[MS-FSDPD]:
Document Processing Distribution Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	Major	Updated and revised the technical content
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Document Processing Distribution Protocol, which is used to distribute items between various item processing related components in a search service application.

1.1 Glossary

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

attribute
fully qualified domain name (FQDN)
UTF-8
XML

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

abstract object reference (AOR)
base port
callback message
Cheetah
Cheetah checksum
Cheetah entity
client proxy
content client
content distributor
document identifier
FAST Search Interface Definition Language (FSIDL)
host name
indexing dispatcher
indexing node
item
item processing
name server
search index

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-FSCF] Microsoft Corporation, "Content Feeding Protocol Specification"

[MS-FSCHT] Microsoft Corporation, "Cheetah Data Structure"

[MS-FSCMW] Microsoft Corporation, "Configuration Middleware Protocol Specification"

[MS-FSCX] Microsoft Corporation, "Configuration (XML-RPC) Protocol Specification"

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

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

[MS-FSNC] Microsoft Corporation, "Node Controller Protocol Specification"

[MS-FSPSCFG] Microsoft Corporation, "Processor Server Configuration File Format Specification"

[MS-FSSCFG] Microsoft Corporation, "Search Configuration File Format Specification"

[MS-FSST] Microsoft Corporation, "Spelltuning File Format Specification"

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[XML-RPC] Winer, D., "XML-RPC Specification", June 1999, http://www.xmlrpc.com/spec

1.2.2 Informative References

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

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

1.3 Protocol Overview (Synopsis)

The Document Processing Distribution Protocol enables communication between a **content distributor** and an **item processing** component in a search service application. Specifically, this protocol enables an item processing component to register with a content distributor, and it enables a content distributor to dispatch **item** operations to an item processing component for item processing.

The content distributor and item processing component belong to a larger, session-based item feeding chain. A **content client** at one end of the chain sends information to an indexing service at the other end of the chain. This information is about operations to be performed on items. The indexing service adds, updates, and removes items. The content client receives a **callback message** from the content distributor when the item operations have been processed, when they have been stored to disk, and when they have been indexed. The feeding chain sequence consists of a content client, a content distributor, an item processing component, an **indexing dispatcher**, and **indexing nodes**, as shown in the following diagram.

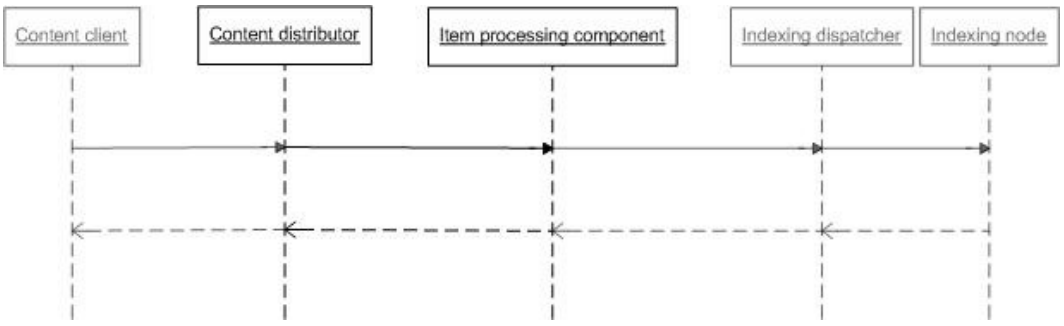

Figure 1: Item feeding chain

In the feeding chain, one or more item processing components register with the content distributor by using two client/server interfaces: one as described in both processing::master_dispatcher Client Details (section 3.4) and processing::master_dispatcher Server Details (section 3.3), and the other as described in both processing::procserver_handler Client Details (section 3.6) and processing::procserver_handler Server Details (section 3.5). After it registers, an item processing component is ready for item processing. Examples of item processing include format detection and conversion, linguistic analysis, and link analysis.

The role of the content distributor is to dispatch content from one or more content clients to the available item processing components. It does so by using a client/server interface as described in both processing::processor_server Client Details (section 3.2) and processing::processor_server Server Details (section 3.1). The content exists in the form of item operations, such as those for adding, updating, and removing items.

The item processing component sends the item operations to the indexing dispatcher for indexing. The item processing component sends a callback message to the content distributor when the item operations have been processed. The callback messages are communicated by means of the client/server interface as described in both processing::procserver_handler Client Details (section 3.6) and processing::procserver_handler Server Details (section 3.5).

The content distributor monitors the item processing components that are registered. If an item processing component is unresponsive, the content distributor contacts the node controller, as described in [MS-FSNC], to terminate that item processing component, as described in Timer Events (section 3.5.5).

The content distributor and the item processing components communicate with the configuration component to obtain configuration data by using a server interface, as described in [MS-FSCMW] section 3, and a configuration component XML-RPC interface, as described in [MS-FSCX] section 3.

Each item processing component also implements a status interface, as described in Status Server Details (section 3.7), that other components query for item processing logs and statistics.

1.4 Relationship to Other Protocols

This protocol uses the Middleware Protocol, as described in [MS-FSMW]. Custom data types are serialized by means of the Cheetah Data Format, as described in [MS-FSCHT].

This protocol also uses XML-RPC by using HTTP for transporting messages in **XML**, as described in [XML-RPC].

This protocol uses HTTP, as described in [RFC2616], over TCP/IP as the transport mechanism. The following diagram shows the underlying messaging and transport stack that the protocol uses:

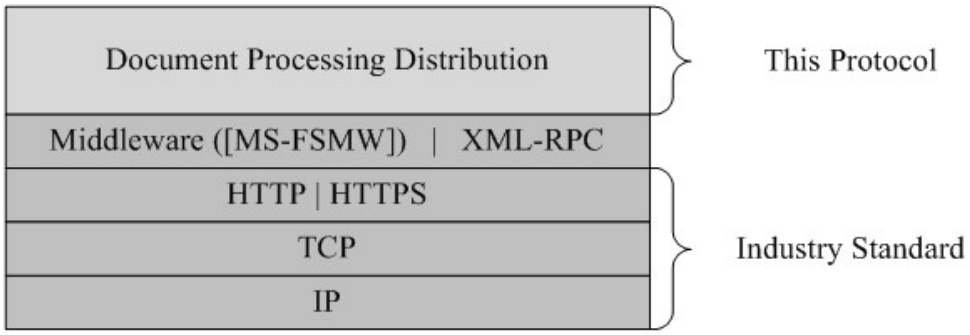

Figure 2: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

It is assumed that a protocol client and a protocol server using the Middleware Protocol [MS-FSMW] as the transport mechanism both have the location and connection information for the shared **name server**.

A protocol client using XML-RPC as the transport mechanism needs to be able to establish a connection to the protocol server over TCP/IP. It is assumed that the protocol client knows the **host name** and port that are associated with the protocol server prior to initiating the connection.

This document explains the use of one content distributor; it does not cover the use of multiple content distributors.

1.6 Applicability Statement

This protocol enables a content distributor to dispatch items to one or more item processing components. This protocol also enables the item processing components to use callback messages to send status information regarding the processing and indexing of the submitted items back to the content distributor.

This protocol is part of a feeding chain that contains an item-feeding content client at one end and an indexing node at the other. The feeding chain components have the ability to send status information about items back through the chain by using asynchronous callback messages.

1.7 Versioning and Capability Negotiation

Regarding capability negotiation, although the Middleware Protocol [MS-FSMW] is connectionless, every message that is sent via the Middleware Protocol needs to specify the correct interface version. For more information, see Initialization (section 3.3.3), Initialization (section 3.5.3), and Common Data Types (section 2.2).

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The content distributor MUST use the Middleware Protocol, as specified in [MS-FSMW], to send messages to the item processing component, configuration server, and indexing dispatcher.

Messages from the content distributor to the node controller, from the item processing component to the configuration component, and from a protocol client to a protocol server implementing the **Status** interface MUST be sent via XML-RPC, as specified in [XML-RPC]. The protocol clients and protocol servers that use XML-RPC MUST use HTTP, as specified in [RFC2616], over TCP/IP as the transport mechanism. The XML body requests and responses MUST be formatted as specified in [XML-RPC]. The HTTP GET path, as specified in [RFC2616], MUST be "/RPC2".

2.2 Common Data Types

The messages for this protocol are specified by using **FAST Search Interface Definition Language (FSIDL)**. The allowed FSIDL data types, as specified in [MS-FSMW], are encoded as specified in [MS-FSMW] section 2. **Cheetah entities** are encoded as specified in [MS-FSCHT] section 2. The **Cheetah checksum** and **Cheetah** type identifier for each Cheetah entity MUST both be integers, as specified in the following table.

Cheetah entity	Cheetah type identifier	Cheetah checksum
cht::documentmessages::key_value_pair	0	211918678
cht::documentmessages::key_value_collection	1	211918678
cht::documentmessages::bool_attribute	2	211918678
cht::documentmessages::bool_collection	3	211918678
cht::documentmessages::bytearray_attribute	4	211918678
cht::documentmessages::bytearray_collection	5	211918678
cht::documentmessages::warning	6	211918678
cht::documentmessages::operation	7	211918678
cht::documentmessages::document_id	11	211918678
cht::documentmessages::document	12	211918678
cht::documentmessages::error	15	211918678
cht::documentmessages::failed_operation	18	211918678
cht::documentmessages::float_attribute	19	211918678
cht::documentmessages::float_collection	20	211918678
cht::documentmessages::processing_error	21	211918678
cht::documentmessages::format_error	22	211918678
cht::documentmessages::string_attribute	26	211918678

Cheetah entity	Cheetah type identifier	Cheetah checksum
cht::documentmessages::integer_attribute	28	211918678
cht::documentmessages::integer_collection	30	211918678
cht::documentmessages::long_attribute	34	211918678
cht::documentmessages::long_collection	35	211918678
cht::documentmessages::operation_dropped	36	211918678
cht::documentmessages::operation_lost	37	211918678
cht::documentmessages::operation_set	38	211918678
cht::documentmessages::subsystem_id_set	39	211918678
cht::documentmessages::operation_status_info	40	211918678
cht::documentmessages::partial_update_operation	42	211918678
cht::documentmessages::remove_operation	45	211918678
cht::documentmessages::server_unavailable	47	211918678
cht::documentmessages::string_collection	49	211918678
cht::documentmessages::update_operation	52	211918678
cht::documentmessages::urlschange_operation	53	211918678
cht::documentmessages::utf8_error	54	211918678
cht::documentmessages::xml_error	55	211918678

For the messages sent via XML-RPC, the **int**, **double**, **string**, **data**, **array**, and **struct** data types MUST be used as specified in [XML-RPC]. In addition, this protocol uses the data structures that are specified in the following table.

Data structure name	Description
ProcessingStatistics	The processing statistics for a single item processing stage or item processing pipeline. For more details, see ProcessingStatistics (section 2.2.39).
ModuleStatus	Information about the status of the protocol server. For more details, see ModuleStatus (section 2.2.40).
Statistics	The processing statistics for the protocol server. For more details, see Statistics (section 2.2.41).
LogMessage	Information about a single log event. For more details, see LogMessage (section 2.2.42).
ItemLog	Information about all the log events for either a single item operation or a single sequence of item operations. For more details, see ItemLog (section 2.2.43).
ItemStatusLog	Log information for either several item operations or several sequences of item

Data structure name	Description
	operations. For more details, see ItemStatusLog (section 2.2.44).

2.2.1 cht::documentmessages::action

The **action** Cheetah enumeration specifies actions that are used in error messages.

```
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.

2.2.2 cht::documentmessages::warning

The **warning** Cheetah entity contains warning information about a specific item operation.

```
entity 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 processing component, 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.

operation_id: A long integer that uniquely identifies the item operation.

2.2.3 cht::documentmessages::error

The **error** Cheetah entity contains error information for a specific item operation.

```
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 cht::documentmessages::action (section 2.2.1), containing the RECOMMENDED 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 processing component, 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 consist of an empty Cheetah collection.

2.2.4 cht::documentmessages::processing_error

The **processing_error** Cheetah entity specifies when an error occurred during the processing of an item operation.

The **processing_error** Cheetah entity is a subclass of the **error** Cheetah entity, which is specified in cht::documentmessages::error (section 2.2.3). The **processing_error** Cheetah entity is also a common superclass for:

- The **format_error** Cheetah entity, which is specified in section cht::documentmessages::format_error (section 2.2.5).
- The **server_unavailable** Cheetah entity, which is specified in section cht::documentmessages::server_unavailable (section 2.2.8).
- The **operation_dropped** Cheetah entity, which is specified in section cht::documentmessages::operation_dropped (section 2.2.9).

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

processor: A string that specifies the name of the item processing stage during which the error occurred.

2.2.5 cht::documentmessages::format_error

The **format_error** Cheetah entity is used to indicate that an item operation has an invalid format.

The **format_error** Cheetah entity is a subclass of the **processing_error** Cheetah entity, which is specified in cht::documentmessages::processing_error (section 2.2.4). The **format_error** Cheetah entity is also a common superclass for the **xml_error** Cheetah entity, which is specified in cht::documentmessages::xml_error (section 2.2.6), and the **utf8_error** Cheetah entity, which is specified in cht::documentmessages::utf8_error (section 2.2.7).

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

2.2.6 cht::documentmessages::xml_error

The **xml_error** Cheetah entity is used to indicate that an item operation contains invalid XML code.

The **xml_error** Cheetah entity is a subclass of the **format_error** Cheetah entity, which is specified in cht::documentmessages::format_error (section 2.2.5).

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

2.2.7 cht::documentmessages::utf8_error

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

The **utf8_error** Cheetah entity is a subclass of the **format_error** Cheetah entity, which is specified in cht::documentmessages::format_error (section 2.2.5).

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

2.2.8 cht::documentmessages::server_unavailable

The **server_unavailable** Cheetah entity is used to indicate that a protocol client was 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, which is specified in cht::documentmessages::processing_error (section 2.2.4).

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

2.2.9 cht::documentmessages::operation_dropped

The **operation_dropped** Cheetah entity is used to indicate that the item processing component has identified an item operation that MUST NOT be indexed.

The **operation_dropped** Cheetah entity is a subclass of the **processing_error** Cheetah entity, which is specified in cht::documentmessages::processing_error (section 2.2.4).

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

2.2.10 cht::documentmessages::operation_lost

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

The **operation_lost** Cheetah entity is a subclass of the **error** Cheetah entity, which is specified in cht::documentmessages::error (section 2.2.3).

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

2.2.11 cht::documentmessages::operation_set

The **operation_set** Cheetah entity contains a set of **operation** objects, which are specified in cht::documentmessages::operation (section 2.2.12).

```
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.

2.2.12 cht::documentmessages::operation

The **operation** Cheetah entity is a common super class for the following Cheetah entities:

- The **update_operation** Cheetah entity, which is specified in cht::documentmessages::update_operation (section 2.2.29).
- The **partial_update_operation** Cheetah entity, which is specified in cht::documentmessages::partial_update_operation (section 2.2.31).
- The **remove_operation** Cheetah entity, which is specified in cht::documentmessages::remove_operation (section 2.2.30).
- The **urlschange_operation** Cheetah entity, which is specified in cht::documentmessages::urlschange_operation (section section 2.2.32)

```
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 cht::documentmessages::warning (section 2.2.2). This collection contains all the warnings for the item operation that is identified by the **id attribute (1)**.

2.2.13 cht::documentmessages::document

The **document** Cheetah entity contains information about a single item.

```
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 2.2.15), that uniquely identifies the item.

document_attributes: A collection of **key_value_pair** Cheetah entities, as specified in cht::documentmessages::key_value_pair (section 2.2.28), that contains the attributes (1) of the item.

2.2.14 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 cht::documentmessages::key_value_pair (section 2.2.28).

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

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

2.2.15 cht::documentmessages::document_id

The **document_id** Cheetah entity uniquely identifies an item by representing the **document identifier (3)** of the item.

```
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.

2.2.16 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 subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

value: A string that contains the value.

2.2.17 cht::documentmessages::bool_attribute

The **bool_attribute** Cheetah entity forms an association between a key and a Boolean value**.**

The **bool_attribute** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

value: A Boolean attribute that contains the value.

2.2.18 cht::documentmessages::float_attribute

The **float_attribute** Cheetah entity forms an association between a key and a floating point value.

The **float_attribute** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

value: A floating point attribute that contains the value.

2.2.19 cht::documentmessages::integer_attribute

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

The **integer_attribute** Cheetah entity is a subclass of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

value: An integer that contains the value.

2.2.20 cht::documentmessages::long_attribute

The **long_attribute** Cheetah entity forms an association between a key and a long integer value.

The **long_attribute** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

value: A long integer that contains the value.

2.2.21 cht::documentmessages::bytearray_attribute

The **bytearray_attribute** Cheetah entity forms an association between a key and a value that is contained in a byte array.

The **bytearray_attribute** Cheetah entity is a subclass of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

value: A byte array that contains the value.

2.2.22 cht::documentmessages::string_collection

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

The **string_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

```
entity string_collection : key_value_pair {
    collection string values;
};
```

values: A string collection that contains the values.

2.2.23 cht::documentmessages::bool_collection

The **bool_collection** Cheetah entity forms an association between a key and a collection of Boolean values.

The **bool_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

```
entity bool_collection : key_value_pair {
    collection bool values;
};
```

values: A collection that contains the Boolean values.

2.2.24 cht::documentmessages::float_collection

The **float_collection** Cheetah entity forms an association between a key and a collection of floating point values.

The **float_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

```
entity float_collection : key_value_pair {
    collection float values;
};
```

values: A collection that contains the floating point values.

2.2.25 cht::documentmessages::integer_collection

The **integer_collection** Cheetah entity forms an association between a key and a collection of integer values.

The **integer_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

```
entity integer_collection : key_value_pair {
    collection integer values;
};
```

values: A collection that contains the integer values.

2.2.26 cht::documentmessages::long_collection

The **long_collection** Cheetah entity forms an association between a key and a collection of long integer values.

The **long_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

values: A collection that contains the long integer values.

2.2.27 cht::documentmessages::bytearray_collection

The **bytearray_collection** Cheetah entity forms an association between a key and a collection of byte arrays.

The **bytearray_collection** Cheetah entity is a subtype of the **key_value_pair** Cheetah entity, which is specified in cht::documentmessages::key_value_pair (section 2.2.28).

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

```
};
```

values: A collection that contains the **bytearray** values.

2.2.28 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. The **key_value_pair** Cheetah entity represents an attribute (1).

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

The **key_value_pair** Cheetah entity has the following attribute:

key: A string that contains the key.

2.2.29 cht::documentmessages::update_operation

The **update_operation** Cheetah entity either adds a specific item to the index or replaces that item. If an item with the specified document identifier (3) already exists in the **search index**, it is replaced.

The **update_operation** Cheetah entity is a subclass of the **operation** Cheetah entity, which is specified in cht::documentmessages::operation (section 2.2.12).

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

doc: A **document** Cheetah entity, as specified in cht::documentmessages::document (section 2.2.13), that represents the item to add or replace.

2.2.30 cht::documentmessages::remove_operation

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

The **remove_operation** Cheetah entity is a subclass of the **operation** Cheetah entity, which is specified in cht::documentmessages::operation (section 2.2.12).

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

doc_id: A **document_id** Cheetah entity, as specified in cht::documentmessages::document_id (section 2.2.15), that uniquely identifies the item.

2.2.31 cht::documentmessages::partial_update_operation

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

The **partial_update_operation** Cheetah entity is a subtype of the **operation** Cheetah entity, which is specified in cht::documentmessages::operation (section 2.2.12).

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

doc: A **document** Cheetah entity, as specified in cht::documentmessages::document (section 2.2.13), that contains the attributes (1) to update.

2.2.32 cht::documentmessages::urlschange_operation

The urlschange_operation Cheetah entity updates one or more attributes for a specific item in the search index.

The **urlschange_operation** Cheetah entity is a subtype of the **partial_update_operation** Cheetah entity, which is specified in cht::documentmessages::operation (section 2.2.12).

```
entity urlschange_operation : partial_update_operation {
};
```

2.2.33 cht::documentmessages::failed_operation

The **failed_operation** Cheetah entity is used to notify the indexing dispatcher that an operation has failed.

The **failed_operation** Cheetah entity is a subtype of the **operation** Cheetah entity, which is specified in cht::documentmessages::operation (section 2.2.12).

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

subsystem: A string that describes where the operation failed. This string MUST have the value "processing".

state: An **operation_state** Cheetah entity, as specified in cht::documentmessages::operation_state (section 2.2.36), that contains the state of the operation.

operation_type: A string that describes the type of operation. This string MUST have the value "failed_operation".

doc_id: A **document_id**, as specified in cht::documentmessages::document_id (section 2.2.15), that uniquely identifies the item.

err: An **error** Cheetah entity, as specified in cht::documentmessages::error (section 2.2.3), that contains information about the error that caused the operation to fail.

2.2.34 cht::documentmessages::subsystem_id_set

The **subsystem_id_set** Cheetah entity contains a collection of names.

```
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".

2.2.35 cht::documentmessages::operation_status_info

The **operation_status_info** Cheetah entity, which contains status information about a set of operations, is used to report the status of submitted item operations to the protocol client.

```
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.

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.

state: An **operation_state** enumeration constant, as specified in cht::documentmessages::operation_state (section 2.2.36), 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 processing component, 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".

errors: A collection of **error** Cheetah entities, which are specified in cht::documentmessages::error (section 2.2.3). 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 cht::documentmessages::warning (section 2.2.2). This value contains the warnings for the set of item operations that is specified by the **first_op_id** and **last_op_id** attributes.

2.2.36 cht::documentmessages::operation_state

The **operation_state** Cheetah enumeration specifies the possible states of an item operation.

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

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.

2.2.37 processing::shutting_down

The **processing::shutting_down** exception is raised by the protocol server if the protocol server is in the process of shutting down when a protocol client calls the **process** method, as specified in processing::processor_server::process (section 3.1.4.1).

```
exception shutting_down {
};
```

2.2.38 processing::system_resource_error

The **system_resource_error** exception is raised by the protocol server if the protocol server receives an exception during processing that is related to a resource constraint. An example of such a constraint is not enough memory being available.

```
exception system_resource_error {
    string what;
};
```

what: A description of the resource constraint that occurred.

2.2.39 ProcessingStatistics

The **ProcessingStatistics** structure contains statistics for a running item processing component or item processing pipeline. The members of this structure are described in the following table.

Member name	Type	Description
OK	**int**	The number of items that have been successfully processed.
ERROR	**int**	The number of items that have been processed with an error.
WorkTime	**double**	The amount of time, in seconds, that has been spent on processing items.
UserTime	**double**	The amount of time, in seconds, that has been spent in user mode.

Member name	Type	Description
SystemTime	**double**	The amount of time, in seconds, that has been spent in system mode.
PageSwaps	**int**	The number of page faults requiring input/output that have occurred.
VirtualMem	**int**	The amount of virtual memory, in bytes, that has been allocated.
ResidentMem	**int**	The amount of resident memory, in bytes, that has been allocated.
MemUsage	**int**	The amount of memory, in bytes, that has been allocated.

2.2.40 ModuleStatus

The **ModuleStatus** structure contains the status of the protocol server. The members of this structure are described in the following table.

Member name	Type	Description
Started	**int**	The time when the protocol server started. This time is expressed in number of seconds since January 1, 1970.
Idletime	**int**	The number of seconds that the protocol server has been idle—that is, has not been processing items.
Uptime	**int**	The number of seconds that has elapsed since the protocol server started.
CurrentWork	**int**	Whether the protocol server is currently processing items. The value MUST be 0 if the protocol server is not processing items, and the value MUST be 1 if the protocol server is processing items.
Verbosity	**int**	The verbosity level, which controls how much logging will be performed. The value MUST be one of the following: ▪ 1 for no statistics or item logging ▪ 2 for statistics and document logging ▪ 3 for statistics, document logging, and the logging of attribute changes to items

2.2.41 Statistics

The **Statistics** structure is used to log item processing statistics. The members of this structure are described in the following table.

Member name	Type	Description
Elapsed	**int**	The amount of time, in seconds, that has elapsed since the last reset of the state.
Statistics	**array**	An array that MUST have two elements. The first element is a map containing the names of the item processing stages and their associated **ProcessingStatistics** structures, as specified in ProcessingStatistics (section 2.2.39). The second element is a map containing the names of the item processing pipelines and their associated **ProcessingStatistics** structures, as specified in ProcessingStatistics

Member name	Type	Description
		(section 2.2.39).

2.2.42 LogMessage

The **LogMessage** array is used to log one event in a log. This array contains two elements, as described in the following table.

Element type	Description
string	The verbosity level of the log message. The value MUST be one of the following values: "INFO", "WARNING", or "ERROR".
string	The log message.

2.2.43 ItemLog

The **ItemLog** structure contains log messages for a single item or a sequence of item operations. The members of this structure are described in the following table.

Member name	Type	Description
Status	**string**	The status of the item or sequence of item operations. The value MUST be one of the following: ▪ "OK" if the item or sequence of item operations has been processed without errors or warnings. ▪ "WARNING" if the item or the sequence of item operations has been processed with one or more warnings. ▪ "ERROR" if the item or sequence of item operations has been processed with one or more errors.
Modified	**int**	The time when the item or sequence of item operations was last modified. This time is expressed in number of seconds since January 1, 1970.
Msgs	**array**	An array of **LogMessage** arrays, as specified in LogMessage (section 2.2.42).
Elapsed	**int**	The number of seconds that the protocol server spent on processing the item or sequence of item operations.

2.2.44 ItemStatusLog

The **ItemStatusLog** array contains zero or more structures of type **ItemLog**, as specified in ItemLog (section 2.2.43), that are associated with an item identifier variable in the log entry.

3 Protocol Details

This protocol consists of the following interfaces:

- **processing::processor_server**
- **processing::master_dispatcher**
- **processing::procserver_handler**
- **Status**

To use this protocol, the content distributor MUST implement the **processing::master_dispatcher** interface, as specified in processing::master_dispatcher Server Details (section 3.3), and the **processing::procserver_handler** interface, as specified in section processing::procserver_handler Server Details (section 3.5). The item processing component acts as the protocol client for these two interfaces, as specified in processing::master_dispatcher Client Details (section 3.4) and processing::processor_server Client Details (section 3.2).

The item processing component MUST implement the **processing::processor_server** interface, as specified in processing::processor_server Server Details (section 3.1). The content distributor acts as the protocol client for the **processing::processor_server** interface, as specified in processing::processor_server Client Details (section 3.2).

The item processing component MUST implement the **Status** interface, as specified in Status Server Details (section 3.7).

The protocol client side of the **Status** interface is simply a pass-through. That is, no additional timers or other states are necessary 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 processing::processor_server Server Details

The **processing::processor_server** interface uses the Middleware Protocol, as specified in [MS-FSMW], as the transport mechanism.

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

The protocol server activates the **Status** interface, as specified in Status Server Details (section 3.7).

The protocol server is a **processing::master_dispatcher** protocol client and MUST register with the **processing::master_dispatcher** protocol server, as specified in Initialization (section 3.4.3).

3.1.4 Message Processing Events and Sequencing Rules

This interface includes the method that is listed in the following table.

Method	Description
process	Processes a sequence of item operations.

3.1.4.1 processing::processor_server::process

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

```
void process(in coreprocessing::session next_subsystem_session,
             in long session_id,
             in string collection,
             in cht::documentmessages::operation_set batch,
             in cht::documentmessages::subsystem_id_set subsystem_ids)
  raises (processing::shutting_down, processing::system_resource_error);
```

Input values:

next_subsystem_session: A **client proxy** for the **coreprocessing::session** server object to the indexing dispatcher.

session_id: A long integer value that contains the session identifier.

collection: A string that contains the name of the collection with which to associate the sequence of items.

batch: A Cheetah entity, as specified in cht::documentmessages::operation_set (section 2.2.11), that contains the sequence of item operations.

subsystem_ids: A Cheetah entity, as specified in cht::documentmessages::subsystem_id_set (section 2.2.34), that specifies where the item operations are sent after the processing has finished. The value of this Cheetah entity MUST be set to "indexing".

Return value:

None.

Exceptions thrown: The possible exceptions are listed in the following table.

Exception	Meaning
shutting_down	This exception MUST be raised if the protocol server is in the process of shutting down at the time that the **process** method call is made.
system_resource_error	This exception MUST be raised if the protocol server is unable to process the sequence of item operations because of a resource constraint, such as insufficient memory.

When it enters this method, the protocol server MUST set the **Processing** state to **1** in the **Status** interface, as specified in Status Server Details (section 3.7).

The protocol server MUST raise the **processing::shutting_down** exception if the **Terminating** state in the **Status** interface is **True**.

The protocol server writes an entry to the **Item Log Table** of the **Status** interface for each item that is processed during processing.

If the **Tracing** state is **True** in the **Status** interface, as specified in Status Server Details (section 3.7), the protocol server MUST write changes to item attributes to the **Item Log Table** in the **Status** interface.

For each item that is processed, the protocol server MUST create an **Item Log Table** in the **Status** interface in which an association between the item identifier and the structure in the table is set to the following values:

- **Status:** A string that contains the value "OK" if the item was processed without errors; a string that contains the value "WARNING" if the item was processed with one or more warnings; or a string that contains the value "ERROR" if an error occurred during processing.
- **Last Modified:** A value that MUST be the time, in number of seconds since January 1, 1970, when the item was last modified.
- **Messages:** A sequence that contains one entity for each logging event, where **Verbosity Level** MUST be set to a string that contains either "OK", "WARNING", or "ERROR", depending on the verbosity of the log entry. The **Message** entry MUST contain a textual message that describes the logging event.
- **Elapsed:** The number of seconds that it took to process the item.

For each sequence of item operations that is processed, the protocol server MUST create an **Item Operation Sequence Log Table** in the **Status** interface in which an association between the identifier of the sequence of item operations and the structure in the table is set to the following values:

- **Status:** A string that contains the value "OK" if the item operation sequence was processed without errors; a string that contains the value "WARNING" if the sequence was processed with one or more warnings; or a string that contains the value "ERROR" if an error occurred during processing.
- **Last Modified:** A value that MUST be the time, in number of seconds since January 1, 1970, when the item was last modified.
- **Messages:** A sequence that contains one entity for each logging event, where **Verbosity Level** MUST be set to a string that contains either "OK", "WARNING", or "ERROR", depending on the verbosity of the log entry. The **Message** entry MUST contain a textual message that describes the logging event.
- **Elapsed:** The number of seconds that it took to process the item.

For each item processing stage in the item processing pipeline, the protocol server MUST update the **Processing Stage Statistics Table** in the **Status** interface, as specified in Status Server Details (section 3.7), with the following values:

- **OK Items:** A value that is incremented by 1 for every item that was processed without any errors.
- **Error Items:** A value that is incremented by 1 for every item that was processed with an error.
- **Work Time:** A value that is incremented by the number of seconds that it took to process the item during this item processing stage.
- **User Time:** A value that is incremented by the number of seconds that the system was in user mode during the processing of this item.

- **System Time:** A value that is incremented by the number of seconds that the system was in system mode during the processing of this item.

For each item processing stage in the item processing pipeline, the protocol server MUST update the **Content Pipeline Statistics Table** in the **Status** interface, as specified in Status Server Details (section 3.7), with the following values:

- **OK Items:** A value that is incremented by 1 for every item that was processed without any errors.
- **Error Items:** A value that is incremented by 1 for every item that was processed with an error.
- **Work Time:** A value that is incremented by the number of seconds that it took to process the item during this item processing stage.
- **User Time:** A value that is incremented by the number of seconds that the system was in user mode during the processing of this item.
- **System Time:** A value that is incremented by the number of seconds that the system was in system mode during the processing of this item.

The protocol server MUST call the **process** method on the **next_subsystem_session** input value, which serves as a client proxy for the **coreprocessing::session** server object (as specified in [MS-FSDP] section 3), when the processing of the item operation sequence has finished.

The protocol server MUST call the **procserver_handler::processed** method, as specified in processing::procserver_handler::processed (section 3.5.4.3), when the processing of the item operation sequence has finished. Invoking **procserver_handler::processed** notifies the protocol client of the outcome from processing the items.

The protocol server MUST set the **Processing** state to **0** in the **Status** interface, as specified in Status Server Details (section 3.7), before it exits this method.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 processing::processor_server Client Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

busy state: A Boolean value that specifies whether this item processing component is processing items. **True** means that the item processing component is processing items; **False** means that the item processing component is not processing items.

item processing component identifier: A string that represents a unique identifier for the item processing component.

3.2.2 Timers

None.

3.2.3 Initialization

The **busy state** MUST initially be set to **False**.

3.2.4 Message Processing Events and Sequencing Rules

This interface includes the method that is listed in the following table.

Method	Description
process	Processes a sequence of item operations.

3.2.4.1 processing::processor_server::process

The protocol client MUST extract the item identifier from every item operation in the **cht::documentmessages::operation_set** input value, as specified in cht::documentmessages::operation_set (section 2.2.11), to the **processing::processor_server::process** method, as specified in processing::processor_server::process (section 3.1.4.1). How the protocol client does so depends on the type of item operation.

For each item operation in the **cht::documentmessages::operation_set** input value, the protocol client MUST do the following:

- Create an association between the **id attribute** of the item operation and the item identifier in the **item identifier holder**, as specified in Abstract Data Model (3.5.1). How the protocol client extracts the item identifier from the item operation depends on the type of item operation:
  - If the item operation is **cht::documentmessages::update_operation**, as specified in cht::documentmessages::update_operation (section 2.2.29), the item identifier is the **doc_id attribute** of the **document attribute**.
  - If the item operation is **cht::documentmessages::partial_update_operation**, as specified in cht::documentmessages::partial_update_operation (section 2.2.31), the item identifier is the **doc_id attribute** of the **document attribute**.
  - If the item operation is **cht::documentmessages::urlschange_operation**, as specified in cht::documentmessages::urlschange_operation (section 2.2.32), the item identifier is the **doc_id attribute** of the **document attribute**.
  - If the item operation is **cht::documentmessages::remove_operation**, as specified in cht::documentmessages::remove_operation (section 2.2.30), the item identifier is the **doc_id attribute**.
- Append the **id attribute** to the list in the **item operation holder**, as specified in Abstract Data Model (3.5.1), that is referenced by the **item processing component identifier** state.

- Create an association between the item operation identifier in the **cht::documentmessages::operation_set** input value and the **session_id** input value, and store that association in the **session mapper** state, as specified in Abstract Data Model (3.5.1).

When the protocol client calls the **processing::processor_server::process** method, it MUST set the **busy state** variable to **True**.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 processing::master_dispatcher Server Details

The **processing::master_dispatcher** interface uses the Middleware Protocol, as specified in [MS-FSMW], as the transport mechanism.

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

item processing component identifier holder: A collection of strings, each of which represents the **item processing component identifier** that is associated with an item processing component.

dispatcher holder: A set of client proxies for **processing::procserver_handler** server objects, where each server object can be referenced by a **dispatcher identifier**, as specified in Abstract Data Model (section 3.5.1).

3.3.2 Timers

None.

3.3.3 Initialization

The protocol server MUST call the **bind** method of the Middleware Protocol, as specified in [MS-FSMW] section 3, to register a **processing::master_dispatcher** server object in the name server.

The input values for the **bind** method are encapsulated in an **abstract object reference (AOR)**, as specified in [MS-FSMW] section 2. These input values MUST be as follows:

- **name:** A string that contains the value "esp/subsystems/processing/dispatcher".
- **object_id:** An integer that is unique for each server object**.**
- **host:** A string that contains the **fully qualified domain name (FQDN)** of the server object on the protocol server. The value is specific to the higher-level application.
- **port:** The **base port** + 390.

- **interface_type:** A string that contains the value "processing::master_dispatcher".
- **interface_version:** A string that contains the value "5.0".

3.3.4 Message Processing Events and Sequencing Rules

This interface includes the methods that are listed in the following table.

Method	Description
register_procserver	Registers an item processing component with the protocol server.
assign_dispatcher	Returns an identifier for a **processing::procserver_handler** interface.
unregister_procserver	Unregisters an item processing component from the protocol server.

3.3.4.1 processing::master_dispatcher::register_procserver

The **register_procserver** method registers a protocol client with the protocol server.

```
void register_procserver(in string name);
```

Input values:

name: A string that represents a unique identifier for the item processing component.

Return value:

None.

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

The protocol server MUST store the **name** string in the **item processing component identifier** state.

3.3.4.2 processing::master_dispatcher::assign_dispatcher

The **assign_dispatcher** method returns the identifier of a **processing::procserver_handler** interface.

```
long assign_dispatcher(in string name);
```

Input values:

name: A string that represents the unique identifier of the item processing component.

Return value:

The identifier of a **processing::procserver_handler** interface. If no **processing::procserver_handler** interfaces are available in the **dispatcher holder** state, the return value MUST be 1.

Exceptions thrown:

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

The protocol server MUST return the identifier of the **processing::procserver_handler** interface that was registered in the name server object, as specified in processing::procserver_handler Server Details (section 3.5).

3.3.4.3 processing::master_dispatcher::unregister_procserver

The **unregister_procserver** method unregisters a protocol client from the protocol server.

```
void unregister_procserver(in string name);
```

Input value:

name: A string that represents the unique identifier of the item processing component.

Return value:

None.

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

The protocol server MUST remove the name from the **item processing component identifier holder** object.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 processing::master_dispatcher Client Details

3.4.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

dispatcher handle: A client proxy for a **processing::procserver_handler** server object, as specified in processing::procserver_handler Server Details (section 3.5).

3.4.2 Timers

The Reregistration Lease timer forces the protocol client to reregister with the protocol server if it has not been processing items when this lease expires. The default value for this lease SHOULD be 180 seconds.

3.4.3 Initialization

The protocol client performs the following steps for initialization:

1. Create a client proxy for a **processing::master_dispatcher** server object, based on the abstract object reference (AOR) that is retrieved from the name server.

2. Call the **master_dispatcher::register_procserver** method to register the protocol client with the protocol server.

3. Call the **master_dispatcher::assign_dispatcher** method to receive an integer that the protocol client uses to resolve a **processing::procserver_handler** interface in the name server.

4. Call the **procserver_handler::register_procserver** method to register with the **processing::procserver_handler** interface.

The client side of the **processing::master_dispatcher** interface MUST use the **resolve** method of the Middleware Protocol, as specified in [MS-FSMW] section 2, to get the client proxy for the **processing::master_dispatcher** server object that is bound in the name server. The input values for the **resolve** method MUST be as follows:

name: A string that contains the value "esp/subsystems/processing/dispatcher".

interface_type: A string that contains the value "processing::master_dispatcher".

interface_version: A string that contains the value "5.0".

The protocol client MUST call the **processing::master_dispatcher::register_procserver** method, as specified in processing::master_dispatcher::register_procserver (section 3.3.4.1), with the **name** input value that uniquely identifies this instance of the protocol client. The protocol client MUST then call the **processing::master_dispatcher::assign_dispatcher method**, as specified in processing::master_dispatcher::assign_dispatcher (section 3.3.4.2), with the same **name** input value. The **assign_dispatcher** method returns a long integer value to the protocol client for use in resolving a **processing::procserver_handler** client proxy. The protocol client MUST replace *C* in the **name** input value with this returned value, as described in Initialization (section 3.6.3). The protocol client MUST store the **processing::procserver_handler** client proxy in the **dispatcher handle** state.

3.4.4 Message Processing Events and Sequencing Rules

None.

3.4.5 Timer Events

The Reregister Lease Timeout event reregisters the protocol client with the protocol server. To perform this reregistration, the protocol client:

1. Calls the **processing::master_dispatcher::unregister_procserver** method, as specified in processing::master_dispatcher::assign_dispatcher (section 3.3.4.2).

2. Calls the **processing::procserver_handler::unregister_procserver** method, as specified in processing::procserver_handler::unregister_procserver (section 3.5.4.2), on the

   ▪**processing::procserver** client proxy that is stored in the **dispatcher handle** state.

3. Registers with the protocol server, as specified in Initialization (section 3.4.3).

3.4.6 Other Local Events

If the Middleware Protocol, as specified in [MS-FSMW], raises a system exception during initialization, as specified in Initialization (section 3.4.3), the protocol client MUST restart the initialization procedure.

3.5 processing::procserver_handler Server Details

The **processing::procserver_handler** interface uses the Middleware Protocol, as specified in [MS-FSMW], as the transport mechanism.

3.5.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

item processing component holder: A set of **processing::processor_server** client proxy objects, where each proxy objet can be referenced by a string that contains the unique name of the item processing component.

item processing component pid holder: A set of integer values that represents the process identifiers for the item processing components, where each process identifier can be referenced by a string that contains the unique name of the item processing component.

dispatcher identifier: An integer greater than or equal to 0 that uniquely identifies this instance of the **processing::procserver_handler interface**.

item operation holder: A set of collections of item operation identifiers, where each collection can be referenced by a string that represents the identifier of an item processing component. This state holds which item processing component one or more item operations have been sent to.

item identifier holder: A set of string values that contains item identifiers, where each item identifier can be referenced by a long integer that contains the item operation identifier. This state is used to map item operation identifiers to item identifiers.

session mapper: A set of long integer values that represent session identifiers, where each session identifier can be referenced by a long integer that represents an item operation identifier.

3.5.2 Timers

The Lease Timeout timer measures the amount of time between when the protocol client calls the **processing::processor_server::process** method, as specified in processing::processor_server::process (section 3.1.4.1), and when the **processing::procserver_handler::processed** method is called. This is the amount of time that it takes to process a sequence of item operations, and this time MUST not exceed the value of the Lease Timeout timer. The default value SHOULD be 300 seconds. The protocol client sets the timer by invoking the **processing::procserver_handler::renew** method, as specified in processing::procserver_handler::renew (section 3.5.4.4).

3.5.3 Initialization

The protocol server MUST call the **bind** method of the Middleware Protocol, as specified in [MS-FSMW] section 3, to register a **processing::procserver_handler** server object in the name server.

Note that the input values for the **bind** method are encapsulated in an abstract object reference (AOR), as specified in [MS-FSMW] section 2. These input values MUST be as follows:

name: A string that contains the value "esp/subsystems/processing/dispatcher/*C*", where *C* is the **dispatcher identifier** state.

object_id: An integer that is unique for each server object**.**

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

port: The base port + 390.

interface_type: A string that contains the value " processing::procserver_handler".

interface_version: A string that contains the value "5.0".

The protocol server MUST insert an association between the **dispatcher identifier** state and a client proxy for this instance of the server object into the **dispatcher holder** state in the **processing::master_dispatcher** interface, as specified in Abstract Data Model (section 3.3.1).

3.5.4 Message Processing Events and Sequencing Rules

This interface includes the methods that are listed in the following table.

Method	Description
register_procserver	Registers a protocol client with the protocol server.
unregister_procserver	Unregisters a protocol client from the protocol server.
processed	Notifies the protocol server that the protocol client has finished processing a set of operations.
renew	Extends the time allowed for a protocol client to process a set of operations.

3.5.4.1 processing::procserver_handler::register_procserver

The **register_procserver** method registers an item processing component with the protocol server.

```
void register_procserver(in processor_server procserver,
                         in string name,
                         in string hostname,
                         in long pid,
                         in long priority);
```

Input values:

procserver: A client proxy for an item processing component that implements the **processing::processor_server** interface, as specified in processing::processor_server Server Details (section 3.1).

name: A string that represents the unique identifier of the item processing component. The value MUST match the one that was used when the item processing component called the **processing::master_dispatcher::register_procserver** method, as specified in processing::master_dispatcher::register_procserver (section 3.3.4.1).

hostname: The name of the host where the item processing component is running.

pid: A long integer that MUST contain the process identifier of the item processing component.

priority: A long integer that MUST contain 0.

Return value:

None.

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

The protocol server MUST make an association in the **item processing component holder** state between the client proxy contained in the **procserver** input value and the string contained in the **name** input value. The protocol server MUST also set the **item processing component identifier** state. For more information, see Abstract Data Model (section 3.2.1).

The protocol server MUST associate the **pid** input value with the **name** input value and store that association in the **item processing component pid holder** state, as specified in Abstract Data Model (section 3.5.1).

The protocol server MUST create an empty list that is referenced by the **name** input value and stored in the **item operation holder** state, as specified in Abstract Data Model (section 3.5.1).

3.5.4.2 processing::procserver_handler::unregister_procserver

The **unregister_procserver** method unregisters a protocol client from the protocol server.

```
void unregister_procserver(in string name);
```

Input values:

name: A string that represents the unique identifier of the item processing component. The value of this string MUST be the same as the value that was used when the item processing component called the **processing::master_dispatcher::register_procserver** method, as specified in processing::master_dispatcher::register_procserver (section 3.3.4.1).

Return value:

None.

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

The protocol server MUST remove the **processing::processor_server** client proxy that is associated with the **name** input value from the **item processor holder** state.

The protocol server MUST remove the process identifier that is associated with the **name** from the **item processor pid holder** state.

The protocol server MUST remove the list that is referenced by the **name** input value from the **item operation holder** state.

3.5.4.3 processing::procserver_handler::processed

The protocol client MUST call the **processed** method when it finishes processing a sequence of item operations—that is, a sequence of item operations received in the **processing::processor_server::process** method, as specified in processing::processor_server::process (section 3.1.4.1).

```
void processed(in string name,
               in boolean completed,
               in cht::documentmessages::operation_status_info status);
```

Input values:

name: A string that represents the unique identifier of the item processing component. The value MUST be the same as the value that was used when the item processing component called the **processing::master_dispatcher::register_procserver** method, as specified in processing::master_dispatcher::register_procserver (section 3.3.4.1).

completed: A Boolean value that MUST be **False** if the item processing component failed to send item operations by calling the **process** method of the **coreprocessing::session** interface, as specified in [MS-FSDP] section 3. Otherwise, the value MUST be **True**.

status: A **cht::document_messages::operation_status_info** Cheetah entity that contains the following attributes:

- **first_op_id**: The lowest identifier of the item operation of the **cht::documentmessages::operation_set** Cheetah entity that was submitted by using the **processing::processor_server::process** method, as specified in processing::processor_server::process (section 3.1.4.1).
- **last_op_id**: The highest identifier of the item operation of the **cht::documentmessages::operation_set** Cheetah entity that was submitted by using the **coreprocessing::session::process** method, as specified in processing::processor_server::process (section 3.1.4.1).
- **state**: A constant that MUST be the **documentmessages::completed** Cheetah enumeration constant, as specified in cht::documentmessages::operation_state (section 2.2.36).
- **subsystem**: A value that MUST be "processing".
- **errors**: A value that MUST consist of the errors associated with the item operations processed by the item processing component.
- **warnings**: A value MUST consist of the warnings associated with the item operations processed by the item processing component.

Return value:

None.

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

The protocol server MUST set the **busy state**, as specified in section Abstract Data Model (section 3.2.1), to **False** for the client proxy object in the **item processing component holder** state that is associated with the **name** input value.

The protocol server MUST remove the entries from the list that is referenced by the **name** input value in the **item operation holder** state that are between the values of **first_op_id** and **last_op_id**.

The protocol server MUST remove the session identifiers referenced by the identifiers of item operations in the **session mapper** state that are between the values of **first_op_id** and **last_op_id**.

The protocol server MUST make the callback messages in the **status** input value available to the content client, as specified in [MS-FSCF] section 3.

3.5.4.4 processing::procserver_handler::renew

The **renew** method is called by the protocol client to set the Lease Timeout timer, as specified in Timers (section 3.5.2).

```
void renew(in string name,
           in long period);
```

Input values:

name: A string that represents the unique identifier of the item processing component. The value MUST be the same as the value that was used when the item processing component called the **processing::master_dispatcher::register_procserver** method, as specified in processing::master_dispatcher::register_procserver (section 3.3.4.1).

period: The new Lease Timeout timer value, in seconds.

Return value:

None.

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

3.5.5 Timer Events

The Lease Timeout event restarts the item processing component. The remainder of this section describes what needs to occur to restart the item processing component.

The protocol client running the **configservice::config** interface MUST call the **resolve** method of the Middleware Protocol to get the AOR for the **configservice::config** server object that is bound in the name server, as specified in [MS-FSMW] section 2. The input values for the **resolve** method are:

- **name:** A string that MUST contain the value "fds/configservice".
- **interface_type:** A string that MUST contain the value "configservice::config".
- **interface_version:** A string that MUST contain the value "5.2".

The **configservice::config::get_active_module_list** method, as specified in [MS-FSCMW] section 3, supplies the port number for the node controller. The protocol client MUST call this method with a **module_type_name** string that contains the value "NodeControl".

The protocol server MUST then create an XML-RPC connection to the node controller by using the port number that was just retrieved and the fully qualified domain name (FQDN) of the host where the protocol server is running:

1. The protocol server MUST call the **GetProcessList** method on the node controller, as specified in [MS-FSNC] section 2, which returns a **ProcessList** structure, as specified in [MS-FSNC] section 2.

2. From the **ProcessList** structure, the protocol server MUST get the **ProcessInfo** element, as specified in [MS-FSNC] section 2, that has the process identifier corresponding to the parent process identifier of the processor identifier that is associated with the name of the unresponsive item processing component in the **item processing component pid holder** state.

3. The protocol server MUST call the **RestartProcess** method on the node controller, as specified in [MS-FSNC] section 3, with the **processes** input value, which is an array that MUST only contain one element. This element MUST be a string that contains the value of the **name** element associated with the **ProcessInfo** element that was retrieved from the **ProcessList** structure.

The protocol server MUST create a **cht::documentmessages::operation_set** Cheetah entity that contains one **cht::documentmessages::failed_operation** Cheetah entity, as specified in cht::documentmessages::failed_operation (section 2.2.33), for each entry in the list from the **item operation holder** state that is associated with the identifier of the item processing component that has been restarted.

The protocol server MUST create each **cht::documentmessages::failed_operation** Cheetah entity with the following attributes:

- **id:** A value that MUST consist of the item operation that was found in the **item operation holder** state for the specified item processing component.
- **subsystem:** A string that MUST contain the value "processing".
- **state:** A constant that MUST be the **documentmessages::lost** Cheetah enumeration constant, as specified in cht::documentmessages::operation_state (section 2.2.36).
- **doc_id:** A **cht::documentmessages::document_id** Cheetah entity, as specified in cht::documentmessages::document_id (section 2.2.15). The item operation identifier is used to get the item identifier in the **item identifier holder** state.
- **err:** A **cht::documentmessages::operation_lost** Cheetah entity, as specified in cht::documentmessages::operation_lost (section 2.2.10), in which the attributes are set as follows:

  - **error_code:** An integer that MUST contain the value 4.
  - **suggested_action:** A **cht::documentation::action** Cheetah enumeration, as specified in cht::documentmessages::action (section 2.2.1), that MUST have the value **resubmit**.
  - **subsystem:** A string that MUST contain the value "processing".
  - **description:** A string that MUST contain the value "failed to submit operation to next subsystem".
  - **session_id:** A long integer that MUST have been retrieved from the **session mapper** state referenced by the **id attribute**.
  - **operation_id:** A value that MUST be the same as that of the **id attribute**.

- **arguments:** A value that MUST consist of an empty Cheetah collection.

The protocol server MUST send the **cht::documentmessages::operation_set** Cheetah entity to the indexing dispatcher by calling the **coreprocessing::session::process** method, as specified in [MS-FSDP] section 3.

3.5.6 Other Local Events

When the protocol server receives a sequence of item operations from the content client for item processing, it MUST choose a **processing::processing_server** client proxy from the **item processing component holder** state for which the **busy state** of the **processing::processing_server** client interface is **False**. The protocol server MUST then use the **processing::processor_server** client proxy to send the sequence of item operations to the item processing component for item processing.

If no **processor::processor_server** client proxies exist for which the **busy state** is **False**, the protocol server MUST notify the content client that it is unable to process the sequence of item operations.

3.6 processing::procserver_handler Client Details

3.6.1 Abstract Data Model

None.

3.6.2 Timers

None.

3.6.3 Initialization

The client side of the **processing::procserver_handler** interface MUST call the **resolve** method of the Middleware Protocol, as specified in [MS-FSMW] section 2, to get the AOR for the **processing::procserver_handler** server object that is bound in the name server. The input values for the **resolve** method MUST be as follows:

name: A string that contains the value "esp/subsystems/processing/dispatcher/*C*", where *C* is a number greater than or equal to 0 that uniquely identifies this instance of the interface.

interface_type: A string that contains the value "processing::procserver_handler".

interface_version: A string that contains the value "5.0".

3.6.4 Message Processing Events and Sequencing Rules

None.

3.6.5 Timer Events

None.

3.6.6 Other Local Events

None.

3.7 Status Server Details

The **Status** interface uses XML-RPC, as specified in [XML-RPC], as the transport mechanism.

3.7.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Module Status Table: A structure that contains the following entries:

- **Start Time:** The time when the protocol server was started.
- **Idle Time:** The number of seconds that the protocol server has been idle—that is, not busy processing items.
- **Processing:** A Boolean value that indicates whether the protocol server is currently processing items. A value of **True** means that the protocol server is processing items; a value of **False** means that the protocol server is not processing items.
- **Logging Verbosity:** A string that contains the level of logging.

Last Reset: The last time that the state of the protocol server was cleared. This time is expressed in seconds since January 1, 1970.

Item Log Table: An association between an item identifier and a structure that contains the following entries:

- **Status:** A string that contains the status of the processing of the item.
- **Last Modified:** The time that the item was last modified. This time is expressed in number of seconds since January 1, 1970.
- **Messages:** A sequence of entities, each of which contains two values:
  - **Verbosity Level:** A string that contains the level of logging for this log message.
  - **Message:** A string that contains the log message.
- **Elapsed:** The number of seconds that it took to process the item.

Item Operation Sequence Log Table: An association between an item sequence identifier and a structure that contains the following values:

- **Status:** A string that contains the status of the processing of the item sequence.
- **Last Modified:** The time that the item sequence was last modified. This time is expressed in number of seconds since January 1, 1970.
- **Messages:** A sequence of entities, each of which contains two values:
  - **Verbosity Level:** A string that contains the level of logging for this log message.
  - **Message:** A string that contains the log message.

- **Elapsed:** The number of seconds that it took to process the item.

Content Pipeline Statistics Table: A structure that acts as a store for statistical information about the item processing that occurs in a content pipeline. This structure consists of an association between the name of a content pipeline (in the form of a string) and an entity that consists of the following values:

- **OK Items:** The number of items that have been processed successfully.
- **Error Items:** The number of items that have been processed with errors.
- **Work Time:** The number of seconds that have been spent on processing items.
- **User Time:** The number of seconds that have been spent in user mode.
- **System Time:** The number of seconds that have been spent in system mode.
- **Page Swaps:** The number of page faults that required input/output.
- **Virtual Memory:** A number of bytes of virtual memory that have been allocated.
- **Resident Memory:** A number of bytes of resident memory that have been allocated.
- **Memory Usage:** A number of bytes of total memory that have been allocated.

Processing Stage Statistics Table: A structure that acts as a store for statistical information about the item processing that occurs during an item processing stage. This structure consists of an association between the name of an item processing stage (in the form of a string) and an entity that consists of the following values:

- **OK Items:** The number of items that have been processed successfully.
- **Error Items:** The number of items that have been processed with errors.
- **Work Time:** The number of seconds that have been spent on processing items.
- **User Time:** The number of seconds that have been spent in user mode.
- **System Time:** The number of seconds that have been spent in system mode.
- **Page Swaps:** The number of page faults that required input/output.
- **Virtual Memory:** A number of bytes of virtual memory that have been allocated.
- **Resident Memory:** A number of bytes of resident memory that have been allocated.
- **Memory Usage:** A number of bytes of total memory that have been allocated.

Terminating: A Boolean value that indicates whether the protocol server is currently in the process of terminating. A value of **True** means that the protocol server is terminating; a value of **False** means that the protocol server is not terminating.

Tracing: A Boolean value that indicates whether the protocol server will write attribute changes to the **Item Log Table**. A value of **True** forces the protocol server to write item attribute changes; a value of **False** prevents the protocol server from writing item attribute changes.

3.7.2 Timers

The Idle timer maintains the number of seconds during which the protocol server has not processed item operations.

3.7.3 Initialization

The protocol server MUST set the **Start Time** entry in the **Module Status Table** state to the number of seconds since January 1, 1970.

The protocol server MUST set the **Log Verbosity** entry in the **Module Status Table** state to the string "normal".

The protocol server MUST set the **Terminating** state to **False**.

The protocol server MUST call the **ResetContainer** method, as specified in ResetContainer (section 3.7.4.3).

3.7.4 Message Processing Events and Sequencing Rules

This interface includes the methods that are listed in the following table.

Method	Description
ConfigurationChanged	Alerts the protocol server that a configuration change has occurred.
GetModuleStatus	Returns the status of the protocol server.
ResetContainer	Resets the internal state and
FlushState	Resets the internal state.
LeakDetect	Forces the logging of memory usage.
GetStatistics	Returns statistics and the elapsed amount of time since the last reset.
SetMemoryProfile	Enables or disables memory profiling.
SetDocumentTracing	Enables or disables item tracing.
GetDocumentStatusLogs	Returns the item logs for all logs.
GetDocumentStatusURIs	Returns a list of the URIs that have item logs.
GetBatchStatus	Returns the log for a single sequence of item operations.
GetBatchStatusIDs	Returns a list of identifiers for item operation sequences that have logs.
SetLogLevel	Sets the logging level at the protocol server.
Shutdown	Shuts down the protocol server.
ping	Checks whether the protocol server is responding.

3.7.4.1 ConfigurationChanged

The **ConfigurationChanged** method notifies the protocol server that the configuration has been updated and that the protocol server MUST reinitialize with the new configuration.

```
int ConfigurationChanged(string alert_type, alert_args);
```

Input values:

alert_type: A string that contains the type of alert.

alert_args: Additional arguments about the alert. The protocol server MUST be able to handle any data type for **alert_args**.

Return value: An integer that MUST be **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

If the **alert_type** input value is "configfile" and the **alert_args** input value consists of an array of two elements, where the first element is a string containing "DocumentProcessor" and the second element is a string containing "webcluster/reload_configfiles", the protocol server MUST call the **ResetContainer** method, as specified in ResetContainer (section 3.7.4.3).

If the **alert_type** input value is not "configfile", the protocol server MUST call the **ResetContainer** method, as specified in ResetContainer (section 3.7.4.3).

3.7.4.2 GetModuleStatus

The **GetModuleStatus** method returns the status of the protocol server.

```
struct GetModuleStatus();
```

Return value: A **ModuleStatus** structure, as specified in ModuleStatus (section 2.2.40).

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

This method returns a **ModuleStatus** XML-RPC structure. This method MUST create the structure based on the entries in the **Module Status Table** state, such that the **Started** attribute maps to the **Start Time** entry, the **Idletime** attribute maps to the **Idle Time** entry, the **Uptime** attribute maps to the **Uptime** entry, the **CurrentWork** attribute maps to the **Processing** entry, and the **Verbosity** attribute maps to the **Logging Verbosity** entry.

3.7.4.3 ResetContainer

The **ResetContainer** method resets the state of the protocol server and then notifies the configuration component about its capabilities.

```
int ResetContainer();
```

Return value: An integer that MUST be **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

The protocol server MUST call the **FlushState** method, as specified in FlushState (section 3.7.4.4).

If the higher-level implementation requires loading configuration files, the protocol server MUST call the **LoadConfigFile** method on the configuration component three times, as specified in [MS-FSCX] section 2, with the input values set as follows:

- The **module** input value set to "DocumentProcessor" and the **filepath** input value set to "formatdetector/user_converter_rules.xml". A specific file format MUST be used, as specified in [MS-FSPSCFG] section 2.
- The **module** input value set to "RTSearch" and the **filepath** input value set to "webcluster/fixmlmappings.xml". A specific file format MUST be used, as specified in [MS-FSSCFG] section 2.
- The **module** input value set to "Schema" and the **filepath** input value set to "webcluster/FieldProperties.xml". A specific file format MUST be used, as specified in [MS-FSSCFG] section 2.

If the higher-level implementation requires loading configuration files, the protocol server MUST call the **LoadConfigFileBase64** method on the configuration component four times, as specified [MS-FSCX] section 2, with the input values set as follows:

- The **module** input value set to "DocumentProcessor" and the **filepath** input value set to "ManagedProperties.xml". A specific file format MUST be used, as specified in [MS-FSPSCFG] section 2.
- The **module** input value set to "DocumentProcessor" and the **filepath** input value set to "OptionalProcessing.xml". A specific file format MUST be used, as specified in [MS-FSPSCFG] section 2.
- The **module** input value set to "DocumentProcessor" and the **filepath** input value set to "PropertyCategories.xml ". A specific file format MUST be used, as specified in [MS-FSPSCFG] section 2.
- The **module** input value set to "DocumentProcessor" and the **filepath** input value set to "linguistics/spelltuning.cfg". A specific file format MUST be used, as specified in [MS-FSST] section 2.

The protocol server MUST call the **RegisterModule** method on the configuration component, as specified in [MS-FSCX] section 2, with a **ModuleRegister** structure, as specified in [MS-FSCX] section 2, as an input value that contains the following members:

- **port:** An integer that contains the port number on which the protocol server listens to XML-RPC requests. This port number is specified as part of the higher-level implementation.
- **type:** A string that contains the value "ProcessorServer".
- **version:** A string that contains an implementation-specific value.
- **name:** A string that contains the value "ProcessorServer".
- **alerts:** An array of strings that contains the following values:
  - "pipelines"
  - "stages"
  - "collection"
  - "pipeline_added"

- "pipeline_modified"
- "pipeline_removed"
- "PipelineLogger"
- "configfile"
- "ProcessorServer"

The protocol server MUST call the **RegisterProcessorPipelines** method on the configuration component, as specified in [MS-FSCX] section 2, with the following input values:

processorserver: A tuple that MUST contain the fully qualified domain name (FQDN) and port number on which the protocol server listens to XML-RPC requests. This port number is specified as part of the higher-level implementation and MUST be the same port number as the one that was used in the preceding **RegisterMethod** method call.

- **pipelines:** An array that MUST contain a single element consisting of a string that contains the value "Office14 (webcluster)".

3.7.4.4 FlushState

The **FlushState** method flushes the internal state of the protocol server.

```
int FlushState();
```

Return value: An integer that MUST be **1** if the method succeeded or **0** if the method failed.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

The protocol server MUST clear the **Module Status Table**, **Item Log Table**, **Item Operation Sequence Log Table**, **Content Pipeline Statistics Table**, and **Processing Stage Statistics Table** states. The protocol server MUST also reset the Idle timer.

The protocol server MUST set the **Last Reset** state to the number of seconds since January 1, 1970.

3.7.4.5 LeakDetect

The **LeakDetect** method activates memory leak detection on the protocol server.

```
int LeakDetect();
```

Return value: An integer that MUST be **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

How the protocol server performs memory leak detection is implementation-specific.

3.7.4.6 GetStatistics

The **GetStatistics** method returns statistics about the item processing on the protocol server.

```
struct GetStatistics();
```

Return value: A **Statistics** structure, as specified in Statistics (section 2.2.41).

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

The protocol server MUST create the **Statistics** XML-RPC structure so that the number of seconds since the value in the **Last Reset** state maps to the value in the **Elapsed** attribute.

For each entry in the **Processing Stage Statistics Table** state, the protocol server MUST create a **ProcessingStatistics** structure, as specified in ProcessingStatistics (section 2.2.39). The protocol server MUST associate the name of the item processing stage with the structure and set that association as the first element in the **Statistics** array of the **Statistics** structure, as specified in Statistics (section 2.2.41). The protocol server MUST set the attributes of the **ProcessingStatistics** structure as follows:

- The **OK Items** entry maps to the **OK** attribute.
- The **Error Items** entry maps to the **ERROR** attribute.
- The **Work Time** entry maps to the **WorkTime** attribute.
- The **User Time** entry maps to the **UserTime** attribute.
- The **System Time** entry maps to the **SystemTime** attribute.
- The number of page faults that required input/output is queried from the operating system and inserted into the **PageSwaps** attribute.
- The number of bytes of virtual memory that was allocated for the protocol server process is queried from the operating system and inserted into the **VirtualMem** attribute.
- The number of bytes of resident memory that was allocated for the protocol server process is queried from the operating system and inserted into the **ResidentMem** attribute.
- The number of bytes of total memory that was allocated for the protocol server process is queried from the operating system and inserted into the **MemoryUsage** attribute.

For each entry in the **Content Pipeline Statistics Table** state, the protocol server MUST create a **ProcessingStatistics** structure, as specified in ProcessingStatistics (section 2.2.39). The protocol server MUST associate the name of the content pipeline with the structure and set that association as the second element in the **Statistics** array in the **Statistics** structure, as specified in Statistics (section 2.2.41). The protocol server MUST set the attributes of the **ProcessingStatistics** structure as follows:

- The **OK Items** entry maps to the **OK** attribute.
- The **Error Items** entry maps to the **ERROR** attribute.
- The **Work Time** entry maps to the **WorkTime** attribute.
- The **User Time** entry maps to the **UserTime** attribute.
- The **System Time** entry maps to the **SystemTime** attribute.

- The number of page faults that required input/output is queried from the operating system and inserted into the **PageSwaps** attribute.
- The number of bytes of virtual memory that was allocated for the protocol server process is queried from the operating system and inserted into the **VirtualMem** attribute.
- The number of bytes of resident memory that was allocated for the protocol server process is queried from the operating system and inserted into the **ResidentMem** attribute.
- The number of bytes of total memory that was allocated for the protocol server process is queried from the operating system and inserted into the **MemoryUsage** attribute.

3.7.4.7 SetMemoryProfile

The **SetMemoryProfile** method enables or disables memory profiling on the protocol server.

```
int SetMemoryProfile(int level);
```

Input values:

level: An integer that specifies whether to enable or disable memory profiling. A level of **0** MUST disable memory profiling, and a level of **1** (or any positive number greater than 1) MUST enable memory profiling.

Return value: An integer that MUST be **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

3.7.4.8 SetDocumentTracing

The **SetDocumentTracing** method enables or disables item tracing on the protocol server. When item tracing is enabled, the changes to an item during item processing are recorded in the **Item Log Table** state.

```
int SetDocumentTracing(int level);
```

Input values:

level: A nonnegative integer for which a value of **0** MUST disable item tracing, and a value of **1** (or any positive number greater than 1) MUST enable item tracing.

Return value: An integer that MUST be **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

If the **level** input value is **0**, the protocol server MUST set the **Tracing** state to the value **False**. If the **level** input value is any value other than **0**, the protocol server MUST set the **Tracing** state to the value **True**.

3.7.4.9 GetDocumentStatusLogs

The **GetDocumentStatusLogs** method MUST return all the log entries that exist in the **Item Log Table** state.

```
array GetDocumentStatusLogs();
```

Return value: A value that MUST consist of an **ItemStatusLog** structure, as specified in ItemStatusLog (section 2.2.44).

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

For each entry in the **Module Status Table** state, the protocol server MUST create an **ItemStatusLog** structure in which the **Status** attribute maps to the **Status** entry, the **Modified** attribute maps to the **Last Modified** entry, and the **Elapsed** attribute maps to the **Processing Time** entry.

For each entry in the **Messages** entity of the **Module Status Table**, the protocol server MUST create a **LogMessage** structure, as specified in LogMessage (section 2.2.42), in which the **Verbosity Level** entry maps to the first element and the **Message** entry maps to the second element. The protocol server MUST insert all the **LogMessage** structures into the **Msgs** attribute of the **ItemStatusLog** structure.

3.7.4.10 GetDocumentStatusURIs

The **GetDocumentStatusURIs** method MUST return an array of strings that contains all the item identifiers in the **Item Log Table** state.

```
array GetDocumentStatusURIs();
```

Return value: An array of strings that contains all the item identifiers in the **Item Log Table** state.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

3.7.4.11 GetBatchStatus

The **GetBatchStatus** method MUST return the entry in the **Item Operation Sequence Log Table** state that is associated with the specified identifier of an item operation sequence.

```
struct GetBatchStatus(int batchid);
```

Input values:

batchid: An integer that contains the identifier of an item operation sequence.

Return value: A value that MUST consist of an **ItemLog** structure, as specified in ItemLog (section 2.2.43).

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

For each entry in the **Item Operation Sequence Log Table** state, the protocol server MUST create an **ItemStatusLog** structure in which the **Status** attribute maps to the **Status** entry, the **Modified** attribute maps to the **Last Modified** entry, and the **Elapsed** attribute maps to the **Processing Time** entry.

For each entry in the **Messages** entity of the **Module Status Table** state, the protocol server MUST create a **LogMessage** structure, as specified in LogMessage (section 2.2.42), in which the **Verbosity Level** entry maps to the first element and the **Message** entry maps to the second element. The protocol server MUST insert all the **LogMessage** structures into the **Msgs** attribute of the **ItemStatusLog** structure.

3.7.4.12 GetBatchStatusIDs

The **GetBatchStatusIDs** method MUST return an array of strings that comprises the set of operation identifiers contained in the **Item Operation Sequence Log Table** state.

```
array GetBatchStatusIDs();
```

Return value: An array of strings that contains the set of operation identifiers that the protocol server has processed. That is, for each entry in the **Item Operation Sequence Log Table** state, the array will contain a string representation of the identifier of the item operation sequence.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

3.7.4.13 SetLogLevel

The **SetLogLevel** method controls the logging level of the protocol server.

```
int SetLogLevel(string level);
```

Input values:

level: A string that MUST contain the value "normal" for normal logging or the value "debug" for debug logging.

Return value: An integer that MUST contain the value **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

This method controls the level of logging for the **Item Log Table** and **Item Operation Sequence Log Table** states. The protocol server MUST store the **level** input value in the **Logging Verbosity** entry in the **Module Status Table** state.

3.7.4.14 Shutdown

The **Shutdown** method shuts down the protocol server.

```
int Shutdown();
```

Return value: An integer that MUST contain the value **1**.

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

Upon the invocation of this method, the protocol server MUST set the **Terminating** state in the protocol server to **True**.

3.7.4.15 ping

The **ping** method determines whether the protocol server is responding to requests.

```
string ping();
```

Return value: A string that MUST contain the value "pong".

Exceptions thrown: No exceptions are thrown beyond those thrown by the underlying XML-RPC protocol, as specified in [XML-RPC].

3.7.5 Timer Events

None.

3.7.6 Other Local Events

Upon termination, the protocol server MUST call the **UnregisterModule** method on the configuration component, as specified in [MS-FSCX] section 2, with a **ModuleRegister** structure, as specified in [MS-FSCX] section 2, as an input value that contains the following members:

- **port:** An integer that contains the port number on which the protocol server listens to XML-RPC requests. This port number is specified as part of the higher-level implementation.
- **type:** A string that contains the value "ProcessorServer".
- **version:** A string that contains an implementation-specific value. This MUST be the same string as the version input value used in the RegisterModule method when used in ResetContainer (section 3.7.4.3).
- **name:** A string that contains the value "ProcessorServer".
- **alerts:** An array of strings that contains the following values:
  - "pipelines"
  - "stages"
  - "collection"
  - "pipeline_added"
  - "pipeline_modified"
  - "pipeline_removed"
  - "PipelineLogger"
  - "configfile"
  - "ProcessorServer"

4 Protocol Examples

4.1 Processing a Sequence of Item Operations

This example describes the three main steps that are involved in processing a sequence of item operations:

1. An item processing component registers with the content distributor.
2. The content distributor sends a sequence of item operations to the item processing component for item processing.
3. The item processing component unregisters.

Initialization

1. The content distributor creates and activates a server object that implements the **processing::master_dispatcher** interface.
2. The content distributor creates and activates a server object that implements the **processing::procserver_handler** interface.
3. The item processing component creates a server object that implements the **processing::processor_server** interface.
4. The item processing component registers the server object with the content distributor by:
    1. Invoking the **processing::master_dispatcher::register_procserver** method.
    2. Getting a dispatcher identifier by invoking the **processing::master_dispatcher::assign_dispatcher** method.
    3. Using the dispatcher identifier, among other information, to resolve a **processing::procserver_handler** interface.
    4. Registering the **processing::processor_server** server object by invoking the **processing::procserver_handler::register_procserver** method.

Item processing

1. The content distributor receives a sequence of item operations for item processing from the content client.
2. The content distributor sends the sequence of item operations to the item processing component by invoking the **processing::processor_server::process** method.
3. The item processing component processes the item operations
4. The item processing component sends the item operations to the indexing dispatcher and reports the status back to the content distributor by invoking the **processing::procserver_handler::processed** method.

Shutting down

- The item processing component unregisters itself by first invoking **processing::procserver_handler::unregister_procserver** and then invoking **processing::master_dispatcher::unregister_procserver**.

4.1.1 Example Code: Initializing the Content Distributor

```
SET master_dispatcher_server_object_instance TO INSTANCE OF processing::master_dispatcher
SERVER OBJECT

SET master_dispatcher_server_object_host TO www.cohowinery.com

SET master_dispatcher_server_object_port TO 13328

SET master_dispatcher_server_object_interface_type TO "processing::master_dispatcher"

SET master_dispatcher_server_object_interface_version TO "5.0"

SET master_dispatcher_server_object_name TO "esp/clusters/webcluster/processing/dispatcher"

SET master_dispatcher_server_object_aor TO master_dispatcher_server_object_host,
master_dispatcher_server_object_port, master_dispatcher_server_object_interface_type,
master_dispatcher_server_object_interface_version AND
master_dispatcher_server_object_interface_name

CALL nameserver.bind WITH master_dispatcher_server_object_instance AND
master_dispatcher_server_object_aor

SET procserver_handler_server_object_instance TO INSTANCE OF processing::procserver_handler
SERVER OBJECT

SET procserver_handler_server_object_host TO www.cohowinery.com

SET procserver_handler_server_object_port TO 13328

SET procserver_handler_server_object_interface_type TO "processing::procserver_handler"

SET procserver_handler_server_object_interface_version TO "5.0"

SET procserver_handler_server_object_name TO
"esp/clusters/webcluster/processing/dispatcher/0"

SET procserver_handler_server_object_aor TO procserver_handler_server_object_host,
procserver_handler_server_object_port, procserver_handler_server_object_interface_type,
procserver_handler_server_object_interface_version AND
procserver_handler_server_object_interface_name

CALL nameserver.bind WITH procserver_handler_server_object_instance AND
procserver_handler_server_object_aor

WAIT FOR procserver_handler_server_object_instance.register_processor RETURNING
processor_server_instance, processer_server_name
```

4.1.2 Example Code: Initializing the Item Processing Component

```
SET master_dispatcher_object_name TO "esp/clusters/webcluster/processing/dispatcher"

SET master_dispatcher_object_interface_type TO "processing::master_dispatcher"

SET master_dispatcher_object_interface_version TO "5.0"
```

```
CALL nameserver.resolve WITH master_dispatcher_object_name,
master_dispatcher_object_interface_type AND master_dispatcher_object_interface_version
RETURNING master_dispatcher_client_proxy

SET procserver_id TO "procserver_1"

CALL master_dispatcher_client_proxy.register_procserver WITH procserver_id RETURNING void

CALL master_dispatcher_client_proxy.assign_dispatcher WITH procserver_id RETURNING
dispatcher_id

SET procserver_handler_object_name TO "esp/clusters/webcluster/processing/dispatcher/" +
dispatcher_id

SET procserver_handler_object_interface_type TO "processing::procserver_handler"

SET procserver_handler_object_interface_version TO "5.0"

CALL nameserver.resolve WITH procserver_handler_object_name,
procserver_handler_object_interface_type AND procserver_handler_object_interface_version
RETURNING procserver_handler_client_proxy

SET processor_server_server_object TO INSTANCE OF processor_server SERVER OBJECT

SET processor_server_hostname TO www.cohowinery.com

CALL get_process_identifier RETURNING pid

SET processor_server_priority TO 0

CALL procserver_handler.register_procserver WITH processor_server_server_object,
processor_server_hostname, pid AND processor_server_priority RETURNING void
```

4.1.3 Example Code: Dispatching Items

```
RECEIVE session, session_id, collection, operation_set, subsystem_id_set FROM CONTENT CLIENT

GET processor_server FROM item_processor_holder WHERE processor_server.busy == False

FOR EACH operation IN operation_set
    SET item_identifier_holder [operation.operation_id] = operation.document_id
    INSERT INTO item_operation_holder[processor_server] VALUES operation.operation_id

CALL processor_server.process WITH session, session_id, operation_set, collection,
subsystem_id_set RETURNING void

SET processor_server.busy TO True

WAIT FOR processor_server_server_object.processed CALL GIVING operation_set_status

SET processor_server.busy TO False

FOR EACH operation_id IN operation_set_status
    REMOVE FROM item_operation_holder[processor_server] VALUES operation_id
    REMOVE FROM item_identifier_holder[operation_id]

MAKE operation_set_status AVAILABLE TO CONTENT CLIENT
```

4.1.4 Example Code: Processing Items

```
WAIT FOR processor_server_object.process CALL GIVING session, session_id, collection,
operation_set, subsystem_ids

FOR EACH operation IN operation_set
    PROCESS operation GIVING operation_status
    INSERT operation_status INTO operation_status_set

CALL session.process WITH operation_set, "indexing"

CALL procserver_handler.processed WITH procserver_id, True, operation_status_set RETURNING
void
```

4.1.5 Example Code: Shutting Down the Item Processing Component

```
CALL procserver_handler.unregister_procserver WITH procserver_id RETURNING void
CALL master_dispatcher.unregister_procserver WITH procserver_id RETURNING void
```

4.1.6 Example Code: Shutting Down the Content Distributor

```
DEACTIVATE procserver_handler_server_object_instance
DEACTIVATE master_dispatcher_server_object_instance
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full FSIDL

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

```
module interfaces {
  module processing {

    exception shutting_down {};

    exception system_resource_error {
      string message;
    };

    interface processor_server {
    #     pragma version processor_server 5.2

      void process(in coreprocessing::session next_subsystem_session,
                   in long session_id,
                   in string collection,
                   in cht::documentmessages::operation_set batch,
                   in cht::documentmessages::subsystem_id_set subsystem_ids)
             raises (shutting_down, system_resource_error);

      void flush();

      void reassign();

    };

    interface procserver_handler {
    #     pragma version procserver_handler 5.2

      void register_procserver(in processor_server procserver,
                               in string name,
                               in string hostname,
                               in long pid,
                               in long priority);

      void unregister_procserver(in string name);

      void processed(in string name,
                     in boolean completed,
                     in cht::documentmessages::operation_status_info status);

      void flush_pipelines();

      void renew(in string name,
                 in long period);
    };

    interface master_dispatcher {
    #     pragma version master_dispatcher 5.0

      void register_dispatcher(in long node_id);

      void unregister_dispatcher(in long node_id);

      void register_procserver(in string procserver_id);
```

```
        long assign_dispatcher(in string procserver_id);

        void unregister_procserver(in string procserver_id);
      };
    };
  };
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® FAST™ Search Server 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

T

V