** method MUST return the indexer row of the indexing node. The structure of this method, as specified in section 6.1, is as follows:

```
long get_row_id();
```

Return values: MUST return the indexer row of the indexing node.

3.1.4.2 column_master::register_backup_node

The **register_backup_node** method MUST add the backup indexing node **client proxy** to the state variable **backups**. The structure of this method, as specified in section 6.1, is as follows:

```
void register_backup_node(in column_backup backup_interface,
                          in long row_id);
```

backup_interface: The client proxy of the backup indexing node registering.

row_id: The indexer row of the backup indexing node registering.

3.1.4.3 column_master::has_backup_node

The **has_backup_node** method MUST return whether the state variable **backups** contains the backup indexing node of a specific indexer row. The structure of this method, as specified in section 6.1, is as follows:

```
boolean has_backup_node(in long row_id);
```

row_id: The indexer row identifier of the backup indexing node.

Return values: The method MUST return **true** if the state variable **backups** contains the specified backup indexing node; otherwise **false**.

3.1.4.4 column_master::check_backup_nodes

The **check_backup_nodes** method MUST remove all unavailable backup indexing nodes from the state variable **backups**. The availability of the backup indexing nodes MUST be ascertained using the **__ping** method on the client proxies, as specified in [MS-FSMW] section 4.2. The structure of this method, as specified in section 6.1, is as follows:

```
void check_backup_nodes();
```

3.1.4.5 column_master::abdicate

The **abdicate** method MUST force the master indexer node to relinquish its status as master indexer node by unbinding the **column_master** interface from the name server, as specified in [MS-FSMW] section 3.4.4.3. The **column_role** state variable MUST be set to "UNKNOWN", and the column role must be newly established in accordance with the description in section 3.1.3. The **abdicate** method MAY return a void value, but SHOULD simply time out. The structure of this method, as specified in section 6.1, is as follows:

```
void abdicate();
```

3.1.4.6 column_master::connect_receiver

The **connect_receiver** method MUST add the **file_receiver** client proxy to the state variable **file_receivers**. The **file_receiver** interface is specified in [MS-FSRFCO]. The availability of the file receiver server object MUST be ascertained using the **__ping** method on the client proxy, as specified in [MS-FSMW] section 4.2. The structure of this method, as specified in section 6.1, is as follows:

```
boolean connect_receiver(in file_receiver receiver,
                         in string hostname,
                         in long port);
```

receiver: The client proxy of the connecting **file_receiver**, as specified in [MS-FSRFCO].

hostname: The host name of the connecting **file_receiver**.

port: The port number of the connecting **file_receiver**.

Return values: The method MUST return **true** if the **file_receivers** state variable is successfully updated and the server object is available; otherwise **false**.

3.1.4.7 column_master::disconnect_receiver

The **disconnect_receiver** method MUST remove the **file_receiver** client proxy associated with the specified host name and port number from the state variable **file_receivers**. The structure of this method, as specified in section 6.1, is as follows:

```
boolean disconnect_receiver(in string hostname, in long port);
```

hostname: The host name of the disconnecting **file_receiver**.

port: The port number of the disconnecting **file_receiver**.

Return values: The method MUST return **true**.

3.1.4.8 content_operation_sequence_store::is_master

The **is_master** method MUST return whether or not the **column_role** state variable equals "MASTER". The structure of this method, as specified in section 6.1, is as follows:

```
boolean is_master();
```

Return values: The method MUST return **true** if the state variable **column_role** equals "MASTER"; otherwise **false**.

3.1.4.9 content_operation_sequence_store::get_stored_sequences

The **get_stored_sequences** method MUST return the range of sequence operations stored in the **fault_tolerance_storage** and the **highest_processed_id** state variables. The structure of this method, as specified in section 6.1, is as follows:

```
cht::rtsmessages::sequence_log_info get_stored_sequences();
```

Return values: The method MUST return a **sequence_log_info** object containing the state variables **highest_sequence_id, lowest_sequence_id** and **highest_processed_id**.

3.1.4.10 content_operation_sequence_store::has_sequence_id

The **has_sequence_id** method MUST return whether the **fault_tolerance_storage** contains the specified sequence operation. The structure of this method, as specified in section 6.1, is as follows:

```
boolean has_sequence_id(in long long sequence_id);
```

sequence_id: The sequence identifier.

Return values: The method MUST return **true** if the **fault_tolerance_storage** contains the sequence operation; otherwise **false**.

3.1.4.11 content_operation_sequence_store::request_sequences

The **request_sequences** method MUST initiate an asynchronous transfer of a range of sequence operations using **sequence_receptor::submit_sequence**. When all of the sequence operations have been submitted, a call to the **sequence_receptor::finished** method MUST be made. The structure of this method, as specified in section 6.1, is as follows:

```
void request_sequences(in sequence_receptor receptor,
                       in long long from_sequence_id,
                       in long long to_sequence_id);
```

receptor: The **sequence_receptor** client proxy through which to send the sequence operations.

from_sequence_id: The lowest identifier of the range of sequence operations to transfer.

to_sequence_id: The highest identifier of the range of sequence operations to transfer.

3.1.4.12 content_operation_sequence_store::get_row_id

The **get_row_id** method MUST return the indexer row identifier of the indexing node. The structure of this method, as specified in section 6.1, is as follows:

```
long get_row_id();
```

Return values: The method MUST return the indexer row identifier of the indexing node.

3.1.4.13 content_operation_sequence_store::get_hostname

The **get_hostname** method MUST return the fully qualified domain name (FQDN) of the indexing node. The structure of this method, as specified in section 6.1, is as follows:

```
string get_hostname();
```

Return values: The method MUST return the fully qualified domain name of the indexing node.

3.1.4.14 content_operation_sequence_store::get_highest_sequence_id

The **get_highest_sequence_id** method MUST return the value of the state variable **highest_sequence_id**. The structure of this method, as specified in section 6.1, is as follows:

```
long long get_highest_sequence_id();
```

Return values: The method MUST return the value of the state variable **highest_sequence_id**.

3.1.4.15 content_operation_sequence_store::get_lowest_sequence_id

The **get_lowest_sequence_id** method MUST return the value of the state variable **lowest_sequence_id**. The structure of this method, as specified in section 6.1, is as follows:

```
long long get_lowest_sequence_id();
```

Return values: The method MUST return the value of the state variable **lowest_sequence_id**.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 column_master and content_operation_sequence_store Client Details

The backup indexing nodes use the **column_master** interface to register with and poll information from the master indexer node. The backup indexing nodes use the **content_operation_sequence_store** interface to obtain missing sequence operations from another indexing node.

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

The protocol client MUST use the Middleware **resolve** method to find the server objects bound in the name server, as specified in [MS-FSMW] section 3.4.4.1.

For resolving the **content_operation_sequence_store** server object, the parameters are:

name: A string that MUST be "esp/clusters/webcluster/indexing/indexer-C-R/opr_seq_store", where *C* is the index column identifier and *R* is the indexer row identifier.

interface_type: A string that MUST be "rtsearch::content_operation_sequence_store".

interface_version: A string that MUST be "5.6".

For resolving the **column_master** server object, the parameters are:

name: A string that MUST be "esp/clusters/webcluster/indexing/indexer-C/columnmaster", where *C* is the index column identifier.

interface_type: A string that MUST be "rtsearch::column_master".

interface_version: A string that MUST be "5.9".

The backup indexing nodes MUST create a server object implementing the **column_backup** interface. The server object must then be registered with the master indexer node using **column_master::register_backup_node**. The server object MUST have the following values set for the abstract object reference (AOR), as specified in [MS-FSMW] section 2.2.14:

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

port: An integer specifying the port number of the server object on the protocol server. The value is base port plus 390.

interface_type: A string that MUST be "rtsearch::column_backup".

interface_version: A string that MUST be "5.14".

object_id: An integer value that MUST be unique for each server object.

To call methods that have parameters of type **sequence_receptor**, a server object implementing the interface **sequence_receptor** MUST first be created. The server object MUST have the following values set for the abstract object reference (AOR), as specified in [MS-FSMW] section 2.2.14:

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

port: An integer specifying the port number of the server object on the protocol server. The value is base port plus 390.

interface_type: A string that MUST be "rtsearch::sequence_receptor".

interface_version: A string that MUST be "5.2".

object_id: An integer value that MUST be unique for each server object.

3.2.4 Message Processing Events and Sequencing Rules

The available methods are specified in the following table.

Method	Description
column_master::get_row_id	Returns the indexer row of the indexing node.
column_master::register_backup_node	Registers a backup indexing node.
column_master::has_backup_node	Returns whether a specific indexing node is registered with the master indexer node.
column_master::check_backup_nodes	Unregisters unresponsive backup indexing nodes.
column_master::abdicate	Relinquishes status as master indexer node.
column_master::connect_receiver	Adds a file receiver to the **file_receivers** state variable.
column_master::disconnect_receiver	Removes a file receiver from the **file_receivers** state variable.
content_operation_sequence_store::is_master	Returns whether the indexing node is the master indexer node of the index column.
content_operation_sequence_store::get_stored_sequences	Returns the range of sequence operations stored in the fault-tolerance storage.

Method	Description
content_operation_sequence_store::has_sequence_id	Returns whether the fault-tolerance storage contains a specified sequence operation.
content_operation_sequence_store::request_sequences	Requests an asynchronous transfer of sequence operations.
content_operation_sequence_store::get_row_id	Returns the indexer row of the indexing node.
content_operation_sequence_store::get_hostname	Returns the host name of the indexing node.
content_operation_sequence_store::get_highest_sequence_id	Returns the value of the state variable **highest_sequence_id**.
content_operation_sequence_store::get_lowest_sequence_id	Returns the value of the state variable **lowest_sequence_id**.

3.2.4.1 column_master::get_row_id

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

3.2.4.2 column_master::register_backup_node

The **register_backup_node** method is specified in section 3.1.4.2. Before this call is made, the backup indexing node MUST first create a **column_backup** server object, as specified in section 3.2.3.

3.2.4.3 column_master::has_backup_node

The **has_backup_node** method is specified in section 3.1.4.3.

3.2.4.4 column_master::check_backup_nodes

The **check_backup_nodes** method is specified in section 3.1.4.4.

3.2.4.5 column_master::abdicate

The **abdicate** method is specified in section 3.1.4.5. The abdicate method will force the protocol server to deactivate the server object implementing the method itself. The **abdicate** method MAY return a void value, but SHOULD simply time out.

3.2.4.6 column_master::connect_receiver

The **connect_receiver** method is specified in section 3.1.4.6.

3.2.4.7 column_master::disconnect_receiver

The **disconnect_receiver** method is specified in section 3.1.4.7.

3.2.4.8 content_operation_sequence_store::is_master

The **is_master** method is specified in section 3.1.4.8.

3.2.4.9 content_operation_sequence_store::get_stored_sequences

The **get_stored_sequences** method is specified in section 3.1.4.9.

3.2.4.10 content_operation_sequence_store::has_sequence_id

The **has_sequence_id** method is specified in section 3.1.4.10.

3.2.4.11 content_operation_sequence_store::request_sequences

The **request_sequences** method is specified in section 3.1.4.11.

3.2.4.12 content_operation_sequence_store::get_row_id

The **get_row_id** method is specified in section 3.1.4.12.

3.2.4.13 content_operation_sequence_store::get_hostname

The **get_hostname** method is specified in section 3.1.4.13.

3.2.4.14 content_operation_sequence_store::get_highest_sequence_id

The **get_highest_sequence_id** method is specified in section 3.1.4.14.

3.2.4.15 content_operation_sequence_store::get_lowest_sequence_id

The **get_lowest_sequence_id** method is specified in section 3.1.4.15.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 column_backup and sequence_receptor Server Details

A backup indexing node implementing the **column_backup** and **sequence_receptor** interfaces receive messages from a master indexer node.

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.

The following data structures are needed by the backup indexing node in the role of protocol server:

fault_tolerance_storage: A data structure containing a backlog of sequence operations.

lowest_sequence_id: An integer value containing the sequence identifier of the oldest sequence operation stored in the **fault_tolerance_storage**.

highest_sequence_id: An integer value containing the sequence identifier of the newest sequence operation stored in the **fault_tolerance_storage**.

highest_processed_id: An integer value containing the sequence identifier of the newest sequence operation that the indexing node has processed.

column_role: The current column role of the indexing node in the index column, either "BACKUP," "MASTER," or "UNKNOWN." See section 3.1.3 for details on establishing the column role.

active_index_set: The currently active index set.

sequence_receptor: A server object implementing the sequence_receptor interface. The server object receives the asynchronous responses to the message **content_operation_sequence_store::request_sequences**.

receptor_finished: A Boolean state variable stating whether or not the **sequence_receptor** has received the submitted set of sequence operations.

3.3.2 Timers

None.

3.3.3 Initialization

The backup indexing nodes MUST create a server object implementing the **column_backup**, as specified in section 3.2.3.

3.3.4 Message Processing Events and Sequencing Rules

The message type is determined at the Middleware level. The Middleware MUST call the correct method of the server object implementing the interface. If custom data types are present in the signature of the method being called, the Middleware MUST unmarshal (1) the Cheetah data before passing the arguments to the server object.

In accordance with the Middleware specification, generic Middleware exceptions may be thrown from any method, and are thus not defined in the FSIDL method signatures.

The available methods are specified in the following table.

Method	Description
column_backup::get_row_id	Returns the indexer row of the indexing node.
column_backup::get_hostname	Returns the host name of the indexing node.
column_backup::submit_sequence	Submits a set of sequence operations.
column_backup::commit_sequence	Persists a set of submitted sequence operations.
column_backup::abort_sequence	Reverts a submitted set of sequence operations.
column_backup::activate_index_set	Updates the active index set.
sequence_receptor::submit_sequence	Submits a set of sequence operations.
sequence_receptor::finished	Sets the state variable **receptor_finished** to **true**.

Method	Description
sequence_receptor::get_hostname	Returns the host name of the remote indexing node.

3.3.4.1 column_backup::get_row_id

The **get_row_id** method MUST return the indexer row identifier of the indexing node. The structure of this method, as specified in section 6.1, is as follows:

```
long get_row_id();
```

Return values: The method MUST return the indexer row identifier of the backup indexing node.

3.3.4.2 column_backup::get_hostname

The **get_hostname** method MUST return the fully qualified domain name (FQDN) of the indexing node. The structure of this method, as specified in section 6.1, is as follows:

```
string get_hostname();
```

Return values: The method MUST return the fully qualified domain name of the indexing node.

3.3.4.3 column_backup::submit_sequence

The **submit_sequence** method MUST update the FIXML files in accordance with the incoming set of sequence operations. The method MUST also update the **highest_processed_id** state variable to include the latest set of sequence operations processed. The structure of this method, as specified in section 6.1, is as follows:

```
boolean submit_sequence(in cht::rtsmessages::content_operation_sequence seq,
                        in string collection_name)
```

seq: The set of sequence operations.

collection_name: The name of the content collection being updated.

Return values: The method MUST return **true** if the FIXML files were successfully updated; otherwise, **false**.

3.3.4.4 column_backup::commit_sequence

The **commit_sequence** method MUST update the state variables **lowest_sequence_id** and **highest_sequence_id** to correspond to the latest set of submitted sequence operations. The structure of this method, as specified in section 6.1, is as follows:

```
void commit_sequence();
```

3.3.4.5 column_backup::abort_sequence

The **abort_sequence** method MUST undo the effect of any submitted but not yet committed set of sequence operations. The structure of this method, as specified in section 6.1, is as follows:

```
void abort_sequence();
```

3.3.4.6 column_backup::activate_index_set

The **activate_index_set** method MUST update the state variable **active_index_set** to correspond to the latest index set available on the backup indexing node. The actual index set is transferred to the backup indexing node using a different protocol, as specified in [MS-FSRFCO]. The structure of this method, as specified in section 6.1, is as follows:

```
void activate_index_set();
```

3.3.4.7 sequence_receptor::submit_sequence

The **submit_sequence** method MUST accept a set of sequence operations, sent as a response to **content_operation_sequence_store::request_sequences**. The interface is used by indexing nodes recovering from downtime, and the method MUST mirror the functionality of a call to **column_backup::submit_sequence** followed by a call to **column_backup::commit_sequence**. The structure of this method, as specified in section 6.1, is as follows:

```
void submit_sequence(in cht::rtsmessages::content_operation_sequence seq);
```

seq: The set of sequence operations submitted from the remote indexing node.

3.3.4.8 sequence_receptor::finished

The **finished** method is an asynchronous callback sent back to the protocol client as a result of the protocol client issuing a **content_operation_sequence_store::request_sequences** call. It MUST set the state variable **receptor_finished** to **true**. The protocol server MUST use it to signal that it has finished submitting the requested sequence operations. The structure of this method, as specified in section 6.1, is as follows:

```
void finished();
```

3.3.4.9 sequence_receptor::get_hostname

The **get_hostname** method MUST return the fully qualified domain name (FQDN) of the indexing node. The structure of this method, as specified in section 6.1, is as follows:

```
string get_hostname();
```

Return values: The method MUST return the fully qualified domain name (FQDN) of the indexing node.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 column_backup and sequence_receptor Client Details

A master indexer node uses the **column_backup** and **sequence_receptor** interfaces to interact with a backup indexing node.

3.4.1 Abstract Data Model

None.

3.4.2 Timers

None.

3.4.3 Initialization

The **column_backup** client proxy is registered with the master indexer node through a call to **column_master::register_backup_node**. The **sequence_receptor** client proxy is supplied as a parameter to **content_operation_sequence_store::request_sequences**.

3.4.4 Message Processing Events and Sequencing Rules

The available methods are specified in the following table.

Method	Description
column_backup::get_row_id	Returns the indexer row of the indexing node.
column_backup::get_hostname	Returns the host name of the indexing node.
column_backup::submit_sequence	Submits a set of sequence operations.
column_backup::commit_sequence	Persists a set of submitted sequence operations.
column_backup::abort_sequence	Reverts a submitted set of sequence operations.
column_backup::activate_index_set	Updates the active index set.
sequence_receptor::submit_sequence	Submits a set of sequence operations.
sequence_receptor::finished	Sets the state variable **receptor_finished** to **true**.
sequence_receptor::get_hostname	Returns the host name of the remote indexing node.

3.4.4.1 column_backup::get_row_id

The **get_row_id** method is specified in section 3.3.4.1.

3.4.4.2 column:backup::get_hostname

The **get_hostname** method is specified in section 3.3.4.2.

3.4.4.3 column_backup::submit_sequence

The **submit_sequence** method is specified in section 3.3.4.3.

3.4.4.4 column_backup::commit_sequence

The **commit_sequence** method is specified in section 3.3.4.4.

3.4.4.5 column_backup::abort_sequence

The **abort_sequence** method is specified in section 3.3.4.5.

3.4.4.6 column_backup::activate_index_set

The **activate_index_set** method is specified in section 3.3.4.6.

3.4.4.7 sequence_receptor::submit_sequence

The **submit_sequence** method is specified in section 3.3.4.7.

3.4.4.8 sequence_receptor::finished

The **finished** method is specified in section 3.3.4.8.

3.4.4.9 sequence_receptor::get_hostname

The **get_hostname** method is specified in section 3.3.4.9.

3.4.5 Timer Events

None.

3.4.6 Other Local Events

None.

4 Protocol Examples

4.1 Recover a Backup Indexing Node

This example illustrates how this protocol is used in a user scenario where a lagging backup indexing node synchronizes with the master indexer node by requesting missing sequence operations.

The recovery process typically involves the following messages:

1. The backup indexing node resolves the master indexer node in the name server.
2. The backup indexing node requests the range endpoint values of the sequence operations stored on the master indexer node.
3. The backup indexing node requests an asynchronous transfer of missing sequence operations from the master indexer node.
4. The master indexer node submits the missing sequence operations to the backup indexing node.
5. The master indexer node submits an end of transmission message to the backup indexing node.

The scenario presupposes that the master indexer node has already created a server object implementing the **operation_sequence_store** interface and bound the server object in a name server using a logical name known to the backup indexing node.

When a backup indexing node is started, it first resolves the **content_operation_sequence_store** interface in the name server. The backup indexing node then proceeds by requesting the range of sequence operations the master indexer node has stored in its fault-tolerant storage. If the backup indexing node does not possess all of the sequence operations on the master indexer node, the backup indexing node must request an asynchronous transfer. Before this request can be made, the backup indexing node must create a server object implementing the **sequence_receptor** interface, so that the backup indexing node can pass a reference to the recipient of the sequence operations. Using the client proxy to the **sequence_receptor**, the master indexer node submits the missing sequence operations to the backup indexing node. After the transfer of sequence operations, a message indicating that all operations have been submitted is sent to the backup indexing node.

4.1.1 Recovery Code

```
SET server_object_name TO "esp/clusters/webcluster/indexing/indexer-0-0/opr_seq_store"

SET server_object_type TO "rtsearch::content_operation_sequence_store"

SET server_object_version TO 5.6

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

CALL op_seq_store_client_proxy.get_stored_sequences RETURNING remote_stored_sequences

IF local_stored_sequences.high_sequence_id < remote_stored_sequences.high_sequence_id THEN
  SET sequence_receptor_server_object TO INSTANCE OF sequence_receptor
    SERVER OBJECT

  CALL op_seq_store_client_proxy.request_sequences WITH   sequence_receptor_server_object,
    local_stored_sequences.high_sequence_id + 1, remote_stored_sequences.high_sequence
ENDIF
```

```
REPEAT
  PRINT "Waiting for master indexer node to submit missing sequence operations"
UNTIL sequence_receptor_server_object.finished HAS BEEN CALLED
```

4.1.2 Recovery Sequence Diagram

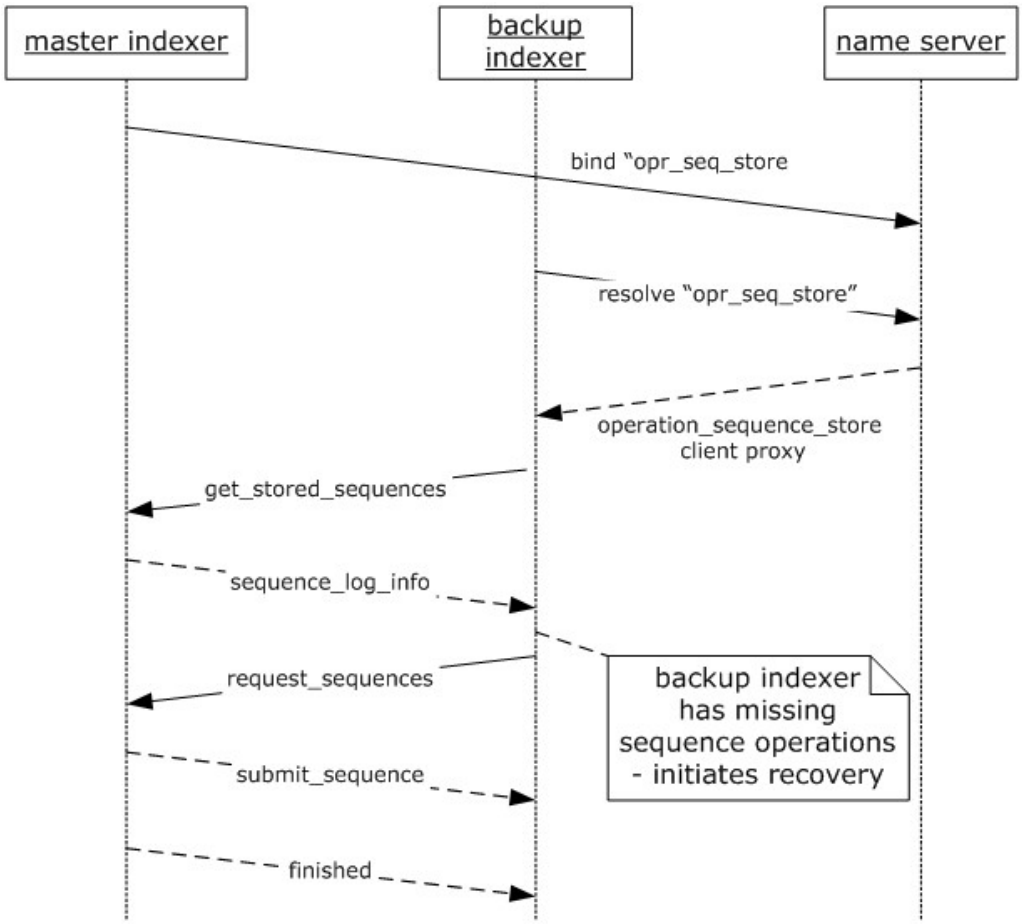

Figure 4: Recovery sequence diagram

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full FSIDL

For ease of implementation, the full FSIDL and Cheetah specifications are provided in the following sections.

6.1 FSIDL

```
module cht {
  module rtsmessages {
    typedef sequence<octet> cheetah;
    typedef cheetah sequence_log_info;
    typedef cheetah content_operation_sequence;
  };
};

module interfaces {
  module rtsearch {
    interface column_backup {
      long get_row_id();
      string get_hostname();
      boolean submit_sequence(in cht::rtsmessages::content_operation_sequence            seq,
                              in string collection_name);
      void commit_sequence();
      void abort_sequence();
      void activate_index_set();
    }

    interface column_master {
      long get_row_id();
      void register_backup_node(in column_backup backup_interface,
                                in long row_id);
      boolean has_backup_node(in long row_id);
      void check_backup_nodes();
      void abdicate();
      boolean connect_receiver(in file_receiver receiver,
                               in string hostname,
                               in long port);
      boolean disconnect_receiver(in string hostname, in long port);
    }

    interface sequence_receptor {
      void submit_sequence(in cht::rtsmessages::content_operation_sequence seq);
      void finished();
      string get_hostname();
    }

    interface content_operation_sequence_store {
      boolean is_master();
      cht::rtsmessages::sequence_log_info get_stored_sequences();
      boolean has_sequence_id(in long long sequence_id);
      void request_sequences(in sequence_receptor receptor,
                             in long long from_sequence_id,
                             in long long to_sequence_id);
      long get_row_id();
      string get_hostname();
      long long get_highest_sequence_id();
      long long get_lowest_sequence_id();
    }
```

```
  }
}
```

6.2 Cheetah

```
entity sequence_log_info {
  attribute longint low_sequence_id;
  attribute longint high_sequence_id;
  attribute longint processed_sequence_id;
};

entity sequence_operation {
  attribute longint sequence_number;
  attribute longint operation_id;
};

entity empty_operation : sequence_operation {
};

entity fixml_invalidation : sequence_operation {
  attribute string document_id;
  attribute int file_id;
  attribute int magic_idx;
  attribute bool is_update;
};

entity remove_collection : sequence_operation {
};

entity fixml_append : sequence_operation {
  attribute string document_id;
  attribute string document_content;
  attribute int file_id;
  attribute int magic_idx;
  attribute bool is_update;
};

entity remdoclist : sequence_operation {
  attribute string  document_id;
  attribute int     old_file_id;
  attribute int     new_file_id;
};

entity exclusionlist : sequence_operation {
  attribute string  document_id;
  attribute int     old_file_id;
};

entity document_error : sequence_operation {
  attribute string document_id;
  attribute int error_code;
  attribute int action;
  attribute string subsystem;
  attribute string error_message;
}

entity content_operation_sequence {
```

```
    attribute int session_id;
    attribute string document_collection;
    attribute longint low_sequence_id;
    attribute longint high_sequence_id;
    collection sequence_operation operations;
  };
```

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

This section identifies changes that were made to the [MS-FSIFT] protocol document between the November 2010 and December 2010 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- Changes made for template compliance.
- Removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type "Editorially updated."

Some important terms used in revision type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change Type
7 Appendix B: Product Behavior	Updated the list of applicable product versions.	N	Content updated.

9 Index

D

E

F

G

I

L

M

N

O

P

R

S

T

V