Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.

- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL’s, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.

- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.

- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft’s delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.

- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.

- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

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

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision History</th>
<th>Revision Class</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/06/2009</td>
<td>0.1</td>
<td>Major</td>
<td>Initial Availability</td>
</tr>
<tr>
<td>02/19/2010</td>
<td>1.0</td>
<td>Major</td>
<td>Updated and revised the technical content</td>
</tr>
<tr>
<td>03/31/2010</td>
<td>1.01</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>04/30/2010</td>
<td>1.02</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>06/07/2010</td>
<td>1.03</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>06/29/2010</td>
<td>1.04</td>
<td>Editorial</td>
<td>Changed language and formatting in the technical content.</td>
</tr>
<tr>
<td>07/23/2010</td>
<td>1.04</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>09/27/2010</td>
<td>1.04</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>11/15/2010</td>
<td>1.04</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>12/17/2010</td>
<td>1.04</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>03/18/2011</td>
<td>1.04</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>06/10/2011</td>
<td>1.04</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>01/20/2012</td>
<td>1.5</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>04/11/2012</td>
<td>1.5</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>07/16/2012</td>
<td>1.5</td>
<td>No change</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
</tbody>
</table>
# Table of Contents

1 **Introduction** .............................................................................................................................. 6  
1.1 Glossary ........................................................................................................................................... 6  
1.2 References ...................................................................................................................................... 6  
1.2.1 Normative References .................................................................................................................. 7  
1.2.2 Informative References .................................................................................................................. 7  
1.3 Protocol Overview (Synopsis) ......................................................................................................... 7  
1.4 Relationship to Other Protocols ...................................................................................................... 8  
1.5 Prerequisites/Preconditions ............................................................................................................. 9  
1.6 Applicability Statement .................................................................................................................. 9  
1.7 Versioning and Capability Negotiation ............................................................................................ 9  
1.8 Vendor-Extensible Fields .................................................................................................................. 9  
1.9 Standards Assignments ................................................................................................................... 9  

2 **Messages** ...................................................................................................................................... 10  
2.1 Transport ......................................................................................................................................... 10  
2.2 Common Data Types ....................................................................................................................... 10  
2.2.1 `cht::documentmessages::action` ............................................................................................. 11  
2.2.2 `cht::documentmessages::operation_state` ................................................................................. 11  
2.2.3 `cht::documentmessages::entity_error` ..................................................................................... 12  
2.2.4 `cht::documentmessages::processing_error` ............................................................................ 13  
2.2.5 `cht::documentmessages::format_error` .................................................................................... 13  
2.2.6 `cht::documentmessages::xml_error` ......................................................................................... 13  
2.2.7 `cht::documentmessages::utf8_error` ....................................................................................... 14  
2.2.8 `cht::documentmessages::server_unavailable` ......................................................................... 14  
2.2.9 `cht::documentmessages::operation_dropped` .......................................................................... 14  
2.2.10 `cht::documentmessages::operation_lost` ............................................................................. 14  
2.2.11 `cht::documentmessages::indexing_error` ............................................................................ 14  
2.2.12 `cht::documentmessages::invalid_content` ........................................................................... 14  
2.2.13 `cht::documentmessages::resource_error` ............................................................................. 15  
2.2.14 `cht::documentmessages::unknown_document` ....................................................................... 15  
2.2.15 `cht::documentmessages::warning` ......................................................................................... 15  
2.2.16 `cht::documentmessages::operation` ....................................................................................... 16  
2.2.17 `cht::documentmessages::operation_set` ............................................................................... 16  
2.2.18 `cht::documentmessages::operation_status_info` .................................................................. 16  
2.2.19 `cht::documentmessages::operation_status_info_set` ................................................................. 17  
2.2.20 `cht::documentmessages::document_id` ................................................................................... 17  
2.2.21 `cht::documentmessages::key_value_pair` .............................................................................. 17  
2.2.22 `cht::documentmessages::key_value_collection` .................................................................... 17  
2.2.23 `cht::documentmessages::string_attribute` ............................................................................ 18  
2.2.24 `cht::documentmessages::integer_attribute` .......................................................................... 18  
2.2.25 `cht::documentmessages::bytearray_attribute` ....................................................................... 18  
2.2.26 `cht::documentmessages::internal_partial_update_operation` ............................................... 18  
2.2.27 `cht::documentmessages::remove_nodes` ............................................................................... 19  
2.2.28 `cht::documentmessages::insert_xml` ...................................................................................... 19  
2.2.29 `cht::documentmessages::string_replace` ............................................................................... 19  
2.2.30 `cht::documentmessages::internal_partial_update` ................................................................. 19  
2.2.31 `cht::documentmessages::document` ....................................................................................... 20  
2.2.32 `cht::documentmessages::no_operation` ............................................................................... 20  
2.2.33 `cht::documentmessages::clear_collection` ......................................................................... 20  
2.2.34 `cht::documentmessages::failed_operation` .......................................................................... 20
3 Protocol Details ............................................................................................................. 22
  3.1 indexingengine::session_factory Server Details ..................................................... 22
    3.1.1 Abstract Data Model ......................................................................................... 23
    3.1.2 Timers .............................................................................................................. 23
    3.1.3 Initialization .................................................................................................... 23
    3.1.4 Message Processing Events and Sequencing Rules ...................................... 23
      3.1.4.1 create_session ......................................................................................... 24
      3.1.4.2 get_highest_session_id ......................................................................... 24
      3.1.4.3 close ........................................................................................................ 25
      3.1.4.4 flush_session ......................................................................................... 25
    3.1.5 Timer Events .................................................................................................. 25
    3.1.6 Other Local Events ....................................................................................... 25
  3.2 indexingengine::session_factory Client Details .................................................... 25
    3.2.1 Abstract Data Model ..................................................................................... 25
    3.2.2 Timers ............................................................................................................ 26
    3.2.3 Initialization .................................................................................................. 26
    3.2.4 Message Processing Events and Sequencing Rules .................................. 26
      3.2.4.1 create_session ......................................................................................... 27
      3.2.4.2 get_highest_session_id ......................................................................... 27
      3.2.4.3 close ........................................................................................................ 27
      3.2.4.4 flush_session ......................................................................................... 27
    3.2.5 Timer Events .................................................................................................. 27
    3.2.6 Other Local Events ....................................................................................... 27
  3.3 indexingengine::session Server Details .................................................................. 27
    3.3.1 Abstract Data Model ..................................................................................... 27
    3.3.2 Timers ............................................................................................................ 28
    3.3.3 Initialization .................................................................................................. 28
    3.3.4 Message Processing Events and Sequencing Rules .................................. 28
      3.3.4.1 process .................................................................................................. 28
      3.3.4.2 get_id .................................................................................................... 29
      3.3.4.3 get_last_operation_id ........................................................ ...................... 29
    3.3.5 Timer Events .................................................................................................. 29
    3.3.6 Other Local Events ....................................................................................... 29
  3.4 indexingengine::session Client Details .................................................................. 30
    3.4.1 Abstract Data Model ..................................................................................... 30
    3.4.2 Timers ............................................................................................................ 30
    3.4.3 Initialization .................................................................................................. 30
    3.4.4 Message Processing Events and Sequencing Rules .................................. 30
      3.4.4.1 process .................................................................................................. 30
      3.4.4.2 get_id .................................................................................................... 30
      3.4.4.3 get_last_operation_id ........................................................ ...................... 30
    3.4.5 Timer Events .................................................................................................. 30
    3.4.6 Other Local Events ....................................................................................... 30
  3.5 indexingengine::callback Server Details ............................................................... 31
    3.5.1 Abstract Data Model ..................................................................................... 31
    3.5.2 Timers ............................................................................................................ 31
    3.5.3 Initialization .................................................................................................. 31
    3.5.4 Message Processing Events and Sequencing Rules .................................. 31
      3.5.4.1 send ....................................................................................................... 31
      3.5.4.2 complete ............................................................................................... 31

[MS-FSID] – v20120630
Indexing Distribution Protocol Specification

Copyright © 2012 Microsoft Corporation.

Release: July 16, 2012
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".


1.2.2 Informative References


[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.
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:

![Diagram showing the relationship between Indexing Distribution, Middleware, HTTP or HTTPS, TCP, and IP, with This Protocol and Industry Standard labels.]

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.

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

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

Valid values MUST be as listed in the following table.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>resubmit</td>
<td>MUST NOT be used.</td>
</tr>
<tr>
<td>1</td>
<td>limited_resubmit</td>
<td>MUST NOT be used.</td>
</tr>
<tr>
<td>2</td>
<td>drop</td>
<td>Drop the item from the index.</td>
</tr>
<tr>
<td>3</td>
<td>terminate</td>
<td>MUST NOT be used.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unknown</td>
<td>Unknown state.</td>
</tr>
<tr>
<td>1</td>
<td>received</td>
<td>Operation received.</td>
</tr>
<tr>
<td>2</td>
<td>secured</td>
<td>Operation persisted to disk, but change not yet reflected in searchable index.</td>
</tr>
<tr>
<td>3</td>
<td>completed</td>
<td>Operation persisted to disk, and change reflected in searchable index.</td>
</tr>
<tr>
<td>4</td>
<td>lost</td>
<td>MUST NOT be used.</td>
</tr>
</tbody>
</table>

2.2.3 cht::documentmessages::entity error

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

```c
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.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Missing attribute (1).</td>
</tr>
<tr>
<td>2</td>
<td>Generic error.</td>
</tr>
<tr>
<td>3</td>
<td>Unknown document.</td>
</tr>
<tr>
<td>4</td>
<td>Indexer suspended. Set by the session::process server object when a batch of operations is received while the docapi suspended state is true.</td>
</tr>
<tr>
<td>5</td>
<td>FAST Index Markup Language (FIXML) write error. Failed to persist the batch of item operations.</td>
</tr>
<tr>
<td>6</td>
<td>Unknown content collection.</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>7</td>
<td>Error during partial update.</td>
</tr>
</tbody>
</table>

**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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All index partitions are full.</td>
</tr>
<tr>
<td>2</td>
<td>Indexing suspended. The indexing suspended state is true.</td>
</tr>
</tbody>
</table>

**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.

```cpp
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.

```cpp
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.

```cpp
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".

effects: 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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```cpp
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.

```cpp
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.

![Diagram showing sessions and callback messages](image)

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_session</td>
<td>Creates a new session server object identified by a session identifier, and returns a session client proxy.</td>
</tr>
<tr>
<td>get_highest_session_id</td>
<td>Returns the highest session identifier number used so far when creating new sessions using session_factory::create_session.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>close</td>
<td>Closes a specific session.</td>
</tr>
<tr>
<td>flush_session</td>
<td>Resets the state of a specific session.</td>
</tr>
</tbody>
</table>

### 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.

```c++
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.

```c++
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-FSWM].

### 3.1.4.3 close

The `close` method closes down a specific session.

```c
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-FSWM].

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.

```c
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-FSWM].

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_session</td>
<td>Creates a new session server object identified by a session identifier, and returns a session client proxy.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>get_highest_session_id</td>
<td>Returns the highest session identifier number used so far when creating new sessions using session_factory::create_session.</td>
</tr>
<tr>
<td>close</td>
<td>Closes a specific session.</td>
</tr>
<tr>
<td>flush_session</td>
<td>Resets the state of a specific session.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>Processes a sequence of operations.</td>
</tr>
<tr>
<td>get_id</td>
<td>Returns the identifier of the session.</td>
</tr>
<tr>
<td>get_last_operation_id</td>
<td>Returns the identifier of the last in-sequence operation submitted to the session.</td>
</tr>
</tbody>
</table>

#### 3.3.4.1 process

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

```cpp
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-FSID].

The minimum free available disk space on the disks holding the indexes is less than the value `diskspaceMBWarning`, as specified in [MS-FSID].

The fault tolerance feature, as specified in [MS-FSID], is enabled, and the free available disk space in the data directory is less than the value `diskspaceMBWarning`, as specified in [MS-FSID].

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.

```cpp
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.

```cpp
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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>Processes a sequence of operations.</td>
</tr>
<tr>
<td>get_id</td>
<td>Returns the identifier of the session.</td>
</tr>
<tr>
<td>get_last_operation_id</td>
<td>Returns the identifier of the last in-sequence operation submitted to the session.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secure</td>
<td>Contains status information about operations within a range which have been secured.</td>
</tr>
<tr>
<td>complete</td>
<td>Contains status information about operations within a range which have been finished.</td>
</tr>
</tbody>
</table>

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.

```cpp
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.

```cpp
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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secure</td>
<td>Contains status information about operations within a range which have been secured.</td>
</tr>
<tr>
<td>complete</td>
<td>Contains status information about operations within a range which have been finished.</td>
</tr>
</tbody>
</table>

#### 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

```fortran
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

```fortran
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

```fortran
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 to this session factory protocol client. It 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);
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

A

Abstract data model client (section 3.2.1 25, section 3.4.1 30, section 3.6.1 32)
server (section 3.1.1 23, section 3.3.1 27, section 3.5.1 31)

Applicability 9

C

Capability negotiation 9
Change tracking 45
cht::documentmessages::action data type 11
cht::documentmessages::bytearray_attribute data type 18
cht::documentmessages::clear_collection data type 20
cht::documentmessages::document data type 20
cht::documentmessages::document_id data type 17
cht::documentmessages::entity_error data type 12
cht::documentmessages::failed_operation data type 20
cht::documentmessages::format_error data type 13
cht::documentmessages::indexing_error data type 14
cht::documentmessages::insert_xml data type 19
cht::documentmessages::integer_attribute data type 18
cht::documentmessages::internal_partial_update data type 19
cht::documentmessages::internal_partial_update_operation data type 18
cht::documentmessages::invalid_content data type 14
cht::documentmessages::key_value_collection data type 17
cht::documentmessages::key_value_pair data type 17
cht::documentmessages::no_operation data type 20
cht::documentmessages::operation data type 16
cht::documentmessages::operation_dropped data type 14
cht::documentmessages::operation_lost data type 14
cht::documentmessages::operation_set data type 16
cht::documentmessages::operation_state data type 11
cht::documentmessages::operation_status_info data type 16
cht::documentmessages::operation_status_info_set data type 17
cht::documentmessages::processing_error data type 13
cht::documentmessages::remove_nodes data type 19
cht::documentmessages::remove_operation data type 21
cht::documentmessages::resource_error data type 15
cht::documentmessages::server_unavailable data type 14
cht::documentmessages::string_attribute data type 18
cht::documentmessages::string_replace data type 19
cht::documentmessages::unknown_document data type 15
cht::documentmessages::update_operationdata 21
cht::documentmessages::utf8_error data type 14
cht::documentmessages::warning data type 15
cht::documentmessages::xml_error data type 13

Client

abstract data model (section 3.2.1 25, section 3.4.1 30, section 3.6.1 32)
close method 27
complete method 33
create_session method 27
flush_session method 27
get_highest_session_id method 27
get_id method 30
get_last_operation_id method 30
indexingengine::callback interface 32
indexingengine::session_interface 30
indexingengine::session_factory interface 25
initialization (section 3.2.3 26, section 3.4.3 30, section 3.6.3 32)
local events (section 3.2.6 27, section 3.4.6 30, section 3.6.6 33)
message processing (section 3.2.4 26, section 3.4.4 30, section 3.6.4 32)
overview (section 3.2 22, section 3.2 25, section 3.4 30, section 3.6 32)
process method 30
secure method 32
sequencing rules (section 3.2.4 26, section 3.4.4 30, section 3.6.4 32)
timer events (section 3.2.5 27, section 3.4.5 30, section 3.6.5 33)
timers (section 3.2.2 26, section 3.4.2 30, section 3.6.2 32)
close method (section 3.1.4.3 25, section 3.2.4.3 27)

Common data types 10
complete method (section 3.5.4.2 31, section 3.6.4.2 33)
create_session method (section 3.1.4.1 24, section 3.2.4.1 27)

D

Data model - abstract
client (section 3.2.1 25, section 3.4.1 30, section 3.6.1 32)
server (section 3.1.1 23, section 3.3.1 27, section 3.5.1 31)

Data types
cht::documentmessages::action 11
cht::documentmessages::bytearray_attribute 18
cht::documentmessages::clear_collection 20
cht::documentmessages::document 20
cht::documentmessages::document_id 17
cht::documentmessages::entity_error 12
cht::documentmessages::failed_operation 20
cht::documentmessages::format_error 13
cht::documentmessages::indexing_error 14
cht::documentmessages::insert_xml 19
cht::documentmessages::integer_attribute 18
cht::documentmessages::internal_partial_update 18
cht::documentmessages::key_value_pair 19
cht::documentmessages::invalid_content 17
cht::documentmessages::key_value_pair 17
cht::documentmessages::no_operation 20
cht::documentmessages::operation 16
cht::documentmessages::operation_dropped 14
cht::documentmessages::operation_lost 14
cht::documentmessages::operation_set 16
cht::documentmessages::operation_stat 11
c::documentmessages::operation_status_info
cht::documentmessages::operation_status_info
cht::documentmessages::operation_status_info
cht::documentmessages::processing_error 13
cht::documentmessages::remove_nodes 19
cht::documentmessages::remove_operation 21
cht::documentmessages::resource_error 15
cht::documentmessages::server_unavailable 14
cht::documentmessages::string_attribute 18
cht::documentmessages::string_replace 19
cht::documentmessages::unknown_document 15
cht::documentmessages::update_operation 21
cht::documentmessages::utf8_error 14
cht::documentmessages::warning 15
cht::documentmessages::xml_error 13

Common - overview 10

E

Events
local - client (section 3.2.6 27, section 3.4.6 30, section 3.6.3 33)
local - server (section 3.1.6 25, section 3.3.6 29, section 3.5.6 32)
timer - client (section 3.2.5 27, section 3.4.5 30, section 3.6.5 33)
timer - server (section 3.1.5 25, section 3.3.5 29, section 3.5.5 32)

Examples
protocol client initialization 34
protocol client message 34

protocol server initialization 34
retrieve the highest session identifier 34
send operations and receive callbacks 35
server response 35
session factory protocol client
close 37
message 36
session factory protocol server
close 37
initialization (section 4.2.1.1 35, section 4.2.1.2 36)
response 36
session protocol client
initialization 37
message 37
response 37
session protocol server
response 37
time sequence diagram 37

F

Fields - vendor-extensible 9
flush_session method (section 3.1.4.4 25, section 3.2.4.4 27)
FSIDL 40
Full FSIDL 40

G

get_highest_session_id method (section 3.1.4.2 24, section 3.2.4.2 27)
get_id method (section 3.3.4.2 29, section 3.4.4.2 30)
get_last_operation_id method (section 3.3.4.3 29, section 3.4.4.3 30)

Glossary 6

I

Implementer - security considerations 39
Index of security parameters 39
indexingengine::callback interface (section 3.5 31, section 3.6 32)
indexingengine::session interface (section 3.3 27, section 3.4 30)
indexingengine::session_factory interface (section 3.1 22, section 3.2 25)

Informative references 7

Initialization
client (section 3.2.3 26, section 3.4.3 30, section 3.6.3 32)
session (section 3.1.3 23, section 3.3.3 28, section 3.5.3 31)

Interfaces - client
indexingengine::callback 32
indexingengine::session 30
indexingengine::session_factory 25

Interfaces - server
indexingengine::callback 31
indexingengine::session 27
indexingengine::session_factory 22
Introduction 6

L

Local events
client (section 3.2.6 27, section 3.4.6 30, section 3.6.6 33)
server (section 3.1.6 25, section 3.3.6 29, section 3.5.6 32)

M

Message processing
client (section 3.2.4 26, section 3.4.4 30, section 3.6.4 32)
server (section 3.1.4 23, section 3.3.4 28, section 3.5.4 31)

Messages
cht::documentmessages::action data type 11
cht::documentmessages::bytearray_attribute data type 18
cht::documentmessages::clear_collection data type 20
cht::documentmessages::document data type 20
cht::documentmessages::document_id data type 17
cht::documentmessages::entity_error data type 12
cht::documentmessages::failed_operation data type 20
cht::documentmessages::format_error data type 13
cht::documentmessages::indexing_error data type 19
cht::documentmessages::integer_attribute data type 14
cht::documentmessages::insert_xml data type 18
cht::documentmessages::internal_partial_update data type 19
cht::documentmessages::internal_partial_update_operation data type 18
cht::documentmessages::invalid_content data type 14
cht::documentmessages::key_value_collection data type 17
cht::documentmessages::key_value_pair data type 17
cht::documentmessages::no_operation data type 20
cht::documentmessages::operation data type 16
cht::documentmessages::operation_dropped data type 14
cht::documentmessages::operation_lost data type 14
cht::documentmessages::operation_set data type 16
cht::documentmessages::operation_state data type 11
cht::documentmessages::operation_status_info data type 16
cht::documentmessages::operation_status_info_set data type 17
cht::documentmessages::processing_error data type 13
cht::documentmessages::remove_nodes data type 19
cht::documentmessages::remove_operation data type 21
cht::documentmessages::resource_error data type 15
cht::documentmessages::server_unavailable data type 14
cht::documentmessages::string_attribute data type 18
cht::documentmessages::string_replace data type 19
cht::documentmessages::unknown_document data type 15
cht::documentmessages::update_operation data type 21
cht::documentmessages::utf8_error data type 14
cht::documentmessages::warning data type 15
cht::documentmessages::xml_error data type 13
common data types 10
transport 10

Methods
close (section 3.1.4.3 25, section 3.2.4.3 27)
create_session (section 3.1.4.1 24, section 3.2.4.1 27)
flush_session (section 3.1.4.4 25, section 3.2.4.4 27)
get_highest_session_id (section 3.1.4.2 24, section 3.2.4.2 27)
get_id (section 3.3.4.2 29, section 3.4.4.2 30)
get_last_operation_id (section 3.3.4.3 29, section 3.4.4.3 30)
process (section 3.3.4.1 28, section 3.4.4.1 30)
secure (section 3.5.4.1 31, section 3.6.4.1 32)

N

Normative references 7

O

Overview (synopsis) 7

P

Parameters - security index 39
Preconditions 9
Prerequisites 9
process method (section 3.3.4.1 28, section 3.4.4.1 30)
Product behavior 44
Protocol client initialization example 34
Protocol client message example 34
Protocol server initialization example 34

R
References  6
informative  7
normative  7
Relationship to other protocols  8
Retrieve the highest session identifier example  34

S
secure method (section 3.5.4.1 31, section 3.6.4.1 32)
Security
implementer considerations  39
parameter index  39
Send operations and receive callbacks example  35
Sequencing rules
client (section 3.2.4 26, section 3.4.4 30, section 3.6.4 32)
server (section 3.1.4 23, section 3.3.4 28, section 3.5.4.1 31)
Server
abstract data model (section 3.1.1 23, section 3.3.1 27, section 3.5.1 31)
close method  25
complete method  31
create_session method  24
flush_session method  25
get_highest_session_id method  24
get_id method  29
get_last_operation_id method  29
indexingengine::callback interface  31
indexingengine::session interface  27
indexingengine::session_factory interface  22
initialization (section 3.1.3 23, section 3.3.3 28, section 3.5.3 31)
local events (section 3.1.6 25, section 3.3.6 29, section 3.5.6 32)
message processing (section 3.1.4 23, section 3.3.4 28, section 3.5.4 31)
overview (section 3.2 22, section 3.1 22, section 3.2 27, section 3.5 31)
process method  28
secure method  31
sequencing rules (section 3.1.4 23, section 3.3.4 28, section 3.5.4 31)
timer events (section 3.1.5 25, section 3.3.5 29, section 3.5.5 32)
timers (section 3.1.2 23, section 3.3.2 28, section 3.5.2 31)
Server response example  35
session
protocol_client_initialization example  37
protocol_client_message example  37
protocol_client_response example  37
protocol_server_response example  37
session factory
protocol_client_close example  37
protocol_client_initialization example  36
protocol_client_message example  36
protocol_server_close example  37
protocol_server_initialization example  35
protocol_server_response example  36
Standards assignments  9

T
Time sequence diagram  37
Timer events
client (section 3.2.5 27, section 3.4.5 30, section 3.6.5 33)
server (section 3.1.5 25, section 3.3.5 29, section 3.5.5 32)
	Timers
client (section 3.2.2 26, section 3.4.2 30, section 3.6.2 32)
server (section 3.1.2 23, section 3.3.2 28, section 3.5.2 31)
Tracking changes  45
Transport  10
V
Vendor-extensible fields  9
Versioning  9