[MS-FSID]:
Indexing Distribution Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	Major	Updated and revised the technical content
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.
01/20/2012	1.5	Minor	Clarified the meaning of the technical content.
04/11/2012	1.5	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	1.5	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Indexing Distribution Protocol is used between two components in the indexing service, the **indexing dispatcher** node and the **indexing node**. It enables a session-based connection as part of an extended item feeding chain, with asynchronous callback messages reporting status back to the feeding client.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

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

attribute
Hypertext Transfer Protocol (HTTP)
UTF-8
XML

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

abstract object reference (AOR)
base port
callback message
Cheetah
Cheetah checksum
client proxy
content collection
FAST Index Markup Language (FIXML)
FAST Search Interface Definition Language (FSIDL)
host name
index column
index partition
indexing dispatcher
indexing node
item
item processing
name server
node
search service application

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

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the technical documents, which are updated frequently. References to other documents include a publishing year when one is available.

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-FSICFG] Microsoft Corporation, "Indexer Configuration File Format".

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

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

1.3 Protocol Overview (Synopsis)

This protocol enables an indexing dispatcher node to feed **items** to one or more indexing nodes, as part of an extended session-based item feeding chain. The process has three steps: Creating a new session, feeding items using the session, and retrieving status information about the items.

A session is created using the **session_factory** interface. That **session** interface is used to feed items, while asynchronous status information is returned using the **callback** interface.

The extended item feeding chain sequence of which this protocol is a part consists of the feeding client, content distributor, document processor, indexing dispatcher, and indexing nodes. This protocol defines the communication between the indexing dispatcher and indexer components in the extended feeding chain, as illustrated in the following figure.

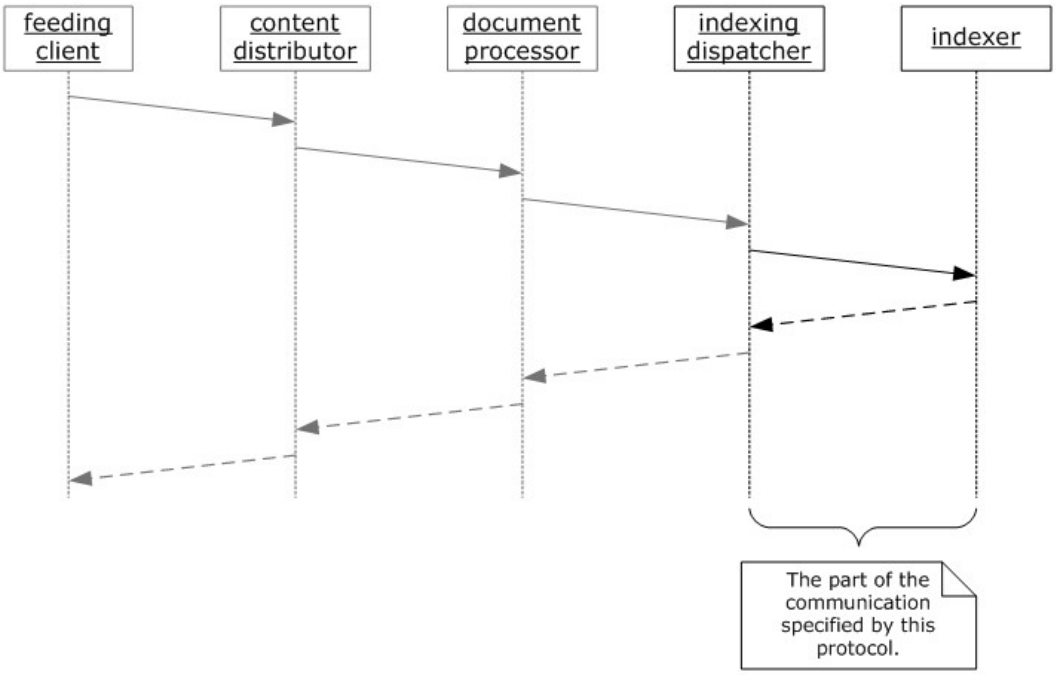

Figure 1: Extended item feeding chain

Both the indexing dispatcher node and indexing node have dual roles as protocol clients and protocol servers with regard to this protocol. For the **session_factory** and **session** interfaces, the indexing dispatcher node acts as the protocol client and the indexing node as the protocol server. For the **callback** interface, the roles are reversed, and the indexing node is the protocol client while the indexing dispatcher node is the protocol server.

1.4 Relationship to Other Protocols

This protocol uses Middleware, an **HTTP** based protocol, as described in [MS-FSMW]. Custom data types are encoded over the wire using **Cheetah**, as described in [MS-FSCHT].

The following figure shows the underlying messaging and transport stack used by the protocol:

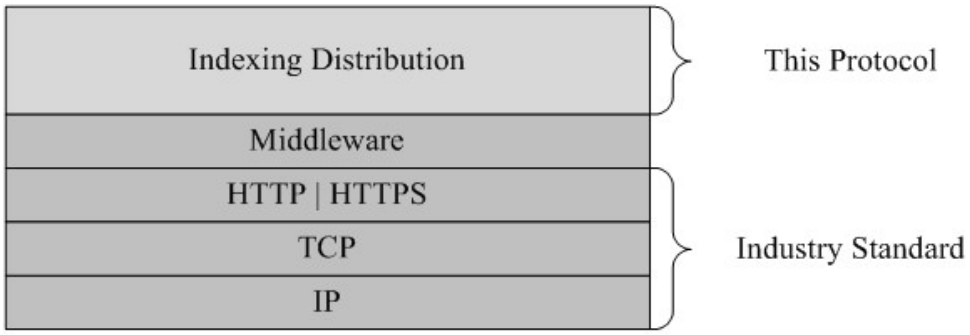

Figure 2: This protocol in relation to other protocols

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 designed to be part of an item feeding chain between an item feeding client and an indexing service, with the ability to send information about item status back up the feeding chain using asynchronous **callback messages**. This protocol is designed to be used in the item feeding chain segment between an indexing dispatcher node and one or more indexing nodes.

1.7 Versioning and Capability Negotiation

Capability negotiation: The Middleware protocol is connectionless, but the correct interface version is specified in every message that is transmitted using the Middleware protocol. See sections 3.1.3 and 3.2.3 for the specific version numbers.

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 value –211918678. The type identifier for the Cheetah entities MUST be integers as specified in the following table.

Cheetah entity	Type identifier
error	15
processing_error	21
format_error	22
xml_error	55
utf8_error	54
server_unavailable	47
operation_dropped	36
operation_lost	37
indexing_error	23
invalid_content	33
resource_error	46
unknown_document	51
warning	6
operation	7
operation_set	38
operation_status_info	40
operation_status_info_set	41
document_id	11
key_value_pair	0
key_value_collection	1
string_attribute	26
integer_attribute	28

Cheetah entity	Type identifier
bytearray_attribute	4
internal_partial_update_operation	25
remove_nodes	44
insert_xml	27
string_replace	50
internal_partial_update	32
document	12
no_operation	8
clear_collection	9
failed_operation	18
remove_operation	45
update_operation	52

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

2.2.1 cht::documentmessages::action

The **action** Cheetah enumeration contains available suggested actions for feeding clients as part of an **error** message, as specified in section 2.2.3. The syntax is as follows.

```
enum action {
  resubmit,
  limited_resubmit,
  drop,
  terminate
};
```

Valid values MUST be as listed in the following table.

Identifier	Action	Description
0	**resubmit**	MUST NOT be used.
1	**limited_resubmit**	MUST NOT be used.
2	**drop**	Drop the item from the index.
3	**terminate**	MUST NOT be used.

2.2.2 cht::documentmessages::operation_state

The **operation_state** Cheetah enumeration contains the available operation state values as part of an **operation_status_info** message, as specified in section 2.2.18. The syntax is as follows.

```
enum operation_state {
  unknown,
  received,
  secured,
  completed,
  lost
};
```

Valid values MUST be as listed in the following table.

Identifier	State	Description
0	unknown	Unknown state.
1	received	Operation received.
2	secured	Operation persisted to disk, but change not yet reflected in searchable index.
3	completed	Operation persisted to disk, and change reflected in searchable index.
4	lost	MUST NOT be used.

2.2.3 cht::documentmessages::entity error

The **error** Cheetah entity contains error information for a specific operation identifier. The syntax is as follows.

```
root entity error {
  attribute int error_code;
  attribute action suggested_action;
  attribute string description;
  attribute string subsystem;
  attribute int session_id;
  attribute longint operation_id;
  collection string arguments;
};
```

error_code: An integer containing an error code. Valid values MUST be as listed in the following table.

Error code	Description
1	Missing **attribute (1)**.
2	Generic error.
3	Unknown document.
4	Indexer suspended. Set by the **session::process** server object when a batch of operations is received while the **docapi suspended** state is **true**.
5	**FAST Index Markup Language (FIXML)** write error. Failed to persist the batch of item operations.
6	Unknown **content collection**.

Error code	Description
7	Error during partial update.

suggested_action: An **action** Cheetah entity, as specified in section 2.2.1, specifying the recommended action to perform by a feeding client to correct the operation error.

description: A string containing a verbal description of the error.

subsystem: A string that contains the name of the subsystem.

session_id: An integer that uniquely identifies the session used.

operation_id: An integer that uniquely identifies the operation to which this error belongs.

arguments: A string, which MUST be empty.

2.2.4 cht::documentmessages::processing_error

The **processing_error** Cheetah entity is a subclass of the **error** Cheetah entity specified in section 2.2.3. It is a common super class used by other error entities, and describes errors that occur during processing of an item operation. The syntax is as follows.

```
entity processing_error : error {
  attribute string processor;
};
```

processor: A string that contains the name of the **item processing** stage in which the error occurred.

2.2.5 cht::documentmessages::format_error

The **format_error** Cheetah entity is a subclass of **processing_error** Cheetah entity specified in section 2.2.4. It is a common super class used by other **error** entities when an item operation has an illegal format. The syntax is as follows.

```
entity format_error : processing_error {
};
```

2.2.6 cht::documentmessages::xml_error

The **xml_error** Cheetah entity is a subclass of the **format_error** Cheetah entity specified in section 2.2.5. It is used when an item operation has an illegal **XML** format. This syntax is as follows.

```
entity xml_error : format_error {
};
```

2.2.7 cht::documentmessages::utf8_error

The **utf8_error** Cheetah entity is a subclass of the **format_error** Cheetah entity specified in section 2.2.5. It is used when an item operation has an illegal **UTF-8** format. The syntax is as follows.

```
entity utf8_error : format_error {
};
```

2.2.8 cht::documentmessages::server_unavailable

The **server_unavailable** Cheetah entity is a subclass of the **processing_error** Cheetah entity specified in section 2.2.4. It is used when an item processing stage is unable to connect to a server during processing of an item operation. The syntax is as follows.

```
entity server_unavailable : processing_error {
};
```

2.2.9 cht::documentmessages::operation_dropped

The **operation_dropped** Cheetah entity is a subclass of the **processing error** Cheetah entity specified in section 2.2.4. It is used when an item processing stage has identified an item operation that should not be sent to the indexing node. The syntax is as follows.

```
entity operation_dropped : processing_error {
};
```

2.2.10 cht::documentmessages::operation_lost

The **operation_lost** Cheetah entity is a subclass of the **error** Cheetah entity specified in section 2.2.3. It is used when an item operation has been lost during processing or indexing. The syntax is as follows.

```
entity operation_lost : error {
};
```

2.2.11 cht::documentmessages::indexing_error

The **indexing_error** Cheetah entity is a subclass of the **error** Cheetah entity specified in section 2.2.3. It is a common superclass used by other **error** entities. It is used when an error has occurred during indexing of an item operation. The syntax is as follows.

```
entity indexing_error : error {
};
```

2.2.12 cht::documentmessages::invalid_content

The **invalid_content** Cheetah entity is a subclass of the **indexing_error** Cheetah entity specified in section 2.2.11. It is used by the indexing when an item operation has invalid content. The syntax is as follows.

```
entity invalid_content : indexing_error {
};
```

2.2.13 cht::documentmessages::resource_error

The **invalid_content** Cheetah entity is a subclass of the **indexing_error** Cheetah entity specified in section 2.2.11. It is used by indexing when a resource error occurs on the indexing node during indexing of an item operation. The syntax is as follows.

```
entity resource_error : indexing_error {
};
```

2.2.14 cht::documentmessages::unknown_document

The **unknown_document** Cheetah entity is a subclass of the **indexing_error** Cheetah entity specified in section 2.2.11. It is used during indexing when a **remove_operation** Cheetah entity, as specified in section 2.2.35, refers to an item that does not exist in the index. The syntax is as follows.

```
entity unknown_document : indexing_error {
};
```

2.2.15 cht::documentmessages::warning

The **warning** Cheetah entity contains warning information for a specific operation identifier. The syntax is as follows.

```
root entity warning {
  attribute int warning_code;
  attribute string description;
  attribute string subsystem;
  attribute int session_id;
  attribute longint operation_id;
};
```

warning_code: An integer containing the warning code. Valid values MUST be as listed in the following table.

Code	Description
1	All **index partitions** are full.
2	Indexing suspended. The **indexing suspended** state is **true**.

description: A string containing a verbal description of the warning.

subsystem: A string that contains the name of the subsystem, which is "indexing".

session_id: An integer that uniquely identifies the session used.

operation_id: An integer that uniquely identifies the operation to which this warning belongs.

2.2.16 cht::documentmessages::operation

The **operation** Cheetah entity is a common superclass used by other **operation** types, such as **remove_operation** specified in section 2.2.35. The syntax is as follows.

```
entity operation {
  attribute longint id;
  collection warning warnings;
};
```

id: An integer that uniquely identifies the operation. This MUST be greater than or equal to zero.

warnings: A collection of **warning** Cheetah entities, as specified in section 2.2.15, which contains warnings for the operation identified by **id**.

2.2.17 cht::documentmessages::operation_set

The **operation_set** Cheetah entity contains a set of **operation** Cheetah entities, as specified in section 2.2.16, along with the identifier of the last finished operation. The syntax is as follows.

```
root entity operation_set {
  attribute longint completed_op_id;
  collection operation operations;
};
```

completed_op_id: An integer containing the highest operation identifier in sequence that is known to be finished by the feeding client.

operations: A collection of **operation** Cheetah entities, as specified in section 2.2.16.

2.2.18 cht::documentmessages::operation_status_info

The **operation_status_info** Cheetah entity contains status information about a batch of operations. The syntax is as follows.

```
root entity operation_status_info {
  attribute longint first_op_id;
  attribute longint last_op_id;
  attribute operation_state state;
  attribute string subsystem;
  collection error errors;
  collection warning warnings;
};
```

first_op_id: An integer containing the operation identifier of the first operation in the operation batch. This MUST be greater than or equal to zero, and smaller than or equal to **last_op_id**.

last_op_id: An integer containing the operation identifier of the last operation in the operation batch. This MUST be greater than or equal to zero, and greater than or equal to **first_op_id**.

state: An **operation_state** type containing the state of the operation batch, as specified in section 2.2.2.

subsystem: A string that contains the name of the subsystem, which is "indexing".

errors: A collection of **error** objects, as specified in section 2.2.3, that contains errors for operations within the batch.

warnings: A collection of **warning** objects, as specified in section 2.2.15, that contains warnings for operations within the batch.

2.2.19 cht::documentmessages::operation_status_info_set

The **operation_status_info_set** Cheetah entity contains operations status information from a set of operation batches. The syntax is as follows.

```
root entity operation_status_info_set {
  collection operation_status_info status;
};
```

status: A collection of **operation_status_info** objects, as specified in section 2.2.18.

2.2.20 cht::documentmessages::document_id

The **document_id** Cheetah entity uniquely identifies an item. The syntax is as follows.

```
root entity document_id {
  attribute string id;
  collection key_value_pair routing_attributes;
};
```

id: A string uniquely identifying the item.

routing_attributes: A collection of **key_value_pair** Cheetah entities containing information used for routing the item to a specific **index column**. The routing is implementation specific of the higher level application.

2.2.21 cht::documentmessages::key_value_pair

```
entity key_value_pair {
  attribute string key;
};
```

The **key_value_pair** Cheetah entity is a common super class used to form relation structures between a key and data of various types. It has the following member:

key: A string containing the key.

2.2.22 cht::documentmessages::key_value_collection

The **key_value_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity specified in section 2.2.22. The syntax is as follows.

```
entity key_value_collection : key_value_pair {
  collection key_value_pair values;
};
```

values: A collection of **key_value_pair** objects, as specified in section 2.2.22.

2.2.23 cht::documentmessages::string_attribute

The **string_attribute** Cheetah entity uses the **key_value_pair** Cheetah entity, specified in section 2.2.22, to form a relation structure between a string key and a string value. The syntax is as follows.

```
entity string_attribute : key_value_pair {
  attribute string value;
};
```

value: A string containing the value.

2.2.24 cht::documentmessages::integer_attribute

The **integer_attribute** Cheetah entity uses the **key_value_pair** Cheetah entity, specified in section 2.2.22, to form a relation structure between an integer key and a string value. The syntax is as follows.

```
entity integer_attribute : key_value_pair {
  attribute int value;
};
```

value: An integer containing the value.

2.2.25 cht::documentmessages::bytearray_attribute

The **bytearray_attribute** Cheetah entity uses the **key_value_pair** Cheetah entity, specified in section 2.2.22, to form a relation structure between a bytearray key and a string value. The syntax is as follows.

```
entity bytearray_attribute : key_value_pair {
  attribute bytearray value;
};
```

value: A byte array containing the value.

2.2.26 cht::documentmessages::internal_partial_update_operation

The **internal_partial_update_operation** Cheetah entity is a super class used by other partial update operations. The syntax is as follows.

```
entity internal_partial_update_operation {
};
```

2.2.27 cht::documentmessages::remove_nodes

The **remove_nodes** Cheetah entity is a subtype of the **internal_partial_update_operation** Cheetah entity specified in section 2.2.26. It is used to remove nodes from an XML structure. The syntax is as follows.

```
entity remove_nodes : internal_partial_update_operation {
  attribute string node_selection;
};
```

node_selection: A string containing the XPath of the items to remove.

2.2.28 cht::documentmessages::insert_xml

The **insert_xml** Cheetah entity is a subtype of the **internal_partial_update_operation** Cheetah entity specified in section 2.2.26. It is used to insert a **string_attribute**, as specified in section 2.2.23, into an XML structure. The syntax is as follows.

```
entity insert_xml : internal_partial_update_operation {
  attribute string_attribute key_value;
};
```

key_value: The key-value pair to insert.

2.2.29 cht::documentmessages::string_replace

The **string_replace** Cheetah entity is a subtype of the **internal_partial_update_operation** Cheetah entity specified in section 2.2.26. It is used to replace values in an XML structure. The syntax is as follows.

```
entity string_replace : internal_partial_update_operation {
  attribute string_attribute key_value;
};
```

key_value: A **string_attribute**, as specified in section 2.2.23, where the **key** defines the XPath for the item to be replaced with **value**.

2.2.30 cht::documentmessages::internal_partial_update

The **internal_partial_update** Cheetah entity is a subtype of the **operation** Cheetah entity specified in section 2.2.16. It is used to perform update operations on a specific item. The syntax is as follows.

```
root entity internal_partial_update : operation {
  attribute document_id doc_id;
  collection internal_partial_update_operation operations;
};
```

doc_id: A **document_id**, as specified in section 2.2.20, uniquely identifying the item.

operations: A collection of **internal_partial_update_operation** objects, as specified in section 2.2.26.

2.2.31 cht::documentmessages::document

The **document** Cheetah entity contains information about one specific item. The syntax is as follows.

```
root entity document {
  attribute document_id doc_id;
  collection key_value_pair document_attributes;
};
```

doc_id: A **document_id**, as specified in section 2.2.20, uniquely identifying the item.

document_attributes: A collection of attributes describing the item to be indexed.

2.2.32 cht::documentmessages::no_operation

The **no_operation** Cheetah entity is used to replace an operation deleted by a filter system earlier in the feeding chain, so that the number of operations is unaltered. The syntax is as follows.

```
entity no_operation : operation {
};
```

2.2.33 cht::documentmessages::clear_collection

The **clear_collection** Cheetah entity is a subtype of the **operation** Cheetah entity specified in section 2.2.16. It is used to clear all other sessions working on the same collection as the current session, and deletes all the items in the collection. The syntax is as follows.

```
entity clear_collection : operation {
};
```

2.2.34 cht::documentmessages::failed_operation

The **failed_operation** Cheetah entity is a subtype of the **operation** Cheetah entity specified in section 2.2.16. It contains information about an operation that has failed earlier in the feeding chain. It is used as a placeholder for the original failed operation to keep the operation sequence complete. The syntax is as follows.

```
entity failed_operation : operation {
  attribute string subsystem;
  attribute operation_state state;
  attribute string operation_type;
  attribute document_id doc_id;
  attribute error err;
};
```

subsystem: A string that contains the name of the subsystem where the failed operation was detected.

state: An **operation_state** type containing the state of the operation, as specified in section 2.2.2.

operation_type: A string that contains a verbal description of the operation type.

doc_id: A **document_id**, as specified in section 2.2.20, uniquely identifying the item.

err: An **error**, as specified in section 2.2.3.

2.2.35 cht::documentmessages::remove_operation

The **remove_operation** Cheetah entity is used to remove a specific item from the index. The syntax is as follows.

```
entity remove_operation : operation {
  attribute document_id doc_id;
};
```

doc_id: A **document_id**, as specified in section 2.2.20, uniquely identifying the item.

2.2.36 cht::documentmessages::update_operation

The **update_operation** Cheetah entity is a subtype of the **operation** Cheetah entity specified in section 2.2.16. It is used to update a specific item in the index, or to add it if the item does not already exist in the index. The syntax is as follows.

```
entity update_operation : operation {
  attribute document doc;
};
```

docs: A **document** Cheetah entity, as specified in section 2.2.31, containing updated information about the item.

3 Protocol Details

This protocol is part of an extended session based item feeding chain between an external feeding client and an indexing service. The indexing dispatcher **node** communicates synchronously with an indexing node, setting up a new session using the **session_factory** interface, then feeding items using this **session** interface. Asynchronous status messages about items are sent from the indexing node to the indexing dispatcher node using the **callback** interface, as shown in the following figure.

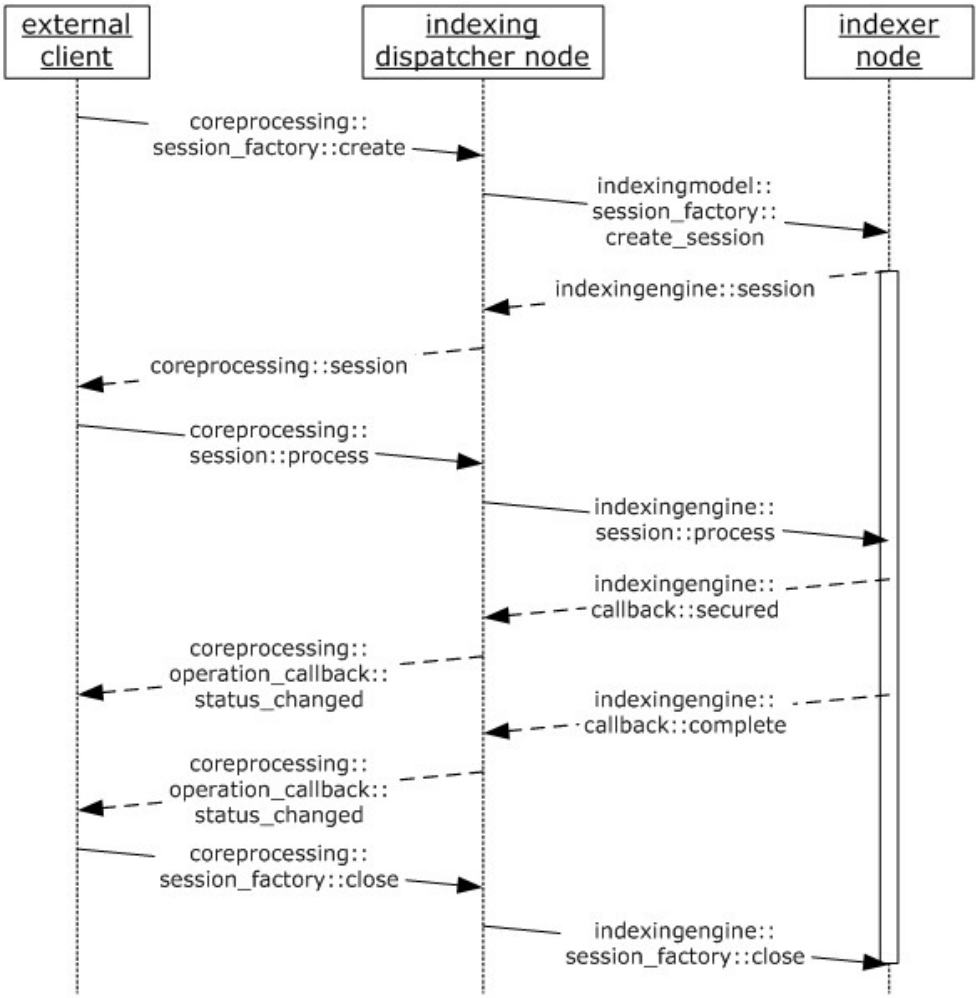

Figure 3: Sessions and callback messages

This protocol consists of three protocol interfaces, **session_factory**, **session**, and **callback**. For **session_factory** and **session**, the indexing dispatcher node acts as a protocol client, and the indexing node as the protocol server. For **callback**, the roles are reversed, and the indexing node is the protocol client, while the indexing dispatcher node is the protocol server.

3.1 indexingengine::session_factory Server Details

An indexing node serving the **session_factory** interface receives messages from an indexing dispatcher node, and manages the creation, administration and closing of **session** objects.

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 indexing node, acting as protocol server for the **session factory** interface, MUST maintain the following states for each session instance:

session server: A state containing a **session** server object, enabling an indexing node to receive items on the **session** interface.

callback client: A state containing a **callback client proxy**, enabling an indexing node to send asynchronous callback messages on the **callback** interface.

session id: A state containing a session identifier.

3.1.2 Timers

None.

3.1.3 Initialization

The protocol server MUST use the Middleware **bind** method to register a **session_factory** 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 that contains the value "esp/clusters/webcluster/indexing/indexer-C/sessionfactory", where *C* is the index column number.

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** plus 390.

interface_type: This MUST be a string that contains the value "indexing::session_factory**"**.

interface_version: This MUST be a string that contains the value "5.7".

3.1.4 Message Processing Events and Sequencing Rules

This interface includes the methods described in the following table.

Method	Description
create_session	Creates a new **session** server object identified by a session identifier, and returns a **session** client proxy.
get_highest_session_id	Returns the highest session identifier number used so far when creating new sessions using **session_factory::create_session.**

Method	Description
close	Closes a specific session.
flush_session	Resets the state of a specific session.

3.1.4.1 create_session

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

```
session create_session(in long session_id, in string collection,
    in callback the_callback)
raises (shutdown_exception, invalid_input_exception)
```

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

collection: A string that contains the name of the content collection to use.

the_callback: A callback message client proxy, as specified in section 3.5.

Return values: A **session** client proxy, as specified in section 3.3.

Exceptions:

The exception **shutdown_exception** MUST be raised if the search component is in a shutdown mode.

The exception **invalid_input_exception** MUST be raised if the length of the **collection** name is larger than 16.

The protocol server MUST store **session_id** in the **session id** state.

The protocol server MUST store **session** in the **session server** state, and activate the **session** server object.

The protocol server MUST store **the_callback** in the **callback client** state.

Based on information received by the feeding client in callback messages from the indexing service, it is possible for the feeding client to re-feed certain items, for example, if the session used during the feeding died. In that case, the feeding client would re-use the session identifier used when originally feeding the items when calling **create_session**, and not use a new session identifier.

3.1.4.2 get_highest_session_id

The **get_highest_session_id** method returns the highest session identifier number used so far when creating new sessions using **session_factory::create_session**, as specified in section 3.1.4.1.

```
long get_highest_session_id()
```

Return values: An integer that MUST be greater than or equal to zero, and the value of the **session id** state with the highest value among session instances.

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

3.1.4.3 close

The **close** method closes down a specific session.

```
void close(in long session_id)
```

session_id: The identifier of the session to close. This MUST be an integer greater than or equal to zero.

Return values: None.

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

This method MUST deactivate the **session** server object in the **session server** state in the session instance where the **session id** state matches **session_id**.

3.1.4.4 flush_session

The **flush_session** method resets the state of a specific session.

```
void flush_session(in long session_id)
```

session_id: The identifier of the session to flush. This MUST be an integer greater than or equal to zero.

Return values: None.

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

This method MUST deactivate and reset the state of the **session** server object in the **session server** state in the session instance where the **session id** state matches **session_id**.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 indexingengine::session_factory Client Details

An indexing dispatcher node uses the **session factory** interface to create and administrate **session** server objects on an indexing node.

3.2.1 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 indexing dispatcher node, acting as protocol client for the **session factory** interface, MUST maintain the following states for each session instance:

session client holder: A state containing a collection of **session** client proxies to indexing nodes, used by the indexing dispatcher node to split a batch of operations received from a feeding client into smaller batches and route them to specific indexing nodes on individual index columns. The routing is implementation specific of the higher level application

operations callback holder: A state containing information about all operations sent to indexing nodes using the **session::process** message, enabling the indexing dispatcher node to detect when callback messages have been received for all operations sent.

3.2.2 Timers

None.

3.2.3 Initialization

The client side of the **session_factory** protocol interface MUST use the Middleware **resolve** method to find the client proxy to the **session_factory** 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 that contains the value "esp/clusters/webcluster/indexing/indexer-C/sessionfactory", where *C* is the index column number.

interface_type: This MUST be a string that contains the value "indexingengine::session_factory".

version: This MUST be a string that contains the value "5.7".

The client side of the **session_factory** protocol interface MUST initialize a **callback** server object for each session instance by using an AOR, as specified in [MS-FSMW] section 2.2.14, containing the following values:

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 plus 390.

interface_type: This MUST be a string that contains the value "indexingengine::callback".

interface_version: This MUST be a string that contains the value "5.0".

3.2.4 Message Processing Events and Sequencing Rules

This interface includes the methods described in the following table.

Method	Description
create_session	Creates a new **session** server object identified by a session identifier, and returns a **session** client proxy.

Method	Description
get_highest_session_id	Returns the highest session identifier number used so far when creating new sessions using **session_factory::create_session.**
close	Closes a specific session.
flush_session	Resets the state of a specific session.

3.2.4.1 create_session

The **create_session** method is specified in section 3.1.4.1. When the **session factory** protocol client sends a **create_session** method, it MUST send along the **callback** server object created during initialization of the **session_factory** interface, as specified in section 3.2.3.

3.2.4.2 get_highest_session_id

The **get_highest_session_id** method is specified in section 3.1.4.2.

3.2.4.3 close

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

3.2.4.4 flush_session

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

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 indexingengine::session Server Details

The **session** server object accepts calls to process incoming items, in addition to providing information about the session.

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 **session** protocol server MUST maintain the following states:

docapi suspended: A Boolean state informing whether or not the protocol server is accepting operations received from the **session::process** method or not.

indexing suspended: A Boolean state informing whether or not the indexing service is suspended. When suspended, it does not process operations or build new indexes.

The **session** protocol server MUST maintain the following state for each session instance:

processed operation id: An integer state containing the identifier of the last in-sequence operation submitted to a session by **session::process**.

In addition, the **session** protocol server MUST have access to the states managed by the **session factory** protocol server.

3.3.2 Timers

None.

3.3.3 Initialization

The **session** server object MUST be created and activated by **session_factory::create_session**, as specified in section 3.1.4.1, and stored in the **session server** state.

3.3.4 Message Processing Events and Sequencing Rules

This interface includes the methods described in the following table.

Method	Description
process	Processes a sequence of operations.
get_id	Returns the identifier of the session.
get_last_operation_id	Returns the identifier of the last in-sequence operation submitted to the session.

3.3.4.1 process

The **process** method processes a sequence of operations.

```
boolean process(in long long last_operation_in_sequence,
    in cht::documentmessages::operation_set operations)
raises (batch_format_error, resource_error)
```

last_operation_in_sequence: The identifier of the last operation in the sequence. This MUST be an integer greater than or equal to zero.

operations: An **operation_set** Cheetah entity, as specified in section 2.2.17, containing a sequence of operations.

Return values: MUST be **true** if ok for protocol client to send another sequence of operations, and MUST be **false** if the protocol server is not capable of handling more data for the moment. The already received operation sequence MUST be processed if **false** is returned.

Exceptions:

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

The exception **resource_error** MUST be raised when the operation is not a **remove operation**, as specified in section 2.2.35, and one or more of the following conditions are **true**:

- The free available disk space on the disk holding the FIXML files is less than the value **diskspaceMBWarning**, as specified in [MS-FSICFG].
- The minimum free available disk space on the disks holding the indexes is less than the value **diskspaceMBWarning**, as specified in [MS-FSICFG].
- The fault tolerance feature, as specified in [MS-FSICFG], is enabled, and the free available disk space in the data directory is less than the value **diskspaceMBWarning**, as specified in [MS-FSICFG].

The protocol server MUST process the commands contained in the **operations** parameter if the **docapi suspended** state is **false**. If the **docapi suspended** state is **true**, the protocol server MUST send a **callback::secure** message to the protocol client with **error_code** 4, as specified in section 2.2.3.

The protocol server MUST store the value of the last sequential operation received to the **processed operation id** state.

3.3.4.2 get_id

The **get_id** method returns the identifier of the session.

```
long get_id()
```

Return values: An integer that MUST be greater than or equal to zero.

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

The protocol server MUST return the value of the **session id** state for the session instance.

3.3.4.3 get_last_operation_id

The **get_last_operation_id** method returns the identifier of the last in-sequence operation submitted to the session.

```
long long get_last_operation_id()
```

Return values: An integer that MUST be greater than or equal to zero.

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

The protocol server MUST return the value of the **processed operation id** state for the session instance.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 indexingengine::session Client Details

An indexing dispatcher node, acting as **session** protocol client, sends calls to process items on an indexing node.

3.4.1 Abstract Data Model

None.

3.4.2 Timers

None

3.4.3 Initialization

The client side of the **session** protocol interface MUST use the **session** client proxy stored in the **session client** state for the session instance.

3.4.4 Message Processing Events and Sequencing Rules

This interface includes the methods described in the following table.

Method	Description
process	Processes a sequence of operations.
get_id	Returns the identifier of the session.
get_last_operation_id	Returns the identifier of the last in-sequence operation submitted to the session.

3.4.4.1 process

The **process** method is specified in section 3.3.4.1. When the **session** protocol client sends a **process** method, it MUST update the **operations callback holder** state with status information about the operations sent.

3.4.4.2 get_id

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

3.4.4.3 get_last_operation_id

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

3.4.5 Timer Events

None.

3.4.6 Other Local Events

None.

3.5 indexingengine::callback Server Details

An indexing dispatcher node serving the **callback** interface receives asynchronous status callback messages from an indexing node.

3.5.1 Abstract Data Model

The **callback** protocol server MUST have access to the states managed by the **session factory** protocol client, as specified in section 3.2.1.

3.5.2 Timers

None.

3.5.3 Initialization

The **callback** server object MUST be created and activated by the **session factory** protocol client, as specified in section 3.2.3.

3.5.4 Message Processing Events and Sequencing Rules

This interface includes the methods described in the following table.

Method	Description
secure	Contains status information about operations within a range which have been secured.
complete	Contains status information about operations within a range which have been finished.

3.5.4.1 secure

The **secure** method contains status information about operations within a range, which have been persisted to disk in the indexing service.

```
void secure(in cht::documentmessages::operation_status_info status)
```

status: An **operation_status_info** Cheetah entity, as specified in section 2.2.18.

Return values: None.

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

This method MUST update the **operations callback holder** state with information about the received operations status.

3.5.4.2 complete

The **complete** method contains status information about operations within a range that have been made searchable by the **search service application**.

```
void complete(in cht::documentmessages::operation_status_info status)
```

status: An **operation_status_info** Cheetah entity, as specified in section 2.2.18.

Return values: None

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

This method MUST update the **operations callback holder** state with information about the received operations status.

3.5.5 Timer Events

None.

3.5.6 Other Local Events

None.

3.6 indexingengine::callback Client Details

An indexing node sends asynchronous status callback messages about operations received in **session::process** messages to an indexing dispatcher node.

3.6.1 Abstract Data Model

The **callback** protocol client MUST have access to the states managed by the **session factory** protocol server, as specified in section 3.1.1.

3.6.2 Timers

None.

3.6.3 Initialization

The client side of the **callback** protocol interface MUST use the **callback** client proxy in the **callback client** state for the session instance.

3.6.4 Message Processing Events and Sequencing Rules

This interface includes the methods described in the following table.

Method	Description
secure	Contains status information about operations within a range which have been secured.
complete	Contains status information about operations within a range which have been finished.

3.6.4.1 secure

Normally, a **secure** callback message, as specified in section 3.5.4.1, MUST be sent by the indexing node when all operations received in a batch via **session::process** have been persisted to disk. However, if the **docapi suspended** state is **true**, a **secure** callback message MUST be sent by the indexing node without delay with an **error** and **error_code** set to 4, as specified in section 2.2.3.

3.6.4.2 complete

Normally, a **complete** callback message, as specified in section 3.6.4.2 MUST be sent by the indexing node when all operations received in a batch via **session::process** have been processed and the actions triggered by the operations are visible in the search service application. However, if the **indexing suspended** state is **true**, a **complete** callback message MUST be sent by the indexing node without delay with a **warning** and **warning_code** set to 2, as specified in section 2.2.15.

3.6.5 Timer Events

None.

3.6.6 Other Local Events

None.

4 Protocol Examples

4.1 Retrieve the Highest Session Identifier

This example describes how to use the **get_highest_session_id** method of the **session_factory** interface, as specified in section 3.1.4.2, so the indexing dispatcher can ask the indexing node in index column 0 for the highest session identifier.

First the protocol server creates a server object implementing the **session_factory** interface, and registers it in the name server. The protocol client can acquire a client proxy to this **session_factory** 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 can now call the **get_highest_session_id** method on the **session_factory** client proxy.

4.1.1 Sample Code

4.1.1.1 Protocol Server Initialization

```
SET server_object_instance TO INSTANCE OF session_factory SERVER OBJECT

SET server_object_host TO "myserver.mydomain.com"

SET server_object_port TO "1234"

SET server_object_interface_type TO "indexingengine::session_factory"

SET server_object_interface_version TO "5.7"

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

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_instance AND server_object_aor
```

4.1.1.2 Protocol Client Initialization

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

SET server_object_type TO "indexingengine::session_factory"

SET server_object_version TO "5.7"

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

4.1.1.3 Protocol Client Message

```
CALL session_factory_client_proxy.get_highest_session_id RETURNING highest_session_id
```

4.1.1.4 Server Response

```
RETURN highest_session_id
```

4.2 Send Operations and Receive Callbacks

This example describes how to create and set up a session, feed 10 item operations on this session, receive callback messages about the status of the items, and then close the session. In this example, the items are sent to the indexing node on index column 0.

Initialization: The **session factory** protocol server creates a server object implementing the **session_factory** interface, and registers it in the name server. The **session factory** protocol client acquires a client proxy to this **session_factory** 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.

Setting up the session: The **session factory** protocol client creates and activates a **callback** server object, and sends a client proxy to this server object to the **session factory** protocol server using the **session_factory::create_session** message.

The **session factory** protocol server receives the **create_session** message, creates, activates, and returns a **session** client proxy, stores the received session identifier in the **session id** state, stores the created **session** server object in the **session server** state, and stores the received **callback** client proxy in the **callback client** state.

The **session factory** protocol client stores the returned **session** client proxy in the **session client holder** state.

Using the session: The **session** protocol client retrieves a **session** client proxy from the **session client holder** state, updates the **operations callback holder** state with the operations being sent, and uses the **session::process** message to send the operations to the indexing node.

Sending callbacks: The **session** protocol server receives the **process** message, and processes the item operations given as parameter. When the operations are persisted to disk, it looks up the **callback** client proxy from the **callback client** state, and sends a **callback::secure** message back to the indexing dispatcher node. The same goes for the **callback::complete** message when the operations are processed and the effect made searchable in the search service application.

Receiving callbacks: The **callback** protocol server receives the **secure** and **complete** messages, and updates the **operations callback holder** state. When information about all operations sent using **session::process** have been received, the feeding process is complete.

Closing the session: The indexing dispatcher node closes the session using the **session_factory::close** message.

4.2.1 Sample Code

4.2.1.1 Session Factory Protocol Server Initialization

```
SET session_factory_server_object_instance TO INSTANCE OF session_factory SERVER OBJECT

SET session_factory_server_object_host TO "myserver.mydomain.com"

SET session_factory_server_object_port TO "1234"
```

```
SET session_factory_server_object_interface_type TO "indexingengine::session_factory"

SET session_factory_server_object_interface_version TO "5.7"

SET session_factory_server_object_name TO "esp/clusters/webcluster/indexing/indexer-
0/sessionfactory"

SET session_factory_server_object_aor TO session_factory_server_object_host,
session_factory_server_object_port, session_factory_server_object_interface_type,
session_factory_server_object_interface_version AND session_factory_server_object_name

CALL nameserver.bind WITH session_factory_server_object_instance AND
session_factory_server_object_aor
```

4.2.1.2 Session Factory Protocol Client Initialization

```
SET session_factory_server_object_name TO "esp/clusters/webcluster/indexing/indexer-
0/sessionfactory"

SET session_factory_server_object_type TO "indexingengine::session_factory"

SET session_factory_server_object_version TO "5.7"

CALL nameserver.resolve WITH session_factory_server_object_name,
session_factory_server_object_type AND session_factory_server_object_version RETURNING
session_factory_client_proxy
```

4.2.1.3 Session Factory Protocol Client Message

```
SET session_id TO "1"

SET collection TO "mycollection"

SET callback_server_object_instance TO INSTANCE OF callback SERVER OBJECT

CALL session_factory_client_proxy.create_session WITH session_id AND collection AND
callback_server_object_instance RETURNING session_client_proxy

ADD session_client_proxy TO session_client_holder_state
```

4.2.1.4 Session Factory Protocol Server Response

```
SET session_id_state TO session_id

SET callback_client_state TO callback_server_object_instance

SET session_server_object_instance TO INSTANCE OF session SERVER OBJECT

SET session_server_state TO session_server_object_instance

RETURN session_server_object_instance
```

4.2.1.5 Session Protocol Client Initialization

```
GET session_client_proxy FROM session_client_holder_state
```

4.2.1.6 Session Protocol Client Message

```
SET last_operation_in_sequence TO "9"

SET operations TO OPERATION_SET_OBJECT_WITH_10_OPERATIONS

ADD RANGE OF operations TO operations_callback_holder_state

CALL session_client_proxy.process WITH last_operation_in_sequence AND operations
```

4.2.1.7 Session Protocol Server Response

```
PROCESS ALL operations

SET callback_client_proxy TO callback_client_state

REPEAT
  IF operation RANGE IN operations IS PERSISTED TO DISK, THEN
    CALL callback_client_proxy.secure WITH OPERATION_STATUS_INFO FOR RANGE
  IF operation RANGE IN operations IS SEARCHABLE, THEN
    CALL callback_client_proxy.complete WITH OPERATION_STATUS_INFO FOR RANGE
UNTIL complete callback HAS BEEN SENT FOR ALL operations
```

4.2.1.8 Session Protocol Client Response

```
REPEAT
  UPDATE operations_callback_holder_state WITH RANGE FROM callback INFORMATION
UNTIL complete callback HAS BEEN RECEIVED FOR ALL operations IN
operations_callback_holder_state
```

4.2.1.9 Session Factory Protocol Client Close

```
CALL session_factory_client_proxy.close WITH session_id
```

4.2.1.10 Session Factory Protocol Server Close

```
GET session_server_object_instance FROM session_server_state FOR session_id

REMOVE session_server_object_instance FROM session_server_state

DEACTIVATE session_server_object_instance
```

4.2.2 Time Sequence Diagram

A time sequence diagram of the example in this section is found in the following figure.

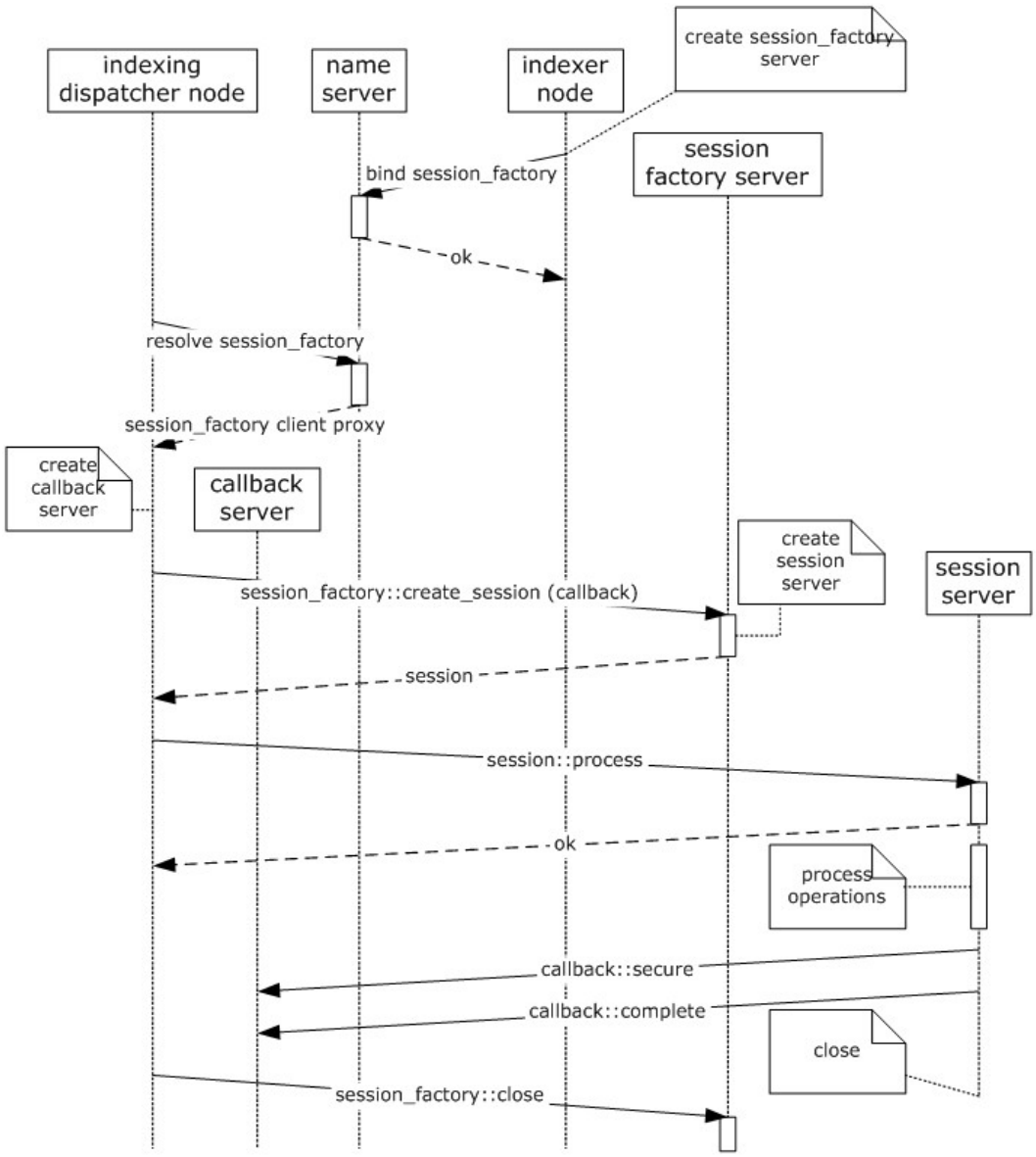

Figure 4: Sending operations and receiving callbacks

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.

6.1 FSIDL

```
module cht {
  module documentmessages {
    typedef sequence<octet> cheetah;
    typedef cheetah error;
    typedef cheetah warning;
    typedef cheetah operation_set;
    typedef cheetah operation_status_info;
    typedef cheetah operation_status_info_set
  };
};

module interfaces {
  module indexingengine {

    exception shutdown_exception {
    };

    exception batch_format_error {
      string what;
    };

    exception resource_error {
      string what;
    };

    exception invalid_input_exception {
      string what;
    };

    interface session_factory {
#pragma version session_factory 5.7
      session create_session(in long session_id, in string collection, in callback
the_callback)
        raises (shutdown_exception, invalid_input_exception);
      long get_highest_session_id();
      void close(in long session_id);
      void flush_session(in long session_id);
    };

    interface session {
#pragma version session 5.11
      boolean process(in long long last_operation_in_sequence,
                      in cht::documentmessages::operation_set operations)
        raises (batch_format_error, resource_error);
      long get_id();
      long long get_last_operation_id();
    };

    interface callback {
#pragma version callback 5.0
      void secure(in cht::documentmessages::operation_status_info status);
```

```
      void complete(in cht::documentmessages::operation_status_info status);
    };
  };
};
```

6.2 Cheetah Entities

```
enum action {
  resubmit,
  limited_resubmit,
  drop,
  terminate
};

enum operation_state {
  unknown,
  received,
  secured,
  completed,
  lost
};

root entity error {
  attribute int error_code;
  attribute action suggested_action;
  attribute string description;
  attribute string subsystem;
  attribute int session_id;
  attribute longint operation_id;
  collection string arguments;
};

entity processing_error : error {
  attribute string processor;
};

entity format_error : processing_error {
};

entity xml_error : format_error {
};

entity utf8_error : format_error {
};

entity server_unavailable : processing_error {
};

entity operation_dropped : processing_error {
};

entity operation_lost : error {
};

entity indexing_error : error {
};
```

```
entity invalid_content : indexing_error {
};

entity resource_error : indexing_error {
};

entity unknown_document : indexing_error {
};

root entity warning {
  attribute int warning_code;
  attribute string description;
  attribute string subsystem;
  attribute int session_id;
  attribute longint operation_id;
};

entity operation {
  attribute longint id;
  collection warning warnings;
};

root entity operation_set {
  attribute longint completed_op_id;
  collection operation operations;
};

root entity operation_status_info {
  attribute longint first_op_id;
  attribute longint last_op_id;
  attribute operation_state state;
  attribute string subsystem;
  collection error errors;
  collection warning warnings;
};

root entity operation_status_info_set {
  collection operation_status_info status;
};

root entity document_id {
  attribute string id;
  collection key_value_pair routing_attributes;
};

entity key_value_pair {
  attribute string key;
};

entity key_value_collection : key_value_pair {
  collection key_value_pair values;
};

entity string_attribute : key_value_pair {
  attribute string value;
};

entity integer_attribute : key_value_pair {
  attribute int value;
```

```
};

entity bytearray_attribute : key_value_pair {
  attribute bytearray value;
};

entity internal_partial_update_operation {
};

entity remove_nodes : internal_partial_update_operation {
  attribute string node_selection;
};

entity insert_xml : internal_partial_update_operation {
  attribute string_attribute key_value;
};

entity string_replace : internal_partial_update_operation {
  attribute string_attribute key_value;
};

root entity internal_partial_update : operation {
  attribute document_id doc_id;
  collection internal_partial_update_operation operations;
};

root entity document {
  attribute document_id doc_id;
  collection key_value_pair document_attributes;
};

entity no_operation : operation {
};

entity clear_collection : operation {
};

entity failed_operation : operation {
  attribute string subsystem;
  attribute operation_state state;
  attribute string operation_type;
  attribute document_id doc_id;
  attribute error err;
};

entity remove_operation : operation {
  attribute document_id doc_id;
};

entity update_operation : operation {
  attribute document doc;
};
```

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

E

F

G

I

S

T

V