[MS-FSCF]:
Content Feeding Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft’s delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft 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 ........................................................................................................................................... 7
  1.1 Glossary ............................................................................................................................................... 7
  1.2 References .......................................................................................................................................... 8
    1.2.1 Normative References .................................................................................................................. 8
    1.2.2 Informative References ............................................................................................................... 8
  1.3 Overview ........................................................................................................................................... 8
  1.4 Relationship to Other Protocols ...................................................................................................... 9
  1.5 Prerequisites/Preconditions ............................................................................................................. 10
  1.6 Applicability Statement ................................................................................................................... 10
  1.7 Versioning and Capability Negotiation ........................................................................................... 10
  1.8 Vendor-Extensible Fields ................................................................................................................ 10
  1.9 Standards Assignments .................................................................................................................. 10

2 Messages .................................................................................................................................................. 11
  2.1 Transport .......................................................................................................................................... 11
  2.2 Common Data Types ....................................................................................................................... 11
    2.2.1 cht::core::guarantee_set ............................................................................................................ 12
    2.2.2 cht::core::guarantee .................................................................................................................. 13
    2.2.3 cht::core::feeding_priority ...................................................................................................... 13
    2.2.4 cht::documentmessages::action ............................................................................................... 13
    2.2.5 cht::documentmessages::operation_state ................................................................................ 13
    2.2.6 cht::documentmessages::error .................................................................................................. 14
    2.2.7 cht::documentmessages::processing_error ............................................................................. 14
    2.2.8 cht::documentmessages::format_error .................................................................................... 15
    2.2.9 cht::documentmessages::xml_error .......................................................................................... 15
    2.2.10 cht::documentmessages::utf8_error ....................................................................................... 15
    2.2.11 cht::documentmessages::server_unavailable .......................................................................... 15
    2.2.12 cht::documentmessages::operation_dropped ........................................................................... 16
    2.2.13 cht::documentmessages::operation_lost ............................................................................... 16
    2.2.14 cht::documentmessages::indexing_error ................................................................................. 16
    2.2.15 cht::documentmessages::invalid_content ............................................................................... 16
    2.2.16 cht::documentmessages::resource_error .................................................................................. 17
    2.2.17 cht::documentmessages::unknown_document .......................................................................... 17
    2.2.18 cht::documentmessages::warning ............................................................................................ 17
    2.2.19 cht::documentmessages::operation .......................................................................................... 18
    2.2.20 cht::documentmessages::operation_set .................................................................................. 18
    2.2.21 cht::documentmessages::operation_status_info ...................................................................... 18
    2.2.22 cht::documentmessages::operation_status_info_set ............................................................... 19
    2.2.23 cht::documentmessages::key_value_pair ............................................................................... 19
    2.2.24 cht::documentmessages::key_value_collection ....................................................................... 19
    2.2.25 cht::documentmessages::bytearray_attribute .......................................................................... 20
    2.2.26 cht::documentmessages::float_attribute ................................................................................ 20
    2.2.27 cht::documentmessages::double_attribute ............................................................................. 20
    2.2.28 cht::documentmessages::integer_attribute ............................................................................. 20
    2.2.29 cht::documentmessages::long_attribute ................................................................................ 20
    2.2.30 cht::documentmessages::string_attribute .............................................................................. 20
    2.2.31 cht::documentmessages::bytearray ........................................................................................ 20
    2.2.32 cht::documentmessages::string ........................................................................................... 20
    2.2.33 cht::documentmessages::bytearray_collection ...................................................................... 20
    2.2.34 cht::documentmessages::string_collection ........................................................................... 22
    2.2.35 cht::documentmessages::float_collection ............................................................................. 22
    2.2.36 cht::documentmessages::double_collection .......................................................................... 22
    2.2.37 cht::documentmessages::integer_collection .......................................................................... 22
    2.2.38 cht::documentmessages::long_collection .............................................................................. 22
    2.2.39 cht::documentmessages::bool_collection .............................................................................. 22
    2.2.40 cht::documentmessages::bool ............................................................................................... 22

[MS-FSCF] — v20120630
Content Feeding Protocol Specification

Copyright © 2012 Microsoft Corporation.

Release: July 16, 2012
3 Protocol Details

3.1 Common Server Details

3.1.1 Abstract Data Model ............................................. 32
3.1.2 Timers ...................................................................... 33
3.1.3 Initialization .......................................................... 33
3.1.4 Message Processing Events and Sequencing Rules .................. 33
3.1.5 Timer Events .......................................................... 33
3.1.6 Other Local Events ..................................................... 34

3.2 processing::session_factory Server Details

3.2.1 Abstract Data Model .................................................... 34
3.2.2 Timers ...................................................................... 34
3.2.3 Initialization .......................................................... 34
3.2.4 Message Processing Events and Sequencing Rules .................. 34
3.2.4.1 processing::session_factory::is_master ..................... 35
3.2.4.2 processing::session_factory::create ....................... 35
3.2.4.3 processing::session_factory::recreate .................... 36
3.2.4.4 processing::session_factory::close ......................... 37
3.2.5 Timer Events .......................................................... 38
3.2.6 Other Local Events ..................................................... 38

3.3 processing::session_factory Client Details

3.3.1 Abstract Data Model .................................................... 38
3.3.2 Timers ...................................................................... 38
3.3.3 Initialization .......................................................... 38
3.3.4 Message Processing Events and Sequencing Rules .................. 39
3.3.4.1 processing::session_factory::is_master ..................... 39
3.3.4.2 processing::session_factory::create ....................... 39
3.3.4.3 processing::session_factory::recreate .................... 39
3.3.4.4 processing::session_factory::close ......................... 39
3.3.5 Timer Events .......................................................... 39
3.3.6 Other Local Events ..................................................... 39

3.4 processing::session Server Details

3.4.1 Abstract Data Model .................................................... 39
3.4.2 Timers ...................................................................... 40
3.4.3 Initialization .......................................................... 40
3.4.4 Message Processing Events and Sequencing Rules .................. 40
3.4.4.1 processing::session::get_operation_timeout ............... 40
1 Introduction

This document specifies the Content Feeding Protocol. This protocol enables a protocol client to submit a set of documents to a protocol server for processing and indexing. This protocol also enables a protocol client to remove or update a set of documents from an index, and to remove a collection of documents from an index. A typical scenario for using this protocol is an application that traverses a file system and submits files to the protocol server for processing and indexing.

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
- certificate
- Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS)
- security identifier (SID)
- UTF-16
- UTF-8

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

- abstract object reference (AOR)
- base port
- callback message
- Cheetah
- Cheetah checksum
- Cheetah entity
- claim type
- claim value
- client proxy
- content client
- content collection
- content distributor
- crawled property
- document identifier
- FAST Search Interface Definition Language (FSIDL)
- host name
- indexing dispatcher
- indexing node
- managed property
- name server
- search index
- security principal identifier
- user store identifier

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".


1.2.2 Informative References


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

1.3 Overview

This protocol enables a content client to send item operations to a content distributor. The content distributor sends the item operations to an item processor for processing. The item processor sends the item operations to the indexing dispatcher for indexing. The content client receives callback messages from the content distributor when item operations have been processed, stored to disk, and when item operations have been indexed. This protocol also allows the content client to remove all items in a content collection.

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

![Protocol overview diagram](image)

**Figure 1: Protocol overview**

A typical scenario for using this protocol is a content client that traverses a file system. Files in the file system that the content client has not seen are submitted to the protocol server as an item add operation. Previously updated files are submitted to protocol server as an item update operation. For files that are no longer available, the content client submits an item remove operation to the protocol server.

The content distributor sends item operations to item processor for processing. The item processor sends the item operation to the indexing dispatcher for indexing. The content client receives callback messages from the content distributor when items have been processed and indexed. The content client uses callback messages to log progress. The content client also notifies the content client user of possible errors that occurred during processing and indexing of item operations.

This protocol consists of three interfaces, as described in `processing::session_factory Server Details` (section 3.2), `processing::session Server Details` (section 3.4), and `processing::control Server Details` (section 3.6). For these three interfaces, the content client acts as the protocol client and the content distributor acts as the protocol server.

### 1.4 Relationship to Other Protocols

This protocol relies on the Cheetah Data Format to serialize data, as described in [MS-FSCHT], and on the Middleware Protocol to transport data, as described in [MS-FSMW]. The following diagram shows the relationship between this protocol and other protocols.
1.5 Prerequisites/Preconditions

The content client is expected to know the location and connection information of the content distributor.

1.6 Applicability Statement

This protocol is designed for submitting items to the protocol server for processing and indexing. Information about status for processing and indexing of submitted items is sent back to protocol client using callback messages. This protocol also removes a content collection of items from an index.

This protocol is part of a feeding chain between an item feeding protocol client and an indexer. Using callback messages, the protocol server can send information about item status to the protocol client. This protocol is designed for the feeding chain segment between a protocol client and a protocol server.

1.7 Versioning and Capability Negotiation

Regarding capability negotiation:

- The Middleware Protocol, as described in [MS-FSMW], requires that the protocol server and the protocol client agree on the server interface version for all method invocations.

- The Cheetah Data Format, as described in [MS-FSCHT], requires that the protocol server and the protocol client agree on the Cheetah type identifiers and Cheetah checksum for all the Cheetah entities that are used by this protocol.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.
2 Messages

2.1 Transport

Messages MUST be transported by using the Middleware Protocol Specification, as specified in [MS-FSMW]. Data serialization MUST be performed by using the Cheetah Data Format Specification, as specified in [MS-FSCHT]. The protocol server uses two ports for transport, as specified in Initialization (section 3.2.3). The protocol server uses HTTP on base port + 390, as specified in [MS-FSMW] section 2. The protocol server MUST use HTTPS and deploy a certificate (1) on base port + 391, as specified in [MS-FSMW] section 2.

2.2 Common Data Types

The messages of this protocol are specified by using FAST Search Interface Definition Language (FSIDL). The allowed FSIDL data types are specified in [MS-FSMW].

FSIDL data types are encoded as specified in [MS-FSMW] section 2. Cheetah entities are encoded as specified in [MS-FSCHT] [MS-FSCHT] section 2. The Cheetah checksum and Cheetah type identifier for the Cheetah entities MUST be integers as specified in the following table.

<table>
<thead>
<tr>
<th>Cheetah entity</th>
<th>Cheetah type identifier</th>
<th>Cheetah checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>cht::core::guarantee</td>
<td>3</td>
<td>-1479218033</td>
</tr>
<tr>
<td>cht::core::feeding_priority</td>
<td>7</td>
<td>-1479218033</td>
</tr>
<tr>
<td>cht::core::guarantee_set</td>
<td>9</td>
<td>-1479218033</td>
</tr>
<tr>
<td>cht::documentmessages::key_value_pair</td>
<td>0</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::key_value_collection</td>
<td>1</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::bool_attribute</td>
<td>2</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::bool_collection</td>
<td>3</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::bytearray_attribute</td>
<td>4</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::bytearray_collection</td>
<td>5</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::warning</td>
<td>6</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::operation</td>
<td>7</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::document_id</td>
<td>11</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::document</td>
<td>12</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::error</td>
<td>15</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::float_attribute</td>
<td>19</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::float_collection</td>
<td>20</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::processing_error</td>
<td>21</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::format_error</td>
<td>22</td>
<td>-211918678</td>
</tr>
<tr>
<td>Cheetah entity</td>
<td>Cheetah type identifier</td>
<td>Cheetah checksum</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>cht::documentmessages::indexing_error</td>
<td>23</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::string_attribute</td>
<td>26</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::integer_attribute</td>
<td>28</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::integer_collection</td>
<td>30</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::invalid_content</td>
<td>33</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::long_attribute</td>
<td>34</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::long_collection</td>
<td>35</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::operation_dropped</td>
<td>36</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::operation_lost</td>
<td>37</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::operation_set</td>
<td>38</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::subsystem_id_set</td>
<td>39</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::operation_status_info</td>
<td>40</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::operation_status_info_set</td>
<td>41</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::partial_update_operation</td>
<td>42</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::remove_operation</td>
<td>45</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::resource_error</td>
<td>46</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::server_unavailable</td>
<td>47</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::string_collection</td>
<td>49</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::unknown_document</td>
<td>51</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::update_operation</td>
<td>52</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::urlschange_operation</td>
<td>53</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::utf8_error</td>
<td>54</td>
<td>-211918678</td>
</tr>
<tr>
<td>cht::documentmessages::xml_error</td>
<td>55</td>
<td>-211918678</td>
</tr>
</tbody>
</table>

The following sections specify these Cheetah entities.

### 2.2.1 cht::core::guarantee_set

The **guarantee_set** Cheetah entity contains a collection of **guarantee** Cheetah entities, as specified in section **2.2.2**. Cheetah entity specification for **guarantee_set**:

```cpp
entity guarantee_set {
    collection guarantee guarantees;
};
```
guarantees: A collection of guarantee Cheetah entities.

2.2.2 cht::core::guarantee

The guarantee Cheetah entity is a parent class for the feeding_priority Cheetah entity, as specified in section 2.2.3. Cheetah entity specification for guarantee:

```cpp
entity guarantee {
};
```

2.2.3 cht::core::feeding_priority

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

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

priority: An integer that MUST be 0.

2.2.4 cht::documentmessages::action

The action Cheetah enumeration contains constants for actions used as part of an error message. Cheetah enumeration specification for action:

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

The action Cheetah enumeration contains the following constants:

resubmit: A constant specifying that the protocol client MUST resubmit the item operation.

limited_resubmit: A constant specifying that the protocol client MUST resubmit the item operation for a limited number of times.

drop: A constant specifying that the protocol client MUST NOT resubmit the item operation.

terminate: A constant that the protocol client MUST NOT use.

2.2.5 cht::documentmessages::operation_state

The operation_state Cheetah enumeration contains constants for the possible states of an item operation. Cheetah enumeration specification for operation_state:

```cpp
enum operation_state {
    unknown,
    received,
    secured,
};
```
unknown: A constant specifying that the item operation is in an unknown state.

received: A constant specifying that the protocol server has received the item operation.

secured: A constant specifying that the item operation has been saved to disk.

completed: A constant specifying that the item operation has finished running.

lost: A constant specifying that the item operation was lost during processing or indexing.

2.2.6 cht::documentmessages::error

The error Cheetah entity contains error information for a specific item operation with a specified item operation identifier. Cheetah entity specification for error:

definition:

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 that contains the error code.

suggested_action: An action Cheetah enumeration value, as specified in section 2.2.4, containing the suggested action that the protocol client can perform to correct the item operation error.

description: A string that contains a description of the error.

subsystem: A string that describes where the error occurred. This string MUST have a value of either "indexing" or "processing". If the error was produced by either the content distributor or the item processor, the string value will be "processing". If the error was produced by either the indexing dispatcher or an indexing node, the string value will be "indexing".

session_id: An integer that uniquely identifies the session.

operation_id: An integer that uniquely identifies the item operation.

arguments: Unused. The value MUST be an empty Cheetah collection.

2.2.7 cht::documentmessages::processing_error

The processing_error Cheetah entity specifies errors that occur during the processing of an item operation.

The processing_error Cheetah entity is a subclass of the error Cheetah entity that is specified in section 2.2.6. The processing_error Cheetah entity is a common superclass for the following Cheetah entities:
• **format_error**, which is specified in section 2.2.8.

• **server_unavailable**, which is specified in section 2.2.11.

• **operation_dropped**, which is specified in section 2.2.12.

Cheetah entity specification for **processing_error**:

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

**processor**: A string that specifies the name of the item processor stage where the error occurred.

### 2.2.8 cht::documentmessages::format_error

The **format_error** Cheetah entity is used when an item operation has an invalid format.

The **format_error** Cheetah entity is a subclass of the **processing_error**, Cheetah entity that is specified in section 2.2.7. The **format_error** Cheetah entity is a common superclass for the **xml_error** Cheetah entity that is specified in section 2.2.9 and the **utf8_error** Cheetah entity that is specified in section 2.2.10. Cheetah entity specification for **format_error**:

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

### 2.2.9 cht::documentmessages::xml_error

The **xml_error** Cheetah entity is used when an item operation contains XML that is not valid.

The **xml_error** Cheetah entity is a subclass of the **format_error**, Cheetah entity that is specified in section 2.2.8. Cheetah entity specification for **xml_error**:

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

### 2.2.10 cht::documentmessages::utf8_error

The **utf8_error** Cheetah entity is used when an item operation contains invalid UTF-8 encoding.

The **utf8_error** Cheetah entity is a subclass of the **format_error** Cheetah entity that is specified in section 2.2.8. Cheetah entity specification for **utf8_error**:

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

### 2.2.11 cht::documentmessages::server_unavailable

The **server_unavailable** Cheetah entity is used when a protocol client is unable to connect to a protocol server during the processing of an item operation.
The **server_unavailable** Cheetah entity is a subclass of the **processing_error**, Cheetah entity that is specified in section 2.2.7. Cheetah entity specification for **server_unavailable**:

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

### 2.2.12  cht::documentmessages::operation_dropped

The **operation_dropped** Cheetah entity is used when item processing has identified an item operation that MUST NOT be indexed.

The **operation_dropped** Cheetah entity is a subclass of the **processing_error** Cheetah entity that is specified in section 2.2.7. Cheetah entity specification for **operation_dropped**:

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

### 2.2.13  cht::documentmessages::operation_lost

The **operation_lost** Cheetah entity is used when an item operation has been lost during processing or indexing.

The **operation_lost** Cheetah entity is a subclass of the **error** Cheetah entity that is specified in section 2.2.6. Cheetah entity specification for **operation_lost**:

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

### 2.2.14  cht::documentmessages::indexing_error

The **indexing_error** Cheetah entity is used when an error occurs during the indexing of an item operation.

The **indexing_error** Cheetah entity is a subclass of the **error** Cheetah entity that is specified in section 2.2.6. The **indexing_error** Cheetah entity is a common superclass for the following Cheetah entities:

- **invalid_content**, which is specified in section 2.2.15.
- **resource_error**, which specified in section 2.2.16.
- **unknown_document**, which is specified in section 2.2.17.

Cheetah entity specification for **indexing_error**:

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

### 2.2.15  cht::documentmessages::invalid_content

An indexing node uses the **invalid_content** Cheetah entity when an item operation contains content that is not valid.
The `invalid_content` Cheetah entity is a subclass of the `indexing_error` Cheetah entity that is specified in section 2.2.14. Cheetah entity specification for `invalid_content`:

```cpp
type invalid_content : indexing_error {
};
```

### 2.2.16 `cht::documentmessages::resource_error`

An indexing node uses the `resource_error` Cheetah entity to indicate that a resource error occurred during the indexing of an item operation.

The `resource_error` Cheetah entity is a subclass of the `indexing_error` Cheetah entity that is specified in section 2.2.14. Cheetah entity specification for `resource_error`:

```cpp
type resource_error : indexing_error {
};
```

### 2.2.17 `cht::documentmessages::unknown_document`

An indexing node uses this Cheetah entity when a `remove_operation` Cheetah entity refers to an item that does not exist in the index.

The `unknown_document` Cheetah entity is a subclass of the `indexing_error` Cheetah entity as specified in section 2.2.14. Cheetah entity specification for `unknown_document`:

```cpp
type unknown_document : indexing_error {
};
```

### 2.2.18 `cht::documentmessages::warning`

The `warning` Cheetah entity contains warning information for an item operation with a specific operation identifier. Cheetah entity specification for `warning`:

```cpp
type warning {
    attribute int warning_code;
    attribute string description;
    attribute string subsystem;
    attribute int session_id;
    attribute longint operation_id;
};
```

- **warning_code**: An integer that indicates the warning code.

- **description**: A string that contains a description of the warning.

- **subsystem**: A string that describes where the warning occurred. This string MUST have a value of either "indexing" or "processing". If the warning was produced by either the content distributor or the item processor, the string value will be "processing". If the warning was produced by either the indexing dispatcher or an indexing node, the string value will be "indexing".

- **session_id**: An integer that uniquely identifies the session.
2.2.19  cht::documentmessages::operation

The operation Cheetah entity is a common superclass for the following Cheetah entities:

- **update_operation**, which is specified in section 2.2.39.
- **partial_update_operation**, which is specified in section 2.2.40.
- **remove_operation**, which is specified in section 2.2.41.
- **urlschange_operation** Cheetah entity, which is specified in section 2.2.42.

Cheetah entity specification for operation:

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

- **id**: A long integer that uniquely identifies the item operation. The value MUST be equal to or greater than 0.
- **warnings**: A collection of warning Cheetah entities, which are specified in section 2.2.18. This collection contains all the warnings for the item operation that is identified by the id attribute.

2.2.20  cht::documentmessages::operation_set

The operation_set Cheetah entity contains a set of operation objects, as specified in section 2.2.19. Cheetah entity specification for operation_set:

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

- **completed_op_id**: A long integer that contains the highest operation identifier in the sequence of operation identifiers for which the content client has received all callback messages.
- **operations**: A collection of operation Cheetah entities.

2.2.21  cht::documentmessages::operation_status_info

The operation_status_info Cheetah entity contains status information about a collection of operations. The operation_status Cheetah entity is used to report the status of submitted item operations to the protocol client. Cheetah entity specification for operation_status_info:

```plaintext
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: A long integer that contains the operation identifier of the first operation in the sequence of item operations. This value MUST be equal to or greater than 0 as well as less than or equal to the value of the last_op_id attribute (1).

last_op_id: A long integer that contains the operation identifier of the last operation in the sequence of item operations. This value MUST be equal to or greater than 0 as well as equal to or greater than the value of the first_op_id attribute (1).

state: An operation_state Cheetah enumeration constant, as specified in section 2.2.5, that represents the state of the sequence of item operations.

subsystem: A string that describes where the operation status info was generated. This string MUST have a value of either "indexing" or "processing". If the operation status info was produced by either the content distributor or the item processor, the string value will be "processing". If the operation status info was produced by either the indexing dispatcher or an indexing node, the string value will be "indexing".

events: A collection of error Cheetah entities, which are specified in section 2.2.6 (section 2.2.6). This value contains the errors for the operations that are specified in the collection of item operations.

warnings: A collection of warning Cheetah entities, which are specified in section 2.2.18. This value contains warnings for the operations that are specified in the collection of item operations.

2.2.22 cht::documentmessages::operation_status_info_set

The operation_status_info_set Cheetah entity contains operations status information for a collection of item operation collections. Cheetah entity specification for operation_status_info_set:

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

status: A collection of operation_status_info Cheetah entities, as specified in section 2.2.21

2.2.23 cht::documentmessages::key_value_pair

The key_value_pair Cheetah entity is a common superclass that associates a key with a value that can be one of various types. Cheetah entity specification for key_value_pair:

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

key: A string that contains the key.

2.2.24 cht::documentmessages::key_value_collection

The key_value_collection Cheetah entity forms an association between a single key and a key_value_pair collection.
The `key_value_collection` Cheetah entity is a subclass of the `key_value_pair` Cheetah entity, which is specified in section 2.2.23. Cheetah entity specification for `key_value_collection`:

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

`values`: A collection of `key_value_pair` Cheetah entities.

### 2.2.25 `cht::documentmessages::document_id`

The `document_id` Cheetah entity uniquely identifies an item. The `document_id` Cheetah entity represents the `document identifier (3)` of the item. Cheetah entity specification for `document_id`:

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

`id`: A string that uniquely identifies the item.

`routing_attributes`: Unused. The value MUST be an empty Cheetah collection.

### 2.2.26 `cht::documentmessages::string_attribute`

The `string_attribute` Cheetah entity forms an association between a key and a string value.

The `string_attribute` Cheetah entity is a subclass of the `key_value_pair` Cheetah entity, which is specified in section 2.2.23. Cheetah entity specification for `string_attribute`:

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

`value`: A string that contains the value.

### 2.2.27 `cht::documentmessages::bool_attribute`

The `bool_attribute` forms an association between a string key and a Boolean value.

The `bool_attribute` Cheetah entity is a subclass of the `key_value_pair` Cheetah entity, specified in section 2.2.23. Cheetah entity specification for `bool_attribute`:

```
entity bool_attribute : key_value_pair {
    attribute bool value;
};
```

`value`: A Boolean that contains the value.
2.2.28 **cht::documentmessages::float_attribute**

The *float_attribute* forms an association between a string key and a float value.

The *float_attribute* Cheetah entity is a subclass of the *key_value_pair* Cheetah entity, as specified in section 2.2.23. Cheetah entity specification for *float_attribute*:

```cpp
entity float_attribute : key_value_pair {
    attribute float value;
};
```

**value:** A float that contains the value.

2.2.29 **cht::documentmessages::integer_attribute**

The *integer_attribute* forms an association between a string key and an integer value.

The *integer_attribute* Cheetah entity is a subclass of the *key_value_pair* Cheetah entity that is specified in section 2.2.23. Cheetah entity specification for *integer_attribute*:

```cpp
entity integer_attribute : key_value_pair {
    attribute integer value;
};
```

**value:** An integer that contains the value.

2.2.30 **cht::documentmessages::long_attribute**

The *long_attribute* forms an association between a string key and a *longint* value.

The *long_attribute* Cheetah entity is a subclass of the *key_value_pair* Cheetah entity, as specified in section 2.2.23. Cheetah entity specification for *long_attribute*:

```cpp
entity long_attribute : key_value_pair {
    attribute longint value;
};
```

**value:** A long that contains the value.

2.2.31 **cht::documentmessages::bytearray_attribute**

The *bytearray_attribute* forms an association between a string key and a byte array value.

The *bytearray_attribute* Cheetah entity is a subclass of the *key_value_pair* Cheetah entity, as specified in section 2.2.23. Cheetah entity specification for *bytearray_attribute*:

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

**value:** A Cheetah byte array that contains the value.
2.2.32  cht::documentmessages::string_collection

The string_collection forms an association between a string key and a string collection.

The string_collection Cheetah entity is a subclass of the key_value_pair Cheetah entity, as specified in section 2.2.23. Cheetah entity specification for string_collection:

    entity string_collection : key_value_pair {
        collection string values;
    };

values: A string collection that contains the values.

2.2.33  cht::documentmessages::bool_collection

The bool_collection forms an association between a string key and a Boolean collection.

The bool_collection Cheetah entity is a subclass of the key_value_pair Cheetah entity, as specified in section 2.2.23. Cheetah entity specification for bool_collection:

    entity bool_collection : key_value_pair {
        collection bool values;
    };

values: A Boolean collection that contains the values.

2.2.34  cht::documentmessages::float_collection

The float_collection forms an association between a string key and a float collection.

The float_collection Cheetah entity is a subclass of the key_value_pair Cheetah entity, as specified in section 2.2.23. Cheetah entity specification for float_collection:

    entity float_collection : key_value_pair {
        collection float values;
    };

values: A float collection that contains the values.

2.2.35  cht::documentmessages::integer_collection

The integer_collection forms an association between a string key and a integer collection.

The integer_collection Cheetah entity is a subclass of the key_value_pair Cheetah entity that is specified in section 2.2.23. Cheetah entity specification for integer_collection:

    entity integer_collection : key_value_pair {
        collection integer values;
    };

values: A integer collection containing the values.
2.2.36  cht::documentmessages::long_collection

The long_collection forms an association between a string key and a longint collection.

The long_collection Cheetah entity is a subclass of the key_value_pair Cheetah entity that is specified in section 2.2.23. Cheetah entity specification for long_collection:

```csharp
entity long_collection : key_value_pair {
    collection longint values;
};
```

values: A long collection containing the values.

2.2.37  cht::documentmessages::bytearray_collection

The bytearray_collection forms an association between a string key and a byte array collection.

The bytearray_collection Cheetah entity is a subclass of the key_value_pair Cheetah entity that is specified in section 2.2.23. Cheetah entity specification for bytearray_collection:

```csharp
entity bytearray_collection : key_value_pair {
    collection bytearray values;
};
```

values: A Cheetah byte array collection that contains the values.

2.2.38  cht::documentmessages::document

The document Cheetah entity contains information about one item. Cheetah entity specification for document:

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

doc_id: A document_id Cheetah entity as specified in cht::documentmessages::document_id (section 2.2.25), that uniquely identifies the item.

document_attributes: A collection of key_value_pair Cheetah entities, as specified in section 2.2.23, that contain the attributes (1) of the item.

Crawled properties in the document_attributes collection are converted to managed properties in item processing. See [MS-FSPSCFG] for more information.

The attributes in the following table SHOULD be added to the document_attributes attribute of the doc attribute of the following Cheetah entities:

- cht::documentmessages::update_operation, as specified in cht::documentmessages::update_operation (section 2.2.39).
- cht::documentmessages::partial_update_operation, as specified in cht::documentmessages::partial_update_operation (section 2.2.40).
- **cht::documentmessages::urlschange_operation**, as specified in cht::documentmessages::urlschange_operation (section 2.2.42).

<table>
<thead>
<tr>
<th>Attribute key</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>cht::documentmessages::string_attribute</td>
<td>URL of the item.</td>
</tr>
<tr>
<td>size</td>
<td>cht::documentmessages::integer_attribute</td>
<td>Size of item (bytes).</td>
</tr>
<tr>
<td>crawltime</td>
<td>cht::documentmessages::integer_attribute</td>
<td>Timestamp for when item was last crawled.</td>
</tr>
<tr>
<td>mime</td>
<td>cht::documentmessages::string_attribute</td>
<td>Item MIME type.</td>
</tr>
<tr>
<td>ip</td>
<td>cht::documentmessages::string_attribute</td>
<td>IP of server item was downloaded from.</td>
</tr>
<tr>
<td>http_header</td>
<td>cht::documentmessages::string_attribute</td>
<td>Raw HTTP header including GET line.</td>
</tr>
<tr>
<td>300 redirects</td>
<td>cht::documentmessages::string_collection</td>
<td>Absolute URIs that redirected to this URI as a HTTP status code 300.</td>
</tr>
<tr>
<td>301 redirects</td>
<td>cht::documentmessages::string_collection</td>
<td>Absolute URIs that redirected to this URI as a HTTP status code 301.</td>
</tr>
<tr>
<td>302 redirects</td>
<td>cht::documentmessages::string_collection</td>
<td>Absolute URIs that redirected to this URI as a HTTP status code 302.</td>
</tr>
<tr>
<td>html redirects</td>
<td>cht::documentmessages::string_collection</td>
<td>Absolute URIs that redirected to this URI as a HTML META refresh.</td>
</tr>
<tr>
<td>duplicates</td>
<td>cht::documentmessages::string_collection</td>
<td>Absolute URIs detected as duplicates of this URI.</td>
</tr>
<tr>
<td>encoding</td>
<td>cht::documentmessages::string_attribute</td>
<td>MUST be either &quot;deflate&quot; or None to indicate if data is compressed. If compressed, zlib is used as specified in [RFC1950].</td>
</tr>
<tr>
<td>crawllinks</td>
<td>cht::documentmessages::string_collection</td>
<td>Absolute URIs extracted as links from this item.</td>
</tr>
<tr>
<td>mirrorsites</td>
<td>cht::documentmessages::string_collection</td>
<td>FQDN mirrors, either automatically detected or specified by configuration.</td>
</tr>
<tr>
<td>referrers</td>
<td>cht::documentmessages::string_collection</td>
<td>Referrer URLs in the item.</td>
</tr>
<tr>
<td>data</td>
<td>cht::documentmessages::bytearray_attribute</td>
<td>Item data. Content that is not represented as other key-value pairs.</td>
</tr>
<tr>
<td>extra_data</td>
<td>cht::documentmessages::bytearray_attribute</td>
<td>Attribute used for RSS feeds and sitemaps, as specified in the following table.</td>
</tr>
</tbody>
</table>

The **extra_data** attribute is encoded using the data structure as specified in [MS-FSWCU]. The **extra_data** attribute SHOULD contain the keys in the following table for web items.
<table>
<thead>
<tr>
<th>Key name</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs</td>
<td>dictionary</td>
<td>rs data dictionary, as specified in the following table.</td>
</tr>
<tr>
<td>sm</td>
<td>dictionary</td>
<td>sitemap dictionary, as specified in the following table.</td>
</tr>
</tbody>
</table>

The *rs* data dictionary MAY contain the keys described in the following table.

<table>
<thead>
<tr>
<th>Key name</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dictionary</td>
<td>Item <em>rss</em> dictionary, as specified in the following table.</td>
</tr>
</tbody>
</table>

The item *rss* dictionary MAY contain the keys described in the following table.

<table>
<thead>
<tr>
<th>Key name</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gl</td>
<td>dictionary</td>
<td>Global dictionary, as specified in the following table.</td>
</tr>
<tr>
<td>lo</td>
<td>dictionary</td>
<td>Local dictionary, as specified in the following table.</td>
</tr>
</tbody>
</table>

The global dictionary MAY contain the keys in the following table.

<table>
<thead>
<tr>
<th>Key name</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>string</td>
<td>Attribute value is the tag value in the <em>channel</em> section of the referring RSS feed.</td>
</tr>
</tbody>
</table>

The local dictionary MAY contain the keys in the following table.

<table>
<thead>
<tr>
<th>Key name</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>string</td>
<td>Attribute value is the tag value in the <em>item</em> section of the referring RSS feed.</td>
</tr>
</tbody>
</table>

The *sm* dictionary MAY contain the keys in the following table.

<table>
<thead>
<tr>
<th>Key name</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>string</td>
<td>Attribute value is the tag value in the <em>url</em> section of the sitemap document describing this specific item.</td>
</tr>
</tbody>
</table>

The attribute in the following table that specifies the permissions of an item MUST be added to the *document_attributes* attribute of the *doc* attribute of the following Cheetah entities:

- *cht::documentmessages::update_operation*, as specified in section 2.2.39.
- *cht::documentmessages::partial_update_operation*, as specified in section 2.2.40.
- *cht::documentmessages::urlschange_operation*, as specified in section 2.2.42.
One of the attributes in the following table that specifies the permissions of an item MUST be added to the `document_attributes` attribute of the `doc` attribute of the following Cheetah entities:

- `cht::documentmessages::update_operation`, as specified in section 2.2.39.
- `cht::documentmessages::partial_update_operation`, as specified in section 2.2.40.
- `cht::documentmessages::urlschange_operation`, as specified in section 2.2.42.

<table>
<thead>
<tr>
<th>Attribute key</th>
<th>Attribute type</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>docaclsystemids</td>
<td><code>cht::documentmessages::string_attribute</code></td>
<td>The <code>docaclsystemids</code> field is a user store identifier of the item.</td>
</tr>
</tbody>
</table>

The `docacl` attribute MUST contain a list of space separated entries in the following format: `<deniedRightFlag><userStoreID><securityIdentifier>`. `<deniedRightFlag>` is 9 if the rest indicates a deny permission or `""` if it is a grant permission. `<userStoreID>` is the user store identifier of the user/group given in `<securityIdentifier>`. `<securityIdentifier>` is the user or group security principal identifier that was granted or denied permission to the item. A `docacl` value MUST only contain alphanumeric characters. If the `<securityIdentifier>` contains other characters, the `<securityIdentifier>` MUST be encoded with a base-32 variant of [RFC4648] using an alphabet with `a-z` and `1-6` and no equal sign (=) padding at the end.

The `docaclms` field MUST contain security descriptor strings. See [MSDN-SDDL].

The `spacl` value contains a list of binary Ace values. Ace value is specified in the following table.

<table>
<thead>
<tr>
<th>Access type (1 byte): 0 if allowed; 1 if denied.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE type (1 byte): 0 if nt; 1 if claim.</td>
</tr>
<tr>
<td>ACE type data (variable): If ACE type is 0, ACE nt value. If ACE type is 1, ACE claim value.</td>
</tr>
</tbody>
</table>

---

*[MS-FSCF] — v20120630
Content Feeding Protocol Specification

Copyright © 2012 Microsoft Corporation.

Release: July 16, 2012*
The **ACE nt value** is specified in the following table.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 1 |
| SID length |
| SID (variable) |

**SID length (4 bytes):** A variable of type INT32 that represents the length of the SID string.

**SID (variable):** An UTF-16 string that represents the security identifier (SID).

The **ACE claim value** is specified in the following table.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 1 |
| Value length |
| Value (variable) |
| ... |
| ClaimType length |
| ClaimType (variable) |
| ... |
| ValueType length |
| ValueType(variable) |
| ... |
| OriginalIssuer length |
| OriginalIssuer(variable) |
| ... |

**Value length (4 bytes):** A variable of type INT32 that represents the length of the claim value.

**Value (variable):** A UTF-16 string that represents the claim value.
ClaimType length (4 bytes): A variable of type INT32 that represents the length of the claim type.

ClaimType (variable): A UTF-16 string that represents the claim type.

ValueType length (4 bytes): A variable of type INT32 that represents the length of the value type.

ValueType (variable): A UTF-16 string that represents the value type.

OriginalIssuer length (4 bytes): A variable of type INT32 that represents the length of the original issuer.

OriginalIssuer (variable): A UTF-16 string that represents the original issuer.

2.2.39 cht::documentmessages::update_operation

The update_operation Cheetah entity adds or replaces a specific item in the index. If an item with the specified document id already exists in the search index, it is replaced.

The update_operation Cheetah entity is a subclass of the Cheetah entity operation that is specified in section 2.2.19. Cheetah entity specification for update_operation:

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

doc: A document Cheetah entity, as specified in section 2.2.38, that represents the item to add or replace.

2.2.40 cht::documentmessages::partial_update_operation

The partial_update_operation Cheetah entity updates one or more of the attributes (1) of a specific item in the search index.

The update_operation Cheetah entity is a subclass of the Cheetah entity operation that is specified in section 2.2.19. Cheetah entity specification for partial_update_operation:

```cpp
entity partial_update_operation : operation {
    attribute document doc;
};
```

The partial_update_operation has the following attribute:

doc: A document Cheetah entity, as specified in section 2.2.38, that contains attributes to update.

2.2.41 cht::documentmessages::remove_operation

The remove_operation Cheetah entity removes a specific item from the search index.

The update_operation Cheetah entity is a subclass of the Cheetah entity operation that is specified in section 2.2.19. Cheetah entity specification for remove_operation:

```cpp
entity remove_operation : operation {
    attribute document_id doc_id;
};
```
2.2.42  cht::documentmessages::urlschange_operation

The urlschange_operation Cheetah entity updates one or more of the attributes (1) of a specific item in the search index.

The urlschange_operation Cheetah entity is a subclass of the Cheetah entity partial_update_operation that is specified in section 2.2.40. Cheetah entity specification for urlschange_operation:

entity urlschange_operation : partial_update_operation {
};

2.2.43  cht::documentmessages::subsystem_id_set

The subsystem_id_set Cheetah entity contains a collection of names. Cheetah entity specification for subsystem_id_set is as follows:

entity subsystem_id_set {
  collection string ids;
};

ids: A collection that MUST consist of either an empty Cheetah collection or a single element that contains the string "indexing".

2.2.44  core::unsupported_guarantee_set

The unsupported_guarantee_set exception specifies that the protocol server is unable to create or recreate a session. The unsupported_guarantee_set exception is specified by the following FSIDL specification:

exception unsupported_guarantee_set {
  string what;
};

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

2.2.45  coreprocessing::timed_out

The timed_out exception states that the protocol server is unable to find an available item processor before a given timeout. The timed_out exception is specified by the following FSIDL specification:

exception timed_out {
  long id;
  string message;
};
id: A long variable that contains the identifier for the exception.

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

### 2.2.46 coreprocessing::service_unavailable

The `service_unavailable` exception states that the protocol server is unable to perform the method invocation. The `service_unavailable` exception is specified by the following FSIDL specification:

```idl
type service_unavailable {
  long id;
  string message;
};
```

id: A long variable that contains the identifier for the exception.

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

### 2.2.47 coreprocessing::format_error

The `format_error` exception indicates that an argument to a method invocation has format that is not valid. The `format_error` exception is specified by the following FSIDL specification:

```idl
type format_error {
  long id;
  string message;
};
```

id: A long variable that contains the identifier for the exception.

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

### 2.2.48 coreprocessing::no_resources

The `no_resources` exception states that the protocol server does not have any resources available to process the method invocation. The `no_resources` exception is specified by the following FSIDL specification:

```idl
type no_resources {
  long id;
  string message;
};
```

id: A long variable that contains the identifier for the exception.

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

### 2.2.49 coreprocessing::unknown_collection_error

The `unknown_collection_error` states that the content collection is unknown. The `unknown_collection_error` exception is specified by the following FSIDL specification:

```idl
type unknown_collection_error {
  long id;
  string message;
};
```
The operation_failed exception states that the protocol server is unable to perform the given operation. The operation_failed exception is specified by the following FSIDL specification:

```idl
exception operation_failed {
    long id;
    string message;
};
```

**id:** Identifier of the exception.

**message:** String explaining cause of exception.
3 Protocol Details

This protocol consists of the three interfaces `processing::session_factory`, `processing::session`, and `coreprocessing::control`. For these three interfaces, the content client acts as the protocol client and the content distributor acts as the protocol server.

The protocol client communicates synchronously with a protocol server, setting up a new `processing::session` using the `processing::session_factory` interface, then feeding item operations and receiving callback messages using this `processing::session` interface. The protocol client uses the `coreprocessing::control` interface to remove all items in a content collection.

The protocol server of this protocol MUST implement the three interfaces `processing::session_factory`, as specified in section 3.2, `processing::session`, as specified in section 3.4, and `coreprocessing::control`, as specified in section 3.6.

The client side of the `processing::session_factory` interface is specified in section 3.3. The client side of the `processing::session` interface is specified in section 3.5. The client side of the `coreprocessing::control` interface is simply a pass-through. That is, no additional timers or other state is required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.1 Common Server Details

The protocol client communicates with a protocol server as part of a larger session based feeding chain between a protocol client, protocol server, indexing dispatcher, and indexing node as specified in Overview (section 1.3).

3.1.1 Abstract Data Model

This section specifies a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The specified 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 specified in this document.

The abstract data model specified is common for the implementation of the three protocol server interfaces `processing::session_factory`, as specified in section 3.2, `processing::session`, as specified in section 3.4, and `coreprocessing::control`, as specified in section 3.6.

The protocol server maintains the following states:

**session factory holder:** A `processing::session_factory` server object used by protocol clients that employ unencrypted HTTP transport, as defined in section 2.1.

**session factory external client holder:** A `processing::session_factory` server object used by protocol clients that employ encrypted HTTPS transport, as defined in section 2.1.

**session holder:** A set of `processing::session` server objects, where each server object can be referenced by a `session id` value. The `session id` value is generated in the `session id generator`.

**session id generator:** An integer used to generate unique identifiers for `processing::session` server objects. At startup, protocol server MUST call `coreprocessing::session_factory::get_highest_session_id`, as specified in [MS-FSDP] section 3, to receive the initial value for the `session id generator`. The initial value is the return value from invoking `coreprocessing::session_factory::get_highest_session_id` +1. For each invocation of
the `processing::session_factory::create` method, as specified in `processing::session_factory::create` (section 3.2.4.2), the value is increased by 1.

**recreating holder:** A collection of **session ids**. Contains **session ids** of sessions being recreated.

**master mode:** A Boolean value that specifies whether the protocol server is running in master mode or in dispatcher mode. When there are multiple protocol servers, one protocol server MUST be running in master mode at any given time. If there is only one protocol server, it MUST be running in master mode all the time. [MS-FSCDFT] specifies the protocol the protocol server uses to switch between dispatcher and master mode.

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

**collection suspended holder:** Contains a collection of strings that represent content collection names. The protocol server MUST use **collection suspended holder** to keep track of content collections where feeding has been suspended.

**operation timeout:** An integer that contains a timeout for the protocol server that specifies the maximum time to find an available item processor for a sequence of item operations.

For each `processing::session` server object in the **session holder** state, the protocol server MUST maintain the following states:

**operation id generator:** An integer state. The protocol server MUST use the **operation id generator** to assign increasing item operation identifier numbers to item operations received from the `processing::session` protocol client.

**callback holder:** A collection that contains callback messages received by item processor, indexing dispatcher and callback messages generated by protocol server. The protocol server MUST add callback messages to this state when they are received in the `coreprocessing::operation_callback::status_changed` method as specified in [MS-FSDP] section 3. Callback messages generated by the content distributor when an item processor is restarted, as specified in [MS-FSDP] section 3, MUST also be added to this state.

**session id:** An integer state that contains the `processing::session` server object identifier.

**content collection id:** A string state that contains the content collection name of the `processing::session` server object.

### 3.1.2 Timers

None.

### 3.1.3 Initialization

None.

### 3.1.4 Message Processing Events and Sequencing Rules

None.

### 3.1.5 Timer Events

None.
3.1.6 Other Local Events

None.

3.2 processing::session_factory Server Details

3.2.1 Abstract Data Model

For more information about common abstract data model for protocol server, see Abstract Data Model (section 3.1.1).

3.2.2 Timers

None.

3.2.3 Initialization

The protocol server MUST initialize two processing::session_factory server objects.

The processing::session_factory server object to be inserted in the session factory holder MUST be initialized using the following abstract object reference (AOR), as specified in [MS-FSMW] section 2.

object_id: An integer value that MUST be 1.

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

port: The base port plus 390.

interface_type: A string value that MUST be "processing::session_factory".

interface_version: A string value that MUST be "5.1".

The processing::session_factory server object to be inserted in the session factory external client holder MUST be initialized using the following AOR, as specified in [MS-FSMW] section 2.

object_id: An integer value that MUST be 1.

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

port: The base port plus 391.

interface_type: A string value that MUST be "processing::session_factory".

interface_version: A string value that MUST be "5.1".

The protocol server MUST initialize the processing::session_factory server objects as specified in [MS-FSMW] section 3.

3.2.4 Message Processing Events and Sequencing Rules

The processing::session_factory interface specifies the methods that are listed in the following table.
### 3.2.4.1 processing::session_factory::is_master

A protocol server MUST be running in one of two modes, either master mode or dispatcher mode. This method checks what mode the protocol server currently is running in. The method is specified by the following FSIDL specification:

```fsidl
boolean is_master();
```

**Return value:** true if protocol server is running in master mode, false if the protocol server is running in dispatcher mode.

**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 master_mode state.

[MS-FSCDFT] specifies the protocol that the protocol server uses to switch between dispatcher and master mode.

### 3.2.4.2 processing::session_factory::create

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

```fsidl
processing::session create(in string collection,
                           in cht::core::guarantee_set guarantees)
raises (coreprocessing::unknown_collection_error,
        core::unsupported_guarantee_set);
```

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

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

**Return value:** A processing::session client proxy instantiated with the AOR specified in section 3.5.3.

**Exceptions**

coreprocessing::unknown_collection_error: Raised if the specified content collection is unknown. Raised if feeding to this content collection has been suspended, as specified in section 3.6.4.1.
core::unsupported_guarantee_set: Raised if the protocol server is unable to create the feeding session.

When the protocol server receives a create method invocation it MUST create and return a new processing::session client proxy to the protocol client and activate the new processing::session server object. The client proxy returned MUST be instantiated with the AOR specified in Initialization (section 3.4.3).

If the create method invocation is received on the processing::session_factory server object in the session factory holder, the processing::session server object MUST be activated on base port plus 390.

If the create method invocation is received on the processing::session_factory server object in the session factory external client holder, the processing::session server object MUST be activated on base port plus 391.

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

The protocol server MUST verify that the collection input value does not exist in the collection suspended holder state. If a collection input value does exist in the collection suspended holder state, the protocol server MUST raise a coreprocessing::unknown_collection_error exception.

The protocol server MUST use the session id generator state to create a new session id for the new processing::session server object.

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

The protocol server MUST be running in master mode.

3.2.4.3 processing::session_factory::recreate

Recreate a feeding session with a given identifier. A session with this session id MUST have been created previously with processing::session_factory::create, as specified in section 3.2.4.2. The method is specified by the following FSIDL specification:

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

id: Identifier for session. Identifier MUST be equal to identifier for session already created.

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

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

Return value: A processing::session client proxy instantiated with the AOR specified in Initialization (section 3.4.3).
Exceptions

`coreprocessing::unknown_collection_error`: Raised if the specified content collection is unknown.

`core::unsupported_guarantee_set`: Raised if the protocol server is unable to create the feeding session. Raised if feeding to this content collection has been suspended, as specified in section 3.6.4.1.

When the protocol server receives a `recreate` method invocation, the protocol server MUST validate the `session holder` state. If the `session holder` state contains a `coreprocessing::session` server object with the specified `session id`, the protocol server MUST return a client proxy to the existing `coreprocessing::session` server object. If no session with the specified `session id` exists, the protocol server MUST create and return a new `coreprocessing::session` client proxy to the protocol client and then activate the new `coreprocessing::session` server object. The client proxy returned MUST be instantiated with the AOR specified in section 3.4.3.

If the `recreate` method invocation is received on the `coreprocessing::session_factory` server object in the `session factory holder`, the `coreprocessing::session` server object MUST be activated on base port plus 390.

If the `recreate` method invocation is received on the `coreprocessing::session_factory` server object in the `session factory external client holder`, the `coreprocessing::session` server object MUST be activated on base port plus 391.

The protocol server MUST check that the `collection` input value exists in the `collection holder` state. If `collection` input value does not exist in the `collection holder` state, the exception `coreprocessing::unknown_collection_error` is raised.

The protocol server MUST verify that the `collection` input value does not exist in the `collection suspended holder` state. If a `collection` input value does exist in the `collection suspended holder` state, the protocol server MUST raise a `coreprocessing::unknown_collection_error` exception.

If the `id` input parameter exists in the recreating holder state, the protocol server MUST raise a `core::unsupported_guarantee_set` exception.

When the recreate method is called, the protocol server MUST call the `coreprocessing::session_factory::recreate` method in the indexing dispatcher, as specified in [MS-FSDP]. If the `coreprocessing::session_factory::recreate` method raises an exception, the protocol server MUST add the `id` input parameter to the recreating holder state and raise the exception `core::unsupported_guarantee_set`. The protocol server MUST continue to call the `coreprocessing::session_factory::recreate` method in the indexing dispatcher. When no exception is raised when invoking the `coreprocessing::session_factory::recreate` method, the protocol server MUST remove the `id` input parameter from the recreating holder state.

The protocol server MUST store the `coreprocessing::session` server object in a `session holder` state, with the `session_id` input value as unique key.

The protocol server MUST be running in master mode.

### 3.2.4.4 `coreprocessing::session_factory::close`

The `close` method closes a session in the protocol server. A session with the specified identifier MUST have been created earlier with the `processing::session_factory::create`, as specified in section 3.2.4.2. The method is specified by the following FSIDL specification:
void close(in long id);

id: Identifier for session. The identifier MUST represent an already created session.

Return value: None.

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

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

The protocol server MUST be running in master mode.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 processing::session_factory Client Details

3.3.1 Abstract Data Model

This section specifies a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The specified 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 specified in this document.

The content client, acting as protocol client for the processing::session_factory interface, MUST maintain the following states for each session instance:

session client holder: A state containing a collection of processing::session client proxies.

3.3.2 Timers

None.

3.3.3 Initialization

The processing::session_factory client is a client proxy that calls remote methods on a processing::session_factory server. To call remote methods on the server, a client MUST first create the client proxy based on an AOR to the processing::session_factory server object, as specified in section 3.2.

The AOR for the processing::session_factory server object MUST be provided by a higher layer of the implementation as an AbstractObjectReference record, as specified in [MS-FSMW] section 2. The InterfaceType field of the AbstractObjectReference record MUST be "processing::session_factory", the ServerObjectId field MUST be 0, and the InterfaceVersion MUST be "5.1". If using unencrypted transport, as specified in section 2.1, the Port field of the

[MS-FSCF] — v20120630
Content Feeding Protocol Specification

Copyright © 2012 Microsoft Corporation.

Release: July 16, 2012
AbstractObjectReference MUST be base port plus 390. If using encrypted transport, as specified in section 2.1, the Port field of the AbstractObjectReference MUST be base port plus 391.

3.3.4 Message Processing Events and Sequencing Rules

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_master</td>
<td>Check if protocol server is running in master mode.</td>
</tr>
<tr>
<td>create</td>
<td>Returns a new processing::session client proxy.</td>
</tr>
<tr>
<td>recreate</td>
<td>Returns a processing::session client proxy with a given identifier.</td>
</tr>
<tr>
<td>close</td>
<td>Closes and removes a session.</td>
</tr>
</tbody>
</table>

The protocol client MUST call the is_master method prior to any of the other methods, and only call create, recreate or close if is_master returns true.

3.3.4.1 processing::session_factory::is_master

The is_master method is specified in processing::session_factory::is_master (section 3.2.4.1).

3.3.4.2 processing::session_factory::create

The create method is specified in processing::session_factory::create (section 3.3.4.2). The return value of the create method MUST be stored in the session client holder.

3.3.4.3 processing::session_factory::recreate

The recreate method is specified in processing::session_factory::create (section 3.2.4.2). The return value of the recreate method MUST be stored in the session client holder. If the protocol server raises an core::unsupported_guarantee_set exception when the protocol client calls this method, the protocol client MUST continue to call the recreate method on the protocol server until this exception is no longer raised.

3.3.4.4 processing::session_factory::close

The close method is specified in section processing::session_factory::close (section 3.2.4.4).

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 processing::session Server Details

3.4.1 Abstract Data Model

For more information about common abstract data model for the protocol server, see section 3.1.1.
3.4.2 Timers

The operation timeout timer measures the time it takes for the protocol server to find an available item processor when protocol server receives a `processing::session::process` method invocation as specified in section 3.4.4.5. The default value for the operation timeout is 60 seconds.

3.4.3 Initialization

The protocol server MUST initialize the `processing::session` server object using the following AOR, as specified in [MS-FSMW]

- **object_id**: The value is implementation specific of the higher level application.
- **host**: A string that contains the host name of the server object on the protocol server. The value is implementation specific of the higher level application.
- **port**: If the `processing::session` server object is created by the server object in the `session factory holder`, the port MUST be base port plus 390. If the `processing::session` server object is created by the server object in the `external session factory holder`, the port MUST be base port plus 391.
- **interface_type**: A string value that MUST be "processing::session".
- **interface_version**: A string value that MUST be "5.2".

The protocol server MUST initialize the `processing::session` server object as specified in [MS-FSMW] section 3.

3.4.4 Message Processing Events and Sequencing Rules

The `processing::session` interface specifies the methods that are listed in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_operation_timeout</td>
<td>Returns timeout used in process method.</td>
</tr>
<tr>
<td>get_session_id</td>
<td>Returns identifier of session interface.</td>
</tr>
<tr>
<td>get_system_ids</td>
<td>Returns a fixed string.</td>
</tr>
<tr>
<td>poll_callbacks</td>
<td>Returns callback messages.</td>
</tr>
<tr>
<td>process</td>
<td>Process and index a sequence of item operations. Callback messages are returned.</td>
</tr>
</tbody>
</table>

3.4.4.1 processing::session::get_operation_timeout

The `get_operation_timeout` method returns the timeout, in seconds, that the protocol server uses in the `process::session::process` method (section 3.4.4.5). The method is specified by the following FSIDL specification:

```idl
long get_operation_timeout()
```

**Input values**: None.

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

When the protocol server receives a get_operation_timeout method invocation on the processing::session server object, it MUST return the value of the operation timeout state.

3.4.4.2 processing::session::get_session_id

The get_session_id method returns the identifier of the session. The method is specified by the following FSIDL specification:

```idl
long get_session_id();
```

Return value: Identifier of the session.

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

When the protocol server receives a get_session_id method invocation on the processing::session server object, it MUST return the value of the session id state.

3.4.4.3 processing::session::get_system_ids

The get_system_ids method returns a description of callback messages given by the protocol server and the indexer. The method is specified by the following FSIDL specification:

```idl
cht::documentmessages::subsystem_id_set get_system_ids();
```

Return value: The ids collection of the cht::documentmessages::subsystem_id_set, as specified in section 2.2.42, MUST contain one string that contains the value "processing:0:1,indexing:1:1". This string describes the callback messages given as defined in section 3.4.4.4.

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

3.4.4.4 processing::session::poll_callbacks

The poll_callbacks method returns callback messages for submitted item operations. The protocol server MUST return all new callback messages received from indexer and item processor and all callback messages generated by protocol server from the last processing::session::process or processing::session::poll_callbacks method invocation. The method is specified by the following FSIDL specification:

```idl
cht::documentmessages::operation_status_info_set poll_callbacks();
```

Return value: cht::documentmessages::operation_status_info_set, as specified in section 2.2.22 containing new callbacks generated by indexing dispatcher, item processor and protocol server from the last session::poll_callback or session::process method invocation, as specified in section 3.4.4.5. Each callback message is represented as one cht::documentmessages::operation_status_info, Cheetah entity, as specified in section 2.2.21. If there are no callbacks, the operations attribute of the
cht::documentmessages::operation_status_info_set Cheetah entity MUST be an empty collection.

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

When the protocol server receives a poll_callbacks method invocation on the processing::session server object, it MUST return the callback holder state. The value of the callback holder state MUST be reset after the return has been sent to the protocol client.

For each session::process method invocation, as specified in section processing::session::process (section 3.4.4.5), the following callback messages are generated by the protocol server, the item processor and the indexing dispatcher.

Completed by processing

The callback message received by the protocol server when the item processor has finished processing a sequence of item operations submitted in one processing::session::process method invocation, as specified in section 3.4.4.5. The cht::documentmessages::operation_status_info Cheetah entity that represents the callback message contains the following attributes:

- **first_op_id**: Item operation identifier assigned by the protocol server to the first item operation in the submitted sequence of item operations.
- **last_op_id**: Item operation identifier assigned by the protocol server to the last item operation in the submitted sequence of item operations.
- **state**: The value for this attribute MUST be Cheetah enumeration value cht::documentmessages::completed, as specified in section 2.2.5.
- **subsystem**: A string that MUST have the value "processing".
- **errors**: Errors for item operations provided by the item processor. The operation_id attribute of the cht::documentmessages::error Cheetah entity identifies the item operation to which an error refers to. The protocol server assigns an item operation identifier to each item operation in the processing::session::process method, as specified in section 3.4.4.5.
- **warnings**: Warnings for item operations provided by the item processor. The operation_id attribute of the cht::documentmessages::warning Cheetah entity identifies the item operation to which an warning refers to. The protocol server assigns an item operation identifier to each item operation in the processing::session::process method, as specified in section 3.4.4.5.

Secured by indexing

The callback message received by the protocol server when the indexing nodes have stored the sequence of item operations submitted in one processing::session::process method invocation, as specified in section 3.4.4.5, to disk. The cht::documentmessages::operation_status_info Cheetah entity that represents the callback message contains the following attributes:

- **first_op_id**: Item operation identifier assigned by the protocol server to the first item operation in the submitted sequence of item operations.
- **last_op_id**: Item operation identifier assigned by the protocol server to the last item operation in the submitted sequence of item operations.
- **state**: The value for this attribute MUST be Cheetah enumeration value cht::documentmessages::secured, as specified in section 2.2.5.
- **subsystem**: A string that MUST have the value "indexing".

- **errors**: Errors for item operations provided by the indexing dispatcher. The `operation_id` attribute of the `cht::documentmessages::error` Cheetah entity identifies the item operation to which an error refers to. The protocol server assigns an item operation identifier to each item operation in the `processing::session::process` method, as specified in section 3.4.4.5.

- **warnings**: Warnings for item operations provided by the indexing dispatcher. The `operation_id` attribute of the `cht::documentmessages::warning` Cheetah entity identifies the item operation to which an warning refers to. The protocol server assigns an item operation identifier to each item operation in the `processing::session::process` method, as specified in section 3.4.4.5.

### Completed by indexing

The callback message received by the protocol server when the indexing nodes have processed the sequence of item operations, submitted in one `processing::session::process` method invocation, as specified in section 3.4.4.5, and the actions that were triggered by the item operations are visible in the search index. The `cht::documentmessages::operation_status_info` Cheetah entity that represents the callback message contains the following attributes:

- **first_op_id**: Item operation identifier assigned by the protocol server to the first item operation in the submitted sequence of item operations.

- **last_op_id**: Item operation identifier assigned by the protocol server to the last item operation in the submitted sequence of item operations.

- **state**: The value for this attribute MUST be Cheetah enumeration value `cht::documentmessages::completed`, as specified in section 2.2.5.

- **subsystem**: A string that MUST have the value "indexing".

- **errors**: Errors for item operations provided by the indexing dispatcher. The `operation_id` attribute of the `cht::documentmessages::error` Cheetah entity identifies the item operation to which an error refers to. The protocol server assigns an item operation identifier to each item operation in the `processing::session::process` method, as specified in section 3.4.4.5.

- **warnings**: Warnings for item operations provided by the indexing dispatcher. The `operation_id` attribute of the `cht::documentmessages::warning` Cheetah entity identifies the item operation to which an warning refers to. The protocol server assigns an item operation identifier to each item operation in the `processing::session::process` method, as specified in section 3.4.4.5.

### 3.4.4.5 `processing::session::process`

The protocol client sends a sequence of item operations to the protocol server for processing and indexing. The return value contains the item operation identifier for the item operations, and callbacks for previously submitted item operation sequences. The method is specified by the following FSIDL specification:

```idl
cht::documentmessages::operation_status_info_set
    process(in cht::documentmessages::operation_set batch,
             in cht::documentmessages::subsystem_id_set subsystems)
    raises (coreprocessing::timed_out,
            coreprocessing::service_unavailable,
            coreprocessing::format_error,
            coreprocessing::no_resources);
```

---

**[MS-FSCF] — v20120630**  
*Content Feeding Protocol Specification*  

*Copyright © 2012 Microsoft Corporation.*  

*Release: July 16, 2012*
**batch**: Sequence of item operations.

**subsystems**: The `ids` attribute of the `cht::documentmessages::subsystem_id_set` Cheetah entity MUST be empty collection or contain the string "indexing".

**Return value**: A `cht::documentmessages::operation_status_info_set` Cheetah entity, as specified in section 2.2.22, containing operation identifier of submitted operations and callback messages received from indexer, item processor and protocol server from the time when the `processing::session::process` or `processing::session::poll_callbacks` method was last called. The first `cht::documentmessages::operation_status_info` Cheetah entity in the `operations` attribute of the `cht::documentmessages::operation_status_info_set` describes the operation identifiers assigned to the submitted item operations. The first `cht::documentmessages::operation_status_info` Cheetah entity in the `operations` attribute MUST have the following attributes:

- **first_op_id**: Item operation identifier assigned by the protocol server to the first item operation in the submitted sequence of item operations. The next item operation in the submitted sequence of item operations has operation identifier `first_op_id` plus 1 and so on.
- **last_op_id**: An integer that MUST be -1.
- **state**: The value for this attribute MUST be `cht::documentmessages::completed`, as specified in section 2.2.5.
- **subsystem**: A string that MUST have the value "firstopid".
- **errors**: The value for this attribute MUST be empty collection.
- **warnings**: The value for this attribute MUST be empty collection.

The rest if the `cht::documentmessages::operation_status_info` Cheetah entities in the `cht::documentmessages::operation_status_info_set` Cheetah entity are callback as specified in section 3.4.4.4.

**Exceptions**

- **coreprocessing::timed_out**: Raised if the protocol server is unable to submit item operations to an item processor before operation timeout is reached, as specified in section 3.4.2. For more information about the protocol between item processor and protocol server, see [MS-FSDPD].
- **coreprocessing::service_unavailable**: Raised if the session has been closed or when the protocol server is shutting down.
- **coreprocessing::format_error**: Raised if submitted sequence of item operations does not contain any operations.
- **coreprocessing::no_resources**: Raised if no item processor is registered in the protocol server. For more information about the protocol between the item processor and the protocol server, see [MS-FSDPD].

When the protocol server receives a `process` method invocation on the `processing::session` server object, it MUST use the `operation id generator` to assign unique operation identifiers to the item operations in the `batch` input value.

The protocol server MUST return the `callback holder` state. The value of the `callback holder` state MUST be reset after the return value has been sent to the protocol client.
3.4.5 Timer Events

The operation timeout event terminates the `processing::session::process` method invocation by raising exception `coreprocessing::timed_out`.

3.4.6 Other Local Events

None.

3.5 `processing::session` Client Details

3.5.1 Abstract Data Model

This section specifies a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The specified 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 specified in this document.

The protocol client maintains the following states:

- **item operation map**: A key-value map for sequences of item operations. The key is a integer value, value is a structure named `item operation callback holder` with the following attributes:
  - **submitted_operations**: A Cheetah entity `cht::documentmessages::operation_set`, as specified in section 2.2.20. Contains a sequence of item operations submitted by invoking the in `processing::session::process` method, as specified in section 3.4.4.5.
  - **num_callbacks**: An integer that represents the number of callback messages received for the `submitted_operations` sequence of item operations.
  - **num_errors**: An integer that represents the number of errors received in callback messages for the `submitted_operations` sequence of item operations.

- **item operation identifier to document identifier map**: The protocol client MUST keep a map from document identifier of submitted items to the item operation identifier assigned by the protocol server in the `processing::session::process` method, as specified in section 3.4.4.5.

- **session id**: An integer that represents the session.

3.5.2 Timers

None.

3.5.3 Initialization

The protocol client that uses the `processing::session` interface MUST create the `session` client proxy using the `processing::session_factory::create` method, as specified in section 3.2.4.2 or the `processing::session_factory::recreate` method, as specified in section 3.2.4.3.

3.5.4 Message Processing Events and Sequencing Rules

This `processing::session` interface specifies the methods that are listed in the following table.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_operation_timeout</td>
<td>Returns timeout used in process method.</td>
</tr>
<tr>
<td>get_session_id</td>
<td>Returns identifier of session interface.</td>
</tr>
<tr>
<td>get_system_ids</td>
<td>Returns a fixed string.</td>
</tr>
<tr>
<td>poll_callbacks</td>
<td>Returns callback messages.</td>
</tr>
<tr>
<td>process</td>
<td>Process and index a sequence of item operations. Callback messages are</td>
</tr>
<tr>
<td></td>
<td>returned.</td>
</tr>
<tr>
<td>__ping</td>
<td>Tests whether a server object is available.</td>
</tr>
</tbody>
</table>

### 3.5.4.1 processing::session::get_operation_timeout

The `get_operation_timeout` method is defined in section [3.4.4.1](#).

### 3.5.4.2 processing::session::get_session_id

The `get_session_id` method is specified in section [3.4.4.2](#). After invoking `create`, as defined in section [3.3.4.2](#), the protocol client MUST call `get_session_id` to update the `session id` state.

### 3.5.4.3 processing::session::get_system_ids

The `get_system_ids` method is specified in section [3.4.4.3](#).

### 3.5.4.4 processing::session::poll_callbacks

The `poll_callbacks` method is specified in processing::session::poll_callbacks (section [3.4.4.4](#)). When the protocol client sends the `processing::session::poll_callbacks` method invocation, as specified in section [3.4.4.4](#), it MUST use the operation id to document identifier mapping table to map operation identifiers in errors and warnings in callback messages to document identifiers of item operations.

The protocol client MUST update the item operation map as specified in section [3.4.4.5](#).

### 3.5.4.5 processing::session::process

The `process` method is specified in section [3.4.4.5](#). When the protocol client calls the `processing::session::process` on the protocol server, as specified in section [3.4.4.5](#), the protocol client MUST use the return value to update the item operation identifier to document identifier mapping state.

The protocol client MUST insert an entry into the item operation map. The key is the first_op_id integer attribute of the first `cht::documentmessages::operation_status_info` Cheetah entity in the status attribute of the `cht::documentmessages::operation_status_info_set` return value of the process method, as specified in section [3.4.4.5](#). The value associated with the key in the item operation map is an operation_set_status structure, as defined in section [3.5.1](#), with the following attributes:

- **submitted_operations** is set to the value of the `cht::documentmessages::operation_set` input value to the `processing::session::process` method.

- **num_callbacks** is set to 0.
• num_errors is set to 0.

For the other cht::documentmessages::operation_status_info Cheetah entities in the status attribute of the cht::documentmessages::operation_status_info_set return value of the process method, protocol client MUST update the item operation map as follows:

• Use the first_op_id attribute of the cht::documentmessages::operation_status_info to find the operation_set_status structure for this collection of item operations.

• Increase num_errors attribute of the operation_set_status structure by the number of errors in the cht::documentmessages::operation_status_info Cheetah entity.

• Increase num_callbacks attributes in the operation_set_status structure by 1.

• If the num_callbacks attribute equals 3, the protocol client MUST remove this entry from the map.

• If the num_errors attribute equals the number of item operations in the submitted_operations attribute, the protocol client MUST remove the operation_set_status structure from the map.

The protocol client MUST use the item operation identifier to document identifier map to map item operation identifiers in errors and warnings in callback messages to the document identifier of item operation.

3.5.4.6 processing::session::__ping

The protocol client MUST call the processing::session::__ping method to the processing::session server object at regular intervals. The __ping method is provided by all server objects as specified in [MS-FSMW] section 2. If invoking the __ping method raises an exception, the protocol client MUST recreate the session as described in section 3.5.6.

3.5.5 Timer Events

None.

3.5.6 Other Local Events

If any of the methods in the processing::session interface, as specified in section 3.4.4 raises a system exception, as specified in [MS-FSMW], the protocol client MUST perform the following steps:

1. Recreate the feeding session by calling the protocol server method processing::session_factory::recreate, as specified in section 3.2.4.3.

2. For all item operation sequences in the item operation map, protocol client MUST resubmit the item operation sequences using the processing::session::process method, as specified in section 3.4.4.5.

3. Reset the item operation map.

4. Reset the item operation identifier to document identifier map.
3.6 coreprocessing::control Server Details

3.6.1 Abstract Data Model

See common abstract data model for more information about protocol server as specified in section 3.1.1.

3.6.2 Timers

None.

3.6.3 Initialization

The protocol server MUST use the Middleware bind method to register a coreprocessing::control server object in the name server, as specified in [MS-FSMW] section 2.

The input values for the bind method are encapsulated in an AOR, as specified in [MS-FSMW] section 2.

name: A string value that MUST be "esp/subsystems/processing dispatcher".
object_id: An integer value that MUST be unique for each server object.
host: A string that contains the host name of the server object on the protocol server. The value is implementation specific of the higher level application.
port: base port plus 390.
interface_type: A string value that MUST be "coreprocessing::control".
interface_version: A string value that MUST be "5.1".

3.6.4 Message Processing Events and Sequencing Rules

The coreprocessing::control interface specifies the methods that are listed in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>suspend_feeding</td>
<td>Suspends feeding to a given content collection.</td>
</tr>
<tr>
<td>resume_feeding</td>
<td>Enables feeding to a given content collection that has been suspended.</td>
</tr>
<tr>
<td>clear_collection</td>
<td>Clears all items in given content collection from the search index.</td>
</tr>
</tbody>
</table>

3.6.4.1 coreprocessing::control::suspend_feeding

Suspends feeding to a given content collection. All existing session objects using this content collection MUST be closed and removed from the protocol server. The method is specified by the following FSIDL specification:

```plaintext
void suspend_feeding(in string collection);
```

collection: Name of content collection to suspend feeding for.

Return value: None.
**Exceptions:** No exceptions are raised beyond those raised by the underlying Middleware Protocol, as specified in [MS-FSMW].

The protocol server MUST NOT allow any `processing::session_factory::create`, as specified in section 3.3.4.2 and `processing::session_factory::recreate`, as specified in section 3.2.4.3 to succeed for the given content collection until `coreprocessing::control::resume_feeding`, as specified in section 3.6.4.2 with the given content collection has been called.

When the protocol server receives a `suspend_feeding` method invocation from the protocol client on the `coreprocessing::control` server object, it MUST add the content collection name given in the `collection` input value to the `collection suspended holder` state.

The protocol server MUST deactivate all `processing::session` server objects in the `session holder` state where the `collection id` of the `processing::session` server object is equal to the `collection` input value.

The `collection` input value MUST be added to the `collection suspended holder`.

The protocol server MUST call the `coreprocessing::session::close` method in the indexing dispatcher, as defined in [MS-FSDP] section 3 for all `processing::session` server objects in the `session holder` state where the `collection id` of the `processing::session` server object is equal to the `collection` input value.

### 3.6.4.2 coreprocessing::control::resume_feeding

Resumes feeding to a given content collection. The method is specified by the following FSIDL specification:

```fsidl
void resume_feeding(in string collection);
```

**collection:** Name of content collection to resume feeding for.

**Return value:** None.

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

When the protocol server receives a `resume_feeding` method invocation from the protocol client on the `control` server object, it MUST remove the content collection name given in the `collection` input value from the `collection suspended holder` state.

### 3.6.4.3 coreprocessing::control::clear_collection

This method removes all items from a given content collection in the search index. The method is specified by the following FSIDL specification:

```fsidl
void clear_collection(    in string collection,    in cht::documentmessages::subsystem_id_set subsystems,    in boolean wait_for_completed) raises (coreprocessing::timed_out,    coreprocessing::unknown_collection_error,    coreprocessing::operation_failed,    core::unsupported_guarantee_set);
```
**collection**: Identifier of the content collection to remove all items from.

**subsystems**: The ids attribute of the cht::documentmessages::subsystem_id_set Cheetah entity MUST be an empty collection.

**wait_for_completed**: If true, this method blocks until the indexer nodes have given a **Completed by indexing** callback message for the remove operation of all items in the given content collection. If false, this method blocks until the indexer nodes have given a **Secured by indexing** callback message for the remove operation of all items in the given content collection.

**Return value**: None.

**Exceptions**

- **coreprocessing::timed_out**: Raised if the protocol server is unable to find an available item processor before the operation timeout is reached. For more information about operation timeout, see section 3.4.2. For more information about the protocol between the item processor and the protocol server, see [MS-FSDPD].

- **coreprocessing::unknown_collection_error**: Raised if given content collection is unknown.

- **coreprocessing::operation_failed**: Raised if the protocol server is unable to perform clear_collection method invocation.

- **core::unsupported_guarantee_set**: Raised if the protocol server is unable to create the feeding session with indexing dispatcher using the coreprocessing::session_factory::create method as specified in [MS-FSDP] section 3.

This method MUST only be called after feeding to the given collection has been suspended using the coreprocessing::control::suspend_feeding method.

The protocol server MUST raise the exception coreprocessing::unknown_collection_error if the given content collection does not exist in the collection holder state.

When the protocol server receives this method invocation from the protocol client, the protocol server MUST call the coreprocessing::session_factory::create method in the indexing dispatcher, as specified in [MS-FSDP] section 3. The collection input value to the coreprocessing::session_factory::create method MUST be equal to the collection input value. The protocol server MUST then use the coreprocessing::session client proxy returned from the coreprocessing::session_factory::create method to call the coreprocessing::session::process method, as specified in [MS-FSDP] section 3. The input parameters to the coreprocessing::session::process method MUST be as follows:

- **batch**: A cht::documentmessages::operation_set Cheetah entity, as specified in section 2.2.20, that contains one cht::documentmessages::clear_collection Cheetah entity, as specified in [MS-FSDP] section 2.

- **subsystems**: The ids attribute of the subsystems input value MUST be an empty collection.

**3.6.5 Timer Events**

None.

**3.6.6 Other Local Events**

None.
4 Protocol Examples

4.1 Sending Item Operations and Receiving Callback Messages

This example describes how to create and set up a session, feed item operations on this session, receive callback messages about the status of the items, recreate the session when a system exception is raised, as specified in [MS-FSMW], and then close the session.

Initializing the session

The `processing::session_factory` protocol server creates a server object implementing the `processing::session_factory` interface, and activates it. The `processing::session_factory` protocol client acquires a client proxy to this `processing::session_factory` interface by instantiating an AOR. This is possible because both the protocol client and protocol server have agreed a priori on the attributes of the AOR. Both protocol client and protocol server use unencrypted HTTP transport and the port base port plus 390, as specified in section 2.1.

Setting up the session

The protocol client creates the feeding session by calling the `processing::session_factory` protocol server using the `create` method.

The `processing::session_factory` protocol server receives the `create` method invocation, creates, activates, and returns a `session` client proxy, stores the `processing::session` server object in the `session holder` state.

The `processing::session_factory` protocol client stores the returned `session` client proxy in the `session client holder` state.

Using the session

The `processing::session` protocol client retrieves a `session` client proxy from the `session client holder` state, and uses the `session::process` message to send the item operations to the `processing::session` protocol server. The protocol client updates the `item operation map` state when the `process` method returns.

Receiving callbacks

The `processing::session` protocol server receives the callback messages from the item processor and indexing dispatcher and stores the callback messages in the `callback holder` state.

The `processing::session` protocol client receives the callback messages when calling `poll_callbacks` and `process` to the protocol server. The protocol client updates the `callback holder` state when receiving the callback messages.

Recreating the session and resubmitting operations

When the protocol client calls the `processing::session::process` method on the protocol server, and the method raises a system exception, the `processing::session` protocol client recreates the session by calling `recreate` to the `processing::session_factory` protocol server. The protocol client resubmits all sequences of item operations in the `callback holder` state to the `processing::session` protocol server using the `process` method.

Closing the session
The `processing::session` protocol client closes the session when the `callback holder` state is empty. The session is closed by calling the `close` method on the `processing::session_factory` protocol server.

4.1.1 Sample code

4.1.1.1 session_factory Protocol Server Initialization

```plaintext
SET session_factory_server_object_instance TO INSTANCE OF processing::session_factory SERVER OBJECT
SET session_factory_server_object_host TO "myserver.mydomain.com"
SET session_factory_server_object_port TO "13390"
SET session_factory_server_object_interface_type TO "processing::session_factory"
SET session_factory_server_object_interface_version TO "5.1"
SET session_factory_server_object_object_id TO 1
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_id
CALL middleware.activate WITH session_factory_server_object_instance
```

4.1.1.2 session_factory Protocol Client Initialization

```plaintext
SET session_factory_client_proxy_instance TO INSTANCE OF processing::session_factory SERVER OBJECT
SET session_factory_client_proxy_host TO "myserver.mydomain.com"
SET session_factory_client_proxy_port TO "13390"
SET session_factory_client_proxy_interface_type TO "processing::session_factory"
SET session_factory_client_proxy_interface_version TO "5.1"
SET session_factory_client_proxy_object_id TO 1
```

4.1.1.3 session_factory Protocol Client Message

```plaintext
SET collection TO "mycollection"
SET guarantees to cht::core::guarantee_set
CALL session_factory_client_proxy.create WITH collection AND guarantees RETURNING
    session_client_proxy
ADD session_client_proxy TO session_client_holder_state
```
4.1.1.4 session_factory Protocol Server Response

SET session_id TO session_id_generator
SET session_id_generator TO session_id_generator + 1
SET session_server_object_instance TO INSTANCE OF processing::session SERVER OBJECT
SET session_server_object_instance TO session id
RETURN session_server_object_instance

4.1.1.5 session Factory Protocol Server Response

GET session_client_proxy FROM session_client_holder_state
SET session_id_generator TO session_id_generator + 1
SET session_server_object_instance TO INSTANCE OF processing::session SERVER OBJECT
SET session_server_object_instance TO session id
RETURN session_server_object_instance

4.1.1.6 session Protocol Client process Method Invocation

SET operations TO OPERATION_SET_OBJECT_WITH_10_OPERATIONS
SET operations.completed_op_id TO 0
SET subsystem_id_set TO subsystem_id_set_object
SET subsystem_id_set.ids to EMPTY COLLECTION
CALL session_client_proxy.process WITH operations AND subsystem_id_set RETURN operation_status_info_set
SET retval TO RETURNED operation_status_info_set
SET first_op_id = retval[0].first_op_id
SET item_operation_callback_holder TO item_operation_callback_holder_object
SET item_operation_callback_holder.submitted_operations TO operations
SET item_operation_callback_holder.num_callbacks TO 0
SET item_operation_callback_holder.num_errors TO 0
SET item_operation_map[first_op_id] = item_operation_callback_holder
FOR i = 1, i < retval.status.size; i++
    SET key = retval[i].first_op_id
    SET item_operation_map[key].num_callbacks = item_operation_map[key].num_callbacks + 1
    SET item_operation_map[key].num_errors = item_operation_map[key].num_errors + retval[i].errors.count
IF item_operation_map[key].num_callback = 3 OR
item_operation_map[key].num_errors =
item_operation_map[key].submitted_operations.count THEN
 item_operation_map.remove(key)

4.1.1.7 session Protocol Server process Method Response

RECEIVE input values batch and subsystems

SET first_op_id = operation_id_generator

FOR i = 0, i < batch.operations.size, i++
 batch.operations[i].id = operation_id_generator
 operation_id_generator = operation_id_generator + 1

SEND operation to item processor for processing

SET retval TO operation_status_info_set

SET opstatus TO operation_status_info

SET opstatus.first_op_id TO first_op_id

SET opstatus.last_op_id TO -1

SET opstatus.stat TO cht::documentmessages::completed

SET opstatus.subsystem TO "firstopid"

ADD opstatus to retval.status

ADD callback_holder to retval.status

CLEAR callback_holder

RETURN retval

4.1.1.8 session Protocol Client poll_callbacks Method Invocation

CALL session_client_proxy.poll_callbacks RETURN operation_status_info_set

SET retval TO RETURNED operation_status_info_set

FOR i = 0, i < retval.status.size; i++

SET key = retval.status[i].first_op_id
SET item_operation_map[key].num_callbacks = item_operation_map[key] + 1
SET item_operation_map[key].num_errors = item_operation_map[key].num_errors
 + retval.status[i].errors.count
IF item_operation_map[key].num_callback = 3 OR
item_operation_map[key].num_errors =
item_operation_map[key].submitted_operations.count THEN
 item_operation_map.remove(key)
4.1.1.9 session Protocol Server poll_callbacks Method Response

SET retval TO operation_status_info_set
ADD callback_holder to retval.status
CLEAR callback_holder

4.1.1.10 session Protocol Server poll_callbacks Method Invocation with System Exception

CALL session_client_proxy.poll_callbacks RAISE system_exception
SET collection TO "mycollection"
SET guarantees to cht::core::guarantee_set
CALL session_factory_client_proxy.recreate WITH session_id AND collection AND guarantees RETURNING session_client_proxy
ADD session_client_proxy TO session_client_holder_state
SET old_item_operation_map = item_operation_map
CALL item_operation_map.reset
FOR i = 0, i < old_item_operation_map.size, i++
    SET operations TO old_item_operation_map[i].operations
    SET operations.completed_op_id TO 0
    SET subsystem_id_set TO subsystem_id_set_object
    SET subsystem_id_set.ids to EMPTY COLLECTION
    CALL session_client_proxy.process WITH operations AND subsystem_id_set
RETURN operation_status_info_set
SET retval TO RETURNED operation_status_info_set
SET first_op_id = retval[0].first_op_id
SET item_operation_callback_holder TO item_operation_callback_holder_object
SET item_operation_callback_holder.submitted_operations TO operations
SET item_operation_callback_holder.num_operations TO 0
SET item_operation_callback_holder.num_errors TO 0
SET item_operation_map[first_op_id] = item_operation_callback_holder
FOR j = 1, j < retval.status.size, i++
    SET key = retval.status[j].first_op_id
    SET item_operation_map[key].num_callbacks = item_operation_map[key].num_callbacks + 1
    SET item_operation_map[key].num_errors = item_operation_map[key].num_errors + retval.status[j].errors.count
    IF item_operation_map[key].num_callbacks = 3 OR item_operation_map[key].num_errors = item_operation_map[key].submitted_operations.count THEN
        item_operation_map.remove(key)

4.1.1.11 session_factory Protocol Client Close

CALL session_factory_client_proxy.close WITH session_id
4.1.1.12  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
5  Security

5.1  Security Considerations for Implementers

Section 2.1 specifies how to secure the transport between protocol client and protocol server.

5.2  Index of Security Parameters

None.
Appendix A: Full FSIDL

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

```idl
module interfaces
{
  module core
  {
    exception unsupported_guarantee_set
    {
      string what;
    },
  },
  module coreprocessing
  {
    exception unknown_collection
    {
      string what;
    },
    exception timed_out
    {
      long id;
      string message;
    },
    exception service_unavailable
    {
      long id;
      string message;
    },
    exception format_error
    {
      long id;
      string message;
    },
    exception no_resources
    {
      long id;
      string message;
    },
  }
},
module processing
{
  interface session_factory
  {
    #pragma version session_factory 5.1
    boolean is_master();
    session create(
      in /*collection_id*/ string collection,
      in cht::core::guarantee_set guarantees)
    raises (coreprocessing::unknown_collection_error,
      core::unsupported_guarantee_set);
    session recreate(in long id,
      in /*collection_id*/ string collection,
      in cht::core::guarantee_set guarantees)
    raises (coreprocessing::unknown_collection_error,
      core::unsupported_guarantee_set);
    void close(in /*session_id*/ long id);
  };
```
interface session
{
  #pragma version session 5.1
  long get_session_id();
  long get_operation_timeout();
  cht::documentmessages::operation_status_info_set
  process(in cht::documentmessages::operation_set batch,
          in cht::documentmessages::subsystem_id_set subsystems)
  raises (coreprocessing::timed_out,
          coreprocessing::service_unavailable,
          coreprocessing::format_error, coreprocessing::no_resources);
  cht::documentmessages::subsystem_id_set get_system_ids();
  cht::documentmessages::operation_status_info_set poll_callbacks();
};
};
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 3.3.1 38, section 3.5.1 45)
server (section 3.1.1 32, section 3.2.1 34,
section 3.4.1 39, section 3.6.1 48)
Applicability 10

C
Capability negotiation 10
Change tracking 61
cht

core
feeding_priority data type 13
guarantee data type 13
guarantee_set data type 12
documentmessages

action data type 13
bool_attribute data type 20
bool_collection data type 22
bytearray_attribute data type 21
bytearray_collection data type 23
document data type 23
document_id data type 20
error data type 14
float_attribute data type 21
float_collection data type 22
format_error data type 16
integer_attribute data type 21
integer_collection data type 22
invalid_content data type 16
key_value_collection data type 19
key_value_pair data type 19
long_attribute data type 21
long_collection data type 23
operation data type 18
operation_dropped data type 16
operation_lost data type 16
operation_set data type 18
operation_state data type 13
operation_status_info data type 18
operation_status_info_set data type 19
partial_update_operation data type 28
processing_error data type 14
remove_operation data type 28
resource_error data type 17
server_unavailable data type 15
string_attribute data type 20
string_collection data type 22
subsystem_id_set data type 29
unknown_document data type 17
update_operation data type 28
urlschange_operation data type 29
utf8_error data type 15

warning data type 17
xml_error data type 15

Client
abstract data model (section 3.3.1 38, section 3.5.1 45)
initialization (section 3.3.3 38, section 3.5.3 45)
local events (section 3.3.6 39, section 3.5.6 47)
message processing (section 3.3.4 39, section 3.5.4 45)
processing::session::ping method 47
processing::session::get_operation_timeout method 46
processing::session::get_session_id method 46
processing::session::get_system_ids method 46
processing::session::poll_callbacks method 46
processing::session::process method 46
processing::session_factory::close method 39
processing::session_factory::create method 39
processing::session_factory::is_master method 39
processing::session_factory::recreate method 39
sequencing rules (section 3.3.4 39, section 3.5.4 45)
timer events (section 3.3.5 39, section 3.5.5 47)
timers (section 3.3.2 38, section 3.5.2 45)
Common data types 11
common interface 32
core
unsupported_guarantee_set data type 29

coreprocessing
format_error data type 30
no_resources data type 30
operation_failed data type 31
service_unavailable data type 30
timed_out data type 29
unknown_collection_error data type 30
coreprocessing::control::clear_collection method 49
coreprocessing::control::resume_feeding method 49
coreprocessing::control::suspend_feeding method 48

D
Data model - abstract
client (section 3.3.1 38, section 3.5.1 45)
server (section 3.1.1 32, section 3.2.1 34,
section 3.4.1 39, section 3.6.1 48)

Data types
cht
core
feeding_priority 13
guarantee 13
guarantee_set 12
documentmessages
action 13
bool_attribute 20
bool_collection 22
bytearray_attribute 21
bytearray_collection 23
document 23
document_id 20
error 14
float_attribute 21
float_collection 22
format_error 15
indexing_error 16
integer_attribute 21
integer_collection 22
invalid_content 16
key_value_collection 19
key_value_pair 19
long_attribute 21
long_collection 22
operation 18
operation_dropped 16
operation_lost 16
operation_set 18
operation_state 13
operation_status_info 19
operation_status_info_set 19
partial_update_operation 28
processing_error 14
remove_operation 28
resource_error 17
string_attribute 20
string_collection 22
subsystem_id_set 19
unknown_document 17
update_operation 28
urlschange_operation 29
utf8_error 15
warning 17
xml_error 15
common - overview 11
core
unsupported_guarantee_set 29
coreprocessing
format_error 30
no_resources 30
operation_failed 31
service_unavailable 30
timed_out 29
unknown_collection_error 30
desession_protocol_client_initialization_example 53
desession_protocol_client_process_method_invocation_example 53
E
Events
local - client (section 3.3.6 39, section 3.5.6 47)

F
Fields - vendor-extendable 10
FSIDL 58
Full FSIDL 58

G
Glossary 7

I
Implementer - security considerations 57
Index of security parameters 57
Informative references 8
Initialization
client (section 3.3.3 38, section 3.5.3 45)
server (section 3.1.3 33, section 3.2.3 34, section 3.4.3 40, section 3.6.3 48)
Interfaces - server
core 32
Introduction 7

L
Local events
client (section 3.3.6 39, section 3.5.6 47)
server (section 3.1.6 34, section 3.2.6 38, section 3.4.6 45, section 3.6.6 50)

M
Message processing
client (section 3.3.4 39, section 3.5.4 45)
server (section 3.1.4 33, section 3.2.4 34, section 3.4.4 40, section 3.6.4 48)
Messages

core

feeding_priority data type 13
guarantee data type 13
guarantee_set data type 12
documentmessages

action data type 13
bool_attribute data type 20
bool_collection data type 22
bytearray_attribute data type 21
bytearray_collection data type 23
document data type 23
document_id data type 20
error data type 14
float_attribute data type 21
float_collection data type 22
format_error data type 15
indexing_error data type 16
integer_attribute data type 21
integer_collection data type 22
invalid_content data type 16
key_value_collection data type 19
key_value_pair data type 19
long_attribute data type 21
long_collection data type 23
operation data type 18
operation_dropped data type 16
operation_lost data type 16
operation_set data type 18
operation_state data type 13
operation_status_info data type 18
operation_status_info_set data type 19
partial_update_operation data type 28
processing_error data type 14
remove_operation data type 28
resource_error data type 28
server_unavailable data type 15
string_attribute data type 20
string_collection data type 22
subsystem_id_set data type 29
unknown_document data type 17
update_operation data type 28
updatechange_operation data type 29
utf8_error data type 15
warning data type 17
xml_error data type 15

common data types 11

core

unsupported_guarantee_set data type 29
coreprocessing

format_error data type 30
no_resources data type 30
operation_failed data type 31
service_unavailable data type 30
timed_out data type 29

unknown_collection_error data type 30

transport 11

Methods

coreprocessing::control::clear_collection 49
coreprocessing::control::resume_feeding 49
coreprocessing::control::suspend_feeding 48
processing::session::__ping 47
processing::session::get_operation_timeout (section 3.4.4.1 40, section 3.5.4.1 46)
processing::session::get_session_id (section 3.4.4.2 41, section 3.5.4.2 46)
processing::session::get_system_ids (section 3.4.4.3 41, section 3.5.4.3 46)
processing::session::poll_callbacks (section 3.4.4.4 41, section 3.5.4.4 46)
processing::session::process (section 3.4.4.5 43, section 3.5.4.5 46)
processing::session_factory::close (section 3.2.4.1 37, section 3.3.4.1 39)
processing::session_factory::create (section 3.2.4.2 35, section 3.3.4.2 39)
processing::session_factory::is_master (section 3.2.4.1 35, section 3.3.4.1 39)
processing::session_factory::recreate (section 3.2.4.3 36, section 3.3.4.3 39)

N

Normative references 8

O

Overview - protocol details 32
Overview (synopsis) 8

P

Parameters - security index 57
Preconditions 10
Prerequisites 10
processing::session::__ping method 47
processing::session::get_operation_timeout method (section 3.4.4.1 40, section 3.5.4.1 46)
processing::session::get_session_id method (section 3.4.4.2 41, section 3.5.4.2 46)
processing::session::get_system_ids method (section 3.4.4.3 41, section 3.5.4.3 46)
processing::session::poll_callbacks method (section 3.4.4.4 41, section 3.5.4.4 46)
processing::session::process method (section 3.4.4.5 43, section 3.5.4.5 46)
processing::session_factory::close method (section 3.2.4.1 37, section 3.3.4.1 39)
processing::session_factory::create method (section 3.2.4.2 35, section 3.3.4.2 39)
processing::session_factory::is_master method (section 3.2.4.1 35, section 3.3.4.1 39)
processing::session_factory::recreate method (section 3.2.4.3 36, section 3.3.4.3 39)
Product behavior 60
Protocol details - overview 32

[MS-FSCF] — v20120630
Content Feeding Protocol Specification

Copyright © 2012 Microsoft Corporation.

Release: July 16, 2012