[MS-FSIADM]:
Indexer Administration and Status Protocol Specification

Revision Summary

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

Table of Contents

1 Introduction

This document specifies the Indexer Administration and Status Protocol, which is used for administering and retrieving information from an **indexing component**.

1.1 Glossary

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

fault-tolerant
XML

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

base port
Cheetah
Cheetah entity
client proxy
combined identifier
content collection
document identifier
exclusion list
FAST Index Markup Language (FIXML)
FAST Search Interface Definition Language (FSIDL)
index column
index generation identifier
index partition
indexer row
indexing component
indexing node
item
master indexer node
name server
query matching node
query processing node
Web crawler

The following terms are specific to this document:

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

1.2 References

1.2.1 Normative References

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

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

[MS-FSFIXML] Microsoft Corporation, "FIXML Data Structure"

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

[MS-FSIPA] Microsoft Corporation, "Index Publication and Activation Protocol Specification"

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

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

1.2.2 Informative References

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

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

1.3 Protocol Overview (Synopsis)

The Indexer Administration and Status Protocol uses three related interfaces. The interfaces are defined using the **FAST Search Interface Definition Language (FSIDL)** language, as specified in [MS-FSMW]. The **Indexer Administration** interface (indexer_admin) contains methods used to remotely control indexers, and the **Indexer Information** interface (indexer_info) contains methods used to retrieve status information from remote **indexing nodes**. Both rely on the **Indexer Information Callback** interface (indexer_info_callback) to asynchronously receive return data.

Item operations, such as adding new items or removing old items, arrive at the indexing node in batches containing several operations. Due to load balancing at higher levels, it can happen that items sent by a **Web crawler**, for example, arrive at the indexing node in a different order than that in which they were initially sent. Batches that arrive out of order are first placed on the spare queue. Batches that arrive in the correct order are placed on the api queue. After a batch has been placed on the api queue, the spare queue is checked for batches that could be placed on the api queue to form a contiguous range of batches. There is one spare queue for every client feeding to the indexing node.

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

In a **fault-tolerant** system setup there are several indexing nodes in the same index column, as shown in the following figure.

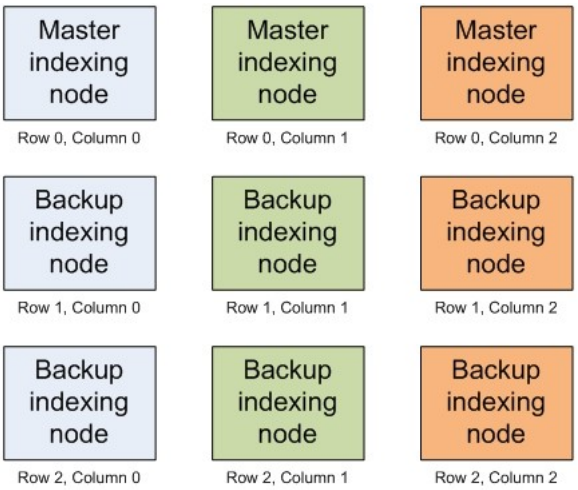

Figure 1: Matrix of indexing nodes

The different indexing nodes in an index column are identified by their **indexer row** number. In every index column, one of the indexing nodes assumes the role of the **master indexer node**, while the rest are referred to as the backup indexing nodes. The backup indexing nodes have all of the item data needed to build indices, but it is only the master indexer node that performs the actual indexing.

Prior to indexing, items are stored in the intermediate **FAST Index Markup Language (FIXML)** format, specified in [MS-FSFIXML]. A single FIXML structure potentially contains several items. The validity of the items in a FIXML structure is decided by its **FIXML meta structure**.

1.3.1 Indexer administration

The **Indexer Administration** interface (indexer_admin) provides methods for remotely controlling an indexing node. The provided functionality includes: suspending the ability to index new items, re-indexing the entire set of items from FIXML, re-indexing a particular index partition, enabling and disabling debug logging and performing a self-diagnosis on the integrity of the data structures.

1.3.2 Indexer Information

The **Indexer Information** interface (indexer_info) provides methods for remotely monitoring an indexing node. The interface provides methods for monitoring: the number of items indexed, how much capacity is left, the size of the api queue and spare queue and whether or not they are full, what items are indexed, how long ago an index partition was last re-indexed, and the search services subscribing to index partition updates.

1.3.3 Indexer Information Callback

The **Indexer Information Callback** interface (indexer_info_callback) provides methods for returning asynchronous callbacks to a protocol client invoking methods of the Indexer Administration or Indexer Information interfaces.

1.4 Relationship to Other Protocols

The interfaces are defined in the FSIDL language specified in [MS-FSMW]. The messages are transported using the HTTP based interoperability framework specified in [MS-FSMW].

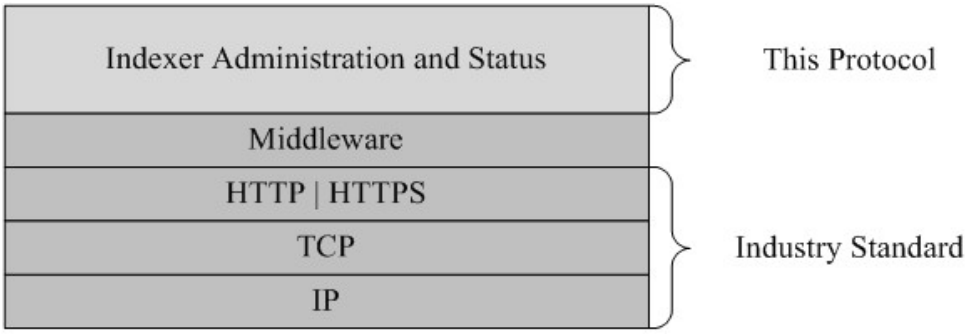

Figure 2: This protocol in relation to other protocols

Invoking some of the methods of this protocol will change state variables that will indirectly affect the behavior of the protocol specified in [MS-FSID]. See section 3.1.1 for a specification of the state variables concerned.

1.5 Prerequisites/Preconditions

The protocol client and protocol server are expected to know the location and connection information of the shared **name server**.

1.6 Applicability Statement

These protocols are applicable for search applications where there is a need to remotely control or monitor the indexing node.

1.7 Versioning and Capability Negotiation

Capability Negotiation: The Middleware protocol is connectionless, but the correct interface version is to be specified in every message, as specified in sections 3.1.3 and 3.2.3.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The messages supported by the interfaces specified in the following subsections MUST be sent as HTTP POST messages, as specified in [MS-FSMW].

2.2 Common Data Types

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

Cheetah entities MUST be encoded as specified in [MS-FSCHT] section 2. The checksum of the Cheetah messages MUST be -211918678. The Cheetah type identifier for the Cheetah entities MUST be as specified in the following table.

Cheetah entity	Cheetah type identifier
document_id	11
document_id_set	14

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

2.2.1 document_id

The **document_id** Cheetah entity uniquely identifies an item. The structure of this data type, as defined in section 6.2, is as follows:

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

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

routing_attributes: These MUST be an empty Cheetah **collection**.

2.2.2 document_id_set

The **document_id_set** Cheetah entity is a collection of **document_id** objects. The structure of this data type, as defined in section 6.2, is as follows:

```
root entity document_id_set {
  collection document_id doc_ids;
};
```

doc_ids: The collection of **document_id**s.

2.2.3 cheetah

The **cheetah** data type is an alias for a collection of bytes. It is used for all custom data types that are serialized using Cheetah. The structure of this data type, as defined in section 6.1, is as follows:

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

2.2.4 collection_list

The **collection_list** is an alias for a collection of strings, where each string is the name of a **content collection**. The structure of this data type, as defined in section 6.1, is as follows:

```
typedef sequence<string> collection_list;
```

2.2.5 search_controller_list

The **search_controller_list** is an alias for a collection of strings, where each string contains the location and status of a query matching node. The structure of this data type, as defined in section 6.1, is as follows:

```
typedef sequence<string> search_controller_list;
```

2.2.6 index_id_set_t

The **index_id_set_t** is an alias for a collection of strings, where each string is an **index generation identifier**. The structure of this data type, as defined in section 6.1, is as follows:

```
typedef sequence<string> index_id_set_t;
```

3 Protocol Details

3.1 Server Details

3.1.1 Abstract Data Model

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

The underlying Middleware protocol is connectionless, and the vast majority of the messages specified in section 3.1.4 are one-step methods: they either perform a single action on the indexing node, as in the case of the **indexer_admin** interface, or they retrieve a single piece of information, as in the case of the **indexer_info** interface. As a result, the protocol requires very little data organization.

The following data structures are needed for the state of the server:

indexing_suspended: A Boolean state variable. When **indexing_suspended** is TRUE, the ability to re-index partitions MUST be deactivated. Existing re-index jobs MUST be allowed to finish. When **indexing_suspended** is FALSE, the protocol server is in its normal state of operation as specified by this protocol and described in section 1.3. Changing the state of this variable also affects the behavior of a related protocol, see [MS-FSID] section 3.3.1.

reset_index_in_progress: A Boolean state variable. Set to TRUE when a reset index operation is in progress. The reset index procedure re-indexes the entire corpus of items into a single partition.

3.1.2 Timers

None.

3.1.3 Initialization

The protocol server MUST use the middleware **bind** method to register an **indexingengine::indexer_admin** server object in the name server, as specified in [MS-FSMW] section 3.4.4.2. The parameters for the **bind** method are encapsulated in an abstract object reference, as specified in [MS-FSMW] section 2.2.15:

host: A string holding the host name of the server object on the protocol server. The value is implementation specific of the higher level application, and is specified in the configuration file %FASTSEARCH%\etc\Node.xml.

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

interface_type: A string that MUST be "indexingengine::indexer_admin".

interface_version: A string that MUST be *"5.16"*.

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

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

The protocol server MUST use the middleware **bind** method to register an **indexingengine::indexer_info** server object in the name server, as specified in [MS-FSMW] section 3.4.4.2. The parameters for the **bind** method are encapsulated in an abstract object reference, as specified in [MS-FSMW] section 2.2.15:

host: A string holding the host name of the server object on the protocol server. The value is implementation specific of the higher level application and is specified in the configuration file %FASTSEARCH%\etc\Node.xml.

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

interface_type: A string that MUST be "indexingengine::indexer_info".

interface_version: A string that MUST be *"5.15"*.

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

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

3.1.4 Message Processing Events and Sequencing Rules

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

Some of the following specified methods use generic middleware exceptions to facilitate the return of error messages. In accordance with the middleware specification, the generic middleware exceptions may be thrown from any method, and are thus not defined in the FSIDL method signatures.

The available methods are specified in the following table.

Method	Description
indexer_admin::suspend_indexing	Sets the state variable **indexing_suspended** to TRUE.
indexer_admin::resume_indexing	Sets the state variable **indexing_suspended** to FALSE.
indexer_admin::reset_index	Re-indexes all indexed items from FIXML into one single index partition.
indexer_admin::enable_tracemode	Enables item tracking, which tracks an item's path through the indexing node using verbose logging.
indexer_admin::disable_tracemode	Disables item tracking, which tracks an item's path through the indexing node using verbose logging.
indexer_admin::enable_debug_logging	Enables debug level logging.
indexer_admin::disable_debug_logging	Disables debug level logging.
indexer_admin::request_reindex	Triggers an asynchronous re-indexing of an index partition.
indexer_admin::self_check	Performs consistency checking between the items in the FIXML files and the items in the set of active indexes.

Method	Description
indexer_info::get_percentage_full	Returns the percentage of items indexed in relation to the total capacity.
indexer_info::get_apiqueue_size	Returns the number of batches stored on the api queue.
indexer_info::get_sparequeue_size	Returns the number of batches stored on the spare queue.
indexer_info::get_apiqueue_full	Returns whether or not the api queue is full.
indexer_info::get_sparequeue_full	Returns whether or not the spare queue is full.
indexer_info::get_total_num_docs	Returns the total number of items indexed.
indexer_info::get_num_docs_in_collection	Returns the number of items in a content collection.
indexer_info::get_collection_names	Returns the names of all the content collections.
indexer_info::get_last_input_time	Returns the time since the set of indexed items last changed.
indexer_info::report_contents	Returns a list of all the indexed items.
indexer_info::report_collection_contents	Returns a list of all the indexed items for a specific content collection.
indexer_info::get_fixml_fill_rate	Returns the arithmetic mean of the fill rates of all FIXML files.
indexer_info::get_search_controllers	Returns the location and status of the query matching nodes registered with the indexing node.
indexer_info::get_column_role	Returns the role of the indexing node, that is, "Master" or "Backup".
indexer_info::get_document	Returns the FIXML of an item.
indexer_info::has_document	Returns whether or not the indexing node contains a specific item.
indexer_info::get_active_index_set	Returns the index generation identifiers of the active index set.
indexer_info::status	Returns an XML formatted status report of the indexing node.

3.1.4.1 indexer_admin::suspend_indexing

The **suspend_indexing** method sets the state variable **indexing_suspended** to TRUE. The structure of this method, as defined in section 6.1, is as follows:

```
void suspend_indexing();
```

3.1.4.2 indexer_admin::resume_indexing

The **resume_indexing** method sets the state variable **indexing_suspended** to FALSE. The structure of this method, as defined in section 6.1, is as follows:

```
void resume_indexing();
```

3.1.4.3 indexer_admin::reset_index

The **reset_index** method asynchronously re-indexes all indexed items from FIXML into one single index partition. Calling **reset_index** also sets the **reset_index_in_progress** state variable to TRUE. The structure of this method, as defined in section 6.1, is as follows:

```
void reset_index();
```

3.1.4.4 indexer_admin::enable_tracemode

The **enable_tracemode** method enables item tracking, which tracks an item's path through the indexer using verbose logging. The structure of this method, as defined in section 6.1, is as follows:

```
void enable_tracemode();
```

3.1.4.5 indexer_admin::disable_tracemode

The **disable_tracemode** method disables item tracking, which tracks an item's path through the indexing node using verbose logging. The structure of this method, as defined in section 6.1, is as follows:

```
void disable_tracemode();
```

3.1.4.6 indexer_admin::enable_debug_logging

The **enable_debug_logging** method enables debug level logging. The structure of this method, as defined in section 6.1, is as follows:

```
void enable_debug_logging();
```

3.1.4.7 indexer_admin::disable_debug_logging

The **disable_debug_logging** method disables debug level logging. The structure of this method, as defined in section 6.1, is as follows:

```
void disable_debug_logging();
```

3.1.4.8 indexer_admin::request_reindex

The **request_reindex** method triggers an asynchronous re-indexing of an index partition. The structure of this method, as defined in section 6.1, is as follows:

```
boolean request_reindex(in long partition);
```

partition: The index partition to re-index. This MUST be a valid index partition number.

Returns: MUST return FALSE if the index partition number is invalid, or if the state variables **reset_index_in_progress** or **indexing_suspended** are set to TRUE. If these conditions are not met, TRUE MUST be returned.

3.1.4.9 indexer_admin::self_check

The **self_check** method performs consistency checking between the items in the FIXML structures and the items in the set of active indices. Calling the **self_check** method sets the **indexing_suspended** state to TRUE. The result of the consistency check is sent back in the form of callbacks:

- For each valid item that is found in a FIXML structure, while at the same time the item is not present in the inverted index, an **indexer_info_callback::report_unindexed_document** message MUST be sent.

- For each instance where an item is present in a pair of inverted indices, a **indexer_info_callback::report_duplicate_document** message MUST be sent.

The structure of this method, as defined in section 6.1, is as follows:

```
void self_check(in indexer_info_callback callback);
```

callback: A reference to the server object receiving the callbacks.

3.1.4.10 indexer_info::get_percentage_full

The **get_percentage_full** method returns the percentage of items indexed in relation to the total capacity. The structure of this method, as defined in section 6.1, is as follows:

```
long get_percentage_full();
```

Returns: MUST return an integer value between 0 and 100 inclusive, describing the percentage of items indexed.

3.1.4.11 indexer_info::get_apiqueue_size

The **get_apiqueue_size** method returns the number of batches stored on the api queue. The structure of this method, as defined in section 6.1, is as follows:

```
long get_apiqueue_size();
```

Returns: MUST return the number of batches stored on the api queue.

3.1.4.12 indexer_info::get_sparequeue_size

The **get_sparequeue_size** method returns the number of batches stored on the spare queue. The structure of this method, as defined in section 6.1, is as follows:

```
long get_sparequeue_size();
```

Returns: MUST return the number of batches stored on the spare queue.

3.1.4.13 indexer_info::get_apiqueue_full

The **get_apiqueue_full** method returns whether or not the number of bytes stored on the api queue is greater than the total allowed size. The structure of this method, as defined in section 6.1, is as follows:

```
boolean get_apiqueue_full();
```

Returns: MUST return TRUE if the api queue is full, FALSE otherwise.

3.1.4.14 indexer_info::get_sparequeue_full

The **get_sparequeue_full** method returns whether or not the number of bytes stored on any of the spare queues is greater than the total allowed size. The structure of this method, as defined in section 6.1, is as follows:

```
boolean get_sparequeue_full();
```

Returns: MUST return TRUE if any of the spare queues are full, false otherwise.

3.1.4.15 indexer_info::get_total_num_docs

The **get_total_num_docs** method returns the total number of items indexed in all index partitions. The structure of this method, as defined in section 6.1, is as follows:

```
long get_total_num_docs();
```

Returns: MUST return the total number of items indexed.

3.1.4.16 indexer_info::get_num_docs_in_collection

The **get_num_docs_in_collection** method returns the number of items in a content collection. The structure of this method, as defined in section 6.1, is as follows:

```
long get_num_docs_in_collection(in string collection)
  raises (unknown_collection_error);
```

collection: The name of the content collection.

Returns: MUST return the number of items in the specified content collection.

Throws: MUST throw an **unknown_collection_error** if the specified content collection does not exist.

3.1.4.17 indexer_info::get_collection_names

The **get_collection_names** method returns the names of all the content collections. The structure of this method, as defined in section 6.1, is as follows:

```
collection_list get_collection_names();
```

Returns: MUST return a collection containing the names of all of the content collections.

3.1.4.18 indexer_info::get_last_input_time

The **get_last_input_time** method returns the time since the set of indexed items in a content collection has changed. The structure of this method, as defined in section 6.1, is as follows:

```
long long get_last_input_time(in string collection);
```

collection: The name of the content collection.

Returns: MUST return the seconds elapsed from midnight January 1st 1970 (UTC), not counting leap seconds, to the point in time when the set of indexed items last changed. MUST return 0 if the content collection does not exist.

3.1.4.19 indexer_info::report_contents

The **report_contents** method uses asynchronous callbacks to list all the items indexed on an indexing node. The protocol client specifies the maximum number of items to report in each callback, and the protocol server MUST then send an **indexer_info_callback::report_contents** callback until all items on the indexing node have been reported. The callbacks returned to the protocol client have sequence numbers, and the first callback returned MUST have the sequence number 0. For subsequent callbacks in the same sequence, the sequence number MUST be incremented by 1. When all items have been reported, the protocol server MUST send an **indexer_info_callback::report_finished** message to the protocol client.

The structure of this method, as defined in section 6.1, is as follows:

```
void report_contents(in indexer_info_callback callback,
                     in long batch_size);
```

callback: A reference to the server object receiving the callbacks.

batch_size: The maximum number of items reported in each callback.

3.1.4.20 indexer_info::report_collection_contents

The **report_collection_contents** method uses asynchronous callbacks to list all the items indexed in a specific content collection on an indexing node. The protocol client specifies the maximum number of items to report in each callback, and the protocol server MUST then send an **indexer_info_callback::report_collection_contents** callback until all items in the content collection have been reported. The callbacks returned to the protocol client have sequence numbers, and the first callback returned MUST have the sequence number 0. For subsequent callbacks in the same sequence, the sequence number MUST be incremented by 1. When all items have been reported, the protocol server MUST send an **indexer_info_callback::report_finished** message to the protocol client.

The structure of this method, as defined in section 6.1, is as follows::

```
void report_collection_contents(in indexer_info_callback callback,
                                in long batch_size,
                                in string collection);
```

callback: A reference to the server object receiving the callbacks.

batch_size: The maximum number of items reported in each callback.

collection: The name of the content collection.

3.1.4.21 indexer_info::get_fixml_fill_rate

The **get_fixml_fill_rate** method returns the arithmetic mean of the fill rates of all FIXML structures. The fill rate is the percentage of valid items in a FIXML structure. The validity of an item in a FIXML structure is defined by the FIXML meta structure. The structure of this method, as defined in section 6.1, is as follows:

```
float get_fixml_fill_rate();
```

Returns: MUST return the arithmetic mean of the fill rates of all FIXML structures.

3.1.4.22 indexer_info::get_search_controllers

The **get_search_controllers** method returns a collection of strings describing the location and status of the query matching nodes registered with the master indexer node. The format of each string is

"[hostname]:[port] - [status]"

where hostname is the query matching node's hostname, and port is the port number used by the query matching node for listening to requests from **query processing nodes**. The status string MUST be either "ok", "down" or "initializing".

To acquire the port number and the status string, the indexing node invokes the **search_controller::get_fdispatch_ptport** and **search_controller::get_status** methods as specified in [MS-FSIPA].

The structure of this method, as defined in section 6.1, is as follows:

```
search_controller_list get_search_controllers();
```

Returns: MUST return a collection of strings describing the location and status of all query matching nodes registered with the indexing node.

3.1.4.23 indexer_info::get_column_role

The **get_column_role** method returns the role of the indexing node: *"Master"* for a master indexer node, *"Backup"* for a backup indexing node, *"Standalone"* if the index column has a single indexing node and *"N/A"* if the role is unknown, for example, before an indexing node has had the chance to look for a registered master indexer node in the name server. The structure of this method, as defined in section 6.1, is as follows:

```
string get_column_role();
```

Returns: MUST return the role of the indexing node in the index column.

3.1.4.24 indexer_info::get_document

The **get_document** method returns the FIXML of an item. The structure of this method, as defined in section 6.1, is as follows:

```
string get_document(in cht::documentmessages::document_id doc_id,
                    in string collection);
```

doc_id: The document identifier (3) of the requested item.

collection: The name of the content collection containing the item.

Returns: MUST return a string containing the FIXML of the requested item, or an empty string if the item does not exist.

3.1.4.25 indexer_info::has_document

The **has_document** method returns whether or not the indexing node contains a specific item. The structure of this method, as defined in section 6.1, is as follows:

```
boolean has_document(in string internal_id)
  raises (invalid_arguments_exception);
```

internal_id: The **combined identifier** of the requested item.

Returns: MUST return TRUE if the item is indexed, FALSE otherwise.

Throws: MUST throw an **invalid_arguments_exception** if the **internal_id** argument does not conform to the syntax of a combined identifier.

3.1.4.26 indexer_info::get_active_index_set

The **get_active_index_set** method returns the index generation identifiers of the active index set. The structure of this method, as defined in section 6.1, is as follows:

```
index_id_set_t get_active_index_set();
```

Returns: MUST return the index generation identifiers of the active indices.

Throws: MUST throw a middleware **system_exception** if there is no active index set.

3.1.4.27 indexer_info::status

The **status** method returns an XML formatted report of the status of the indexing node. The structure of this method, as defined in section 6.1, is as follows:

```
string status();
```

Returns: A string that MUST be in **XML** format, which uses the XML Schema specified in section 7 and the content of which is implementation specific. The following are descriptions of elements and element specific attributes in this XML Schema.

3.1.4.27.1 indexer

The <indexer> element is the root element. Its attributes are specified in the following table.

Attribute name	Description
hostname	The fully qualified domain name of the indexing node.
port	The lowest port in the range of ports used by the indexing node.
cluster	MUST be "webcluster".
column	The index column number.
row	The indexer row number.
factory_type	MUST be "filehash".
preferred_master	"yes" if the indexing node is the preferred master, "no" otherwise. If the indexing node is the preferred master, it will force an already registered master to become a backup.
connected_feeds	The number of **sessions** connected to the indexing node. Creating sessions is handled by the protocol specified in [MS-FSID].
status	If the **indexing_suspended** state variable is TRUE, and the **reset_index_in_progress** state variable is FALSE, the status attribute MUST be set to "indexing suspended". However, if the **reset_index_in_progress** state variable is TRUE, the status attribute MUST be set to "reset index in progress". If both of the aforementioned state variables are FALSE, the status string MUST be set to "running ok".
heartbeat	If fault-tolerance is enabled, the attribute is either "Running" if the heartbeat is running, or "Suspended" if the heartbeat is suspended. If fault-tolerance is disabled, the attribute is set to "-". The heartbeat is a periodic thread that backup indexing nodes use to ascertain the availability of the master indexer node.
exclusion_interval	The interval, specified in seconds, at which the **exclusion lists** are updated.
time_since_last_index	The number of seconds since a new index set last became active, or 0 if no index set has become active during the uptime of the indexing node.

The child elements of the <indexer> element are specified in the following subsections.

3.1.4.27.1.1 documents

The <documents> element specifies information concerning the corpus of indexed items. Its attributes are specified in the following table.

Attribute name	Description
size	The total size, in bytes, of the FAST Index Markup Language (FIXML) structures describing the indexed items.
total	The total number of items intended for indexing.
indexed	The total number of items present in the active index set.

Attribute name	Description
not_indexed	The difference between the attributes "total" and "indexed".

3.1.4.27.1.2 column_role

The <column_role> element specifies the role of the indexing node in the index column. Its attributes are specified in the following table.

Attribute name	Description
state	The role of the indexing node. The string is identical to that returned by indexer_info::get_column_role.
backups	The number of backup indexing nodes registered with the master indexer node. If the Protocol Server is a backup indexing node, the value MUST be 0.

3.1.4.27.1.3 index_frequence

The <index_frequence> element specifies the number of items indexed per second. Its attributes are specified in the following table.

Attribute name	Description
min	The lowest measurement made of indexed items per second, or "0" if no new index has been built during the uptime of the indexing node.
max	The highest measurement made of indexed items per second, or "0" if no new index has been built during the uptime of the indexing node.

3.1.4.27.1.4 partition

The <partition> element specifies information concerning the index partitions. The root node contains one <partition> child element per index partition. Its attributes are specified in the following table.

Attribute name	Description
id	The index partition number.
index_id	A unique string identifying the current inverted index associated with the index partition.
status	The status string has either of the following values: "idle": The index partition is in an idle state. "waiting": The index partition is waiting on another resource. "copying (x%)": The inverted index associated with the index partition is being copied to another server. If the percentage that has finished copying is greater than or equal to 1, "x" MUST be replaced by an integer value specifying the percentage finished. "copying": Same as preceding case, used when the percentage finished is less

Attribute name	Description
	than 1. "indexing (x%)": The inverted index associated with the index partition is being re-indexed. If the percentage of items that has finished re-indexing is greater than or equal to 1, "x" MUST be replaced by an integer value specifying the percentage finished. "indexing": Same as preceding case, used when the percentage finished is less than 1. "unknown": Used when no other case applies.
type	The type of the index partition. If the index partition cannot be re-indexed the type is "static", otherwise it is "dynamic".
timestamp_indexed	The point in time when the index partition was last updated, specified as the number of seconds since midnight January 1st 1970 (UTC), not counting leap seconds.
indexed_per_second	The average number of items indexed per second, or "0" if no new index has been built for the index partition during the uptime of the indexing node.
pending_exclusionlist	The total number of items scheduled to be added to the exclusion list.
range	The indexed items are stored in FIXML structures that are numbered using increasing sequence numbers. The **range** attribute describes the range of FIXML structures included in the inverted index associated with the index partition. If no items are indexed, the range is specified as "0-0".

The child elements of the <partition> element are specified in the following subsections.

3.1.4.27.1.4.1 documents

The <documents> element specifies information concerning the indexed items in the index partition. Its attributes are specified in the following table.

Attribute name	Description
active	The total number of items in the inverted index associated with the index partition, subtracted by the number of items in the exclusion lists.
total	The total number of items in the inverted index associated with the index partition.

3.1.4.27.1.5 document_api

The <document_api> element specifies information concerning the api queue. Its attributes are specified in the following table.

Attribute name	Description
number_of_elements	The number of batches stored on the api queue.
last_sequence	The identifier of the last sequence operation processed.
frequence	The number of operations processed per second, measured as the average over the uptime of the indexing node. See the **operation Cheetah entity** as specified in [MS-FSID] section 2.2.16.

Attribute name	Description
load	The amount of memory used by the api queue as a percentage of the maximum allowed size.
status_updates	The number of failed item operations registered during the uptime of the indexing node. See the **operation** Cheetah entity as specified in [MS-FSID] section 2.2.16.

The child elements of the <document_api> element are specified in the following subsections.

3.1.4.27.1.5.1 queue_size

The <queue_size> element specifies the size of the api queue. Its attributes are specified in the following table.

Attribute name	Description
current	The size of the api queue, in bytes.

3.1.4.27.1.5.2 operations_processed

The <operations_processed> element specifies the number of items processed. Its attributes are specified in the following table.

Attribute name	Description
api	The number of item operations processed during the uptime of the indexing node. See the **operation** Cheetah entity specified in [MS-FSID].

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 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.

The underlying Middleware protocol is connectionless, and the vast majority of the messages specified in section 3.1.4 are one-step methods: they either perform a single action on the indexing node, in the case of the **indexer_admin** interface, or they retrieve a single piece of information, as in the case of the **indexer_info** interface. As a result, the protocol requires very little data organization. The exceptions are the three messages that involve callbacks:

indexer_admin::self_check, indexer_info::report_contents and **indexer_info::report_collection_contents**.

The following data structures are needed for the state of the client:

callback: A server object implementing the **indexer_info_callback** interface. The server object receives the asynchronous responses to the messages **report_contents**, **report_collection_contents** and **self_check**.

3.2.2 Timers

None.

3.2.3 Initialization

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

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

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

interface_type: A string that MUST be "indexingengine::indexer_admin".

version: A string that MUST be "*5.16*".

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

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

interface_type: A string that MUST be "indexingengine::indexer_info".

version: A string that MUST be "*5.15*".

To call methods that have parameters of type **indexer_info_callback**, a server object implementing the interface **indexer_info_callback** MUST first be created. The callback server object MUST have the following values set for the abstract object reference, as specified in [MS-FSMW] section 2.2.15:

host: A string holding the host name of the server object on the protocol server. The value is implementation specific of the higher level application and is specified in the configuration file %FASTSEARCH%\etc\Node.xml.

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

interface_type: A string that MUST be "indexingengine::indexer_info_callback".

interface_version: A string that MUST be "*5.6*".

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

3.2.4 Message Processing Events and Sequencing Rules

The client only processes messages that are the result of the client itself invoking callback generating methods. A client's invocation of each method typically results from local application activity, which specifies values for all input parameters.

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

The following table specifies the available methods.

Method	Description
indexer_info_callback::report_contents	Used to receive asynchronous callbacks in response to the protocol client calling the **indexer_info::report_contents** method.
indexer_info_callback::report_collection_contents	Used to receive asynchronous callbacks in response to the protocol client calling the **indexer_info::report_collection_contents** method.
indexer_info_callback::report_finished	Used to receive asynchronous callbacks in response to the protocol client calling either **indexer_info::report_collection** or **indexer_info::report_collection_contents.**
indexer_info_callback::report_unindexed_document	Used to receive asynchronous callbacks in response to the protocol client calling the **indexer_info::self_check** method.
indexer_info_callback::report_duplicate_document	Used to receive asynchronous callbacks in response to the protocol client calling the **indexer_info::self_check** method.

3.2.4.1 indexer_info_callback::report_contents

The **report_contents** method is an asynchronous callback sent back to the protocol client as a result of the protocol client issuing an **indexer_info::report_contents** call. The method MUST accept a sequence of callbacks from the protocol server, where each callback contains a different subset of the total corpus of items on the indexer. The structure of this method, as defined in section 6.1, is as follows:

```
void report_contents(in long column_id,
                     in long row_id,
                     in cht::documentmessages::document_id_set set,
                     in long batch_num);
```

column_id: The index column number of the indexer sending the callback.

row_id: The indexer row number of the indexer sending the callback.

set: A set of document identifiers (3) describing indexed items on the indexer.

batch_num: An increasing integer identifying the callback. This value MUST begin at 0.

3.2.4.2 indexer_info_callback::report_collection_contents

The **report_collection_contents** method is an asynchronous callback sent back to the protocol client as a result of the protocol client issuing an **indexer_info::report_collection_contents**. The method MUST accept a sequence of callbacks from the protocol server, where each callback contains a different subset of the total corpus of items belonging to a specific content collection on the indexer. The structure of this method, as defined in section 6.1, is as follows:

```
void report_collection_contents(in long column_id,
                                in long row_id,
                                in string collection,
                                in cht::documentmessages::document_id_set set,
                                in long batch_num);
```

column_id: The index column number of the indexer sending the callback.

row_id: The indexer row number of the indexer sending the callback.

collection: The name of the content collection containing the reported documents.

set: A set of document identifiers (3) indexed on the indexer.

batch_num: An increasing integer identifying the callback. This value MUST begin at 0.

3.2.4.3 indexer_info_callback::report_finished

The **report_finished** method is an asynchronous callback sent back to the protocol client in response to the protocol client issuing an **indexer_info::report_collection** or an **indexer_info::report_collection_contents** call. The protocol server MUST use it to signal that all of the items have been reported using the methods specified in sections 3.2.4.1 and 3.2.4.2. The structure of this method, as defined in section 6.1, is as follows:

```
void report_finished(in long column_id,
                     in long row_id,
                     in long num_batches,
                     in long num_docs);
```

column_id: The column number of the indexing node sending the callback.

row_id: The row number of the indexing node sending the callback.

num_batches: The number of batches sent back to the client using indexer_info_callback::report_contents or indexer_info_callback::report_collection_contents.

num_docs: The number of documents reported back to the client using indexer_info_callback::report_contents or indexer_info_callback::report_collection_contents.

3.2.4.4 indexer_info_callback::report_unindexed_document

The **report_unindexed_document** method is an asynchronous callback sent back to the protocol client in response to the protocol client issuing an **indexer_admin::self_check** call. It MUST be accepted for every non-invalidated item found in a FIXML document, while at the same time the item is not present in the active index set. The structure of this method, as defined in section 6.1, is as follows:

```
void report_unindexed_document(in long column_id,
                               in long row_id,
                               in string combined_id,
                               in string message);
```

column_id: The column number of the indexing node sending the callback.

row_id: The row number of the indexing node sending the callback.

combined_id: The combined identifier of the un-indexed item.

message: The index generation identifier of the index partition where the item was expected to be found.

3.2.4.5 indexer_info_callback::report_duplicate_document

The **report_duplicate_document** method is an asynchronous callback sent back to the protocol client in response to the protocol client issuing an **indexer_admin::self_check** call. It MUST be accepted for every pair of index partitions containing the same item.

The structure of this method, as defined in section 6.1, is as follows:

```
void report_duplicate_document(in long column_id,
                               in long row_id,
                               in string combined_id,
                               in string message);
```

column_id: The index column number of the indexing node sending the callback.

row_id: The indexer row number of the indexing node sending the callback.

combined_id: The combined identifier of the duplicate document.

message: The index generation identifiers of the pair of index partitions containing the duplicate document. The index generation identifiers MUST be separated by a comma character.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

4 Protocol Examples

4.1 Example using Callbacks

This example describes how to use the **report_contents** method of the **indexer_info** interface.

First, the indexing node (indexer row 0, index column 0) acting as the protocol server in this transaction, creates a server object implementing the **indexer_info** interface. The protocol server then binds the server object in the name server. Subsequently, the protocol client acquires a **client proxy** to the **indexer_info** server object by resolving the server object in the name server. This is possible because both protocol client and protocol server have agreed a priori on both the location of the shared name server, and the symbolic name of the server object.

The protocol client then creates a server object implementing the **indexer_info_callback** interface. Callbacks are used to retrieve data asynchronously, as opposed to acquiring the result of a method call in the form of a return value from the method.

The protocol client is now ready to call the **report_contents** method on the **indexer_info** client proxy. The protocol client passes a reference to its callback server object as an argument to the method call, and the protocol server invokes methods on the received callback client proxy to send the results back.

4.1.1 Code Example

4.1.1.1 Server Initialization

```
SET server_object_instance TO INSTANCE OF indexer_info SERVER OBJECT
SET server_object_id TO UNIQUE INTEGER
SET server_object_host TO "myserver.mydomain.com"
SET server_object_port TO "1234"
SET server_object_interface_type TO "indexingengine::indexer_info"
SET server_object_interface_version TO "5.15"
SET server_object_name TO "esp/clusters/webcluster/indexing/indexer-0-0/indexer_info"
SET server_object_aor TO server_object_host, server_object_port,
server_object_interface_type, server_object_interface_version, server_object_id AND
server_object_name
CALL nameserver.bind WITH server_object_name AND server_object_aor
```

4.1.1.2 Client Message

```
SET server_object_name TO "esp/clusters/webcluster/indexing/indexer-0-0/indexer_info"
SET server_object_interface_type TO "indexingengine::indexer_info"
SET server_object_interface_version TO "5.15"
CALL nameserver.resolve WITH server_object_name, server_object_interface_type AND
server_object_interface_version RETURNING indexer_info_client_proxy
SET callback_server_object TO INSTANCE OF indexer_info_callback SERVER OBJECT
SET batch_size TO 100
CALL indexer_info_client_proxy.report_contents WITH callback_server_object AND batch_size
```

4.1.1.3 Server Response

```
SET row TO 0
SET column TO 0
SET batch_num TO 0
```

```
REPEAT
  SET reported_docs TO THE MINIMUM OF batch_size AND THE REMAINDING NUMBER OF
    UNREPORTED DOCUMENTS
  SET batch TO reported_docs UNREPORTED DOCUMENTS
  CALL callback_client_proxy.report_contents WITH row, column, batch AND batch_num
  INCREMENT batch_num
  ADD reported_docs TO num_documents
UNTIL ALL DOCUMENTS ARE REPORTED
CALL callback_client_proxy.report_finished WITH column, row, batch_num AND num_documents
```

4.1.2 Time Sequence Diagram

The protocol client receives the callback, and then aggregates all the batches it receives. This is described in the following diagram.

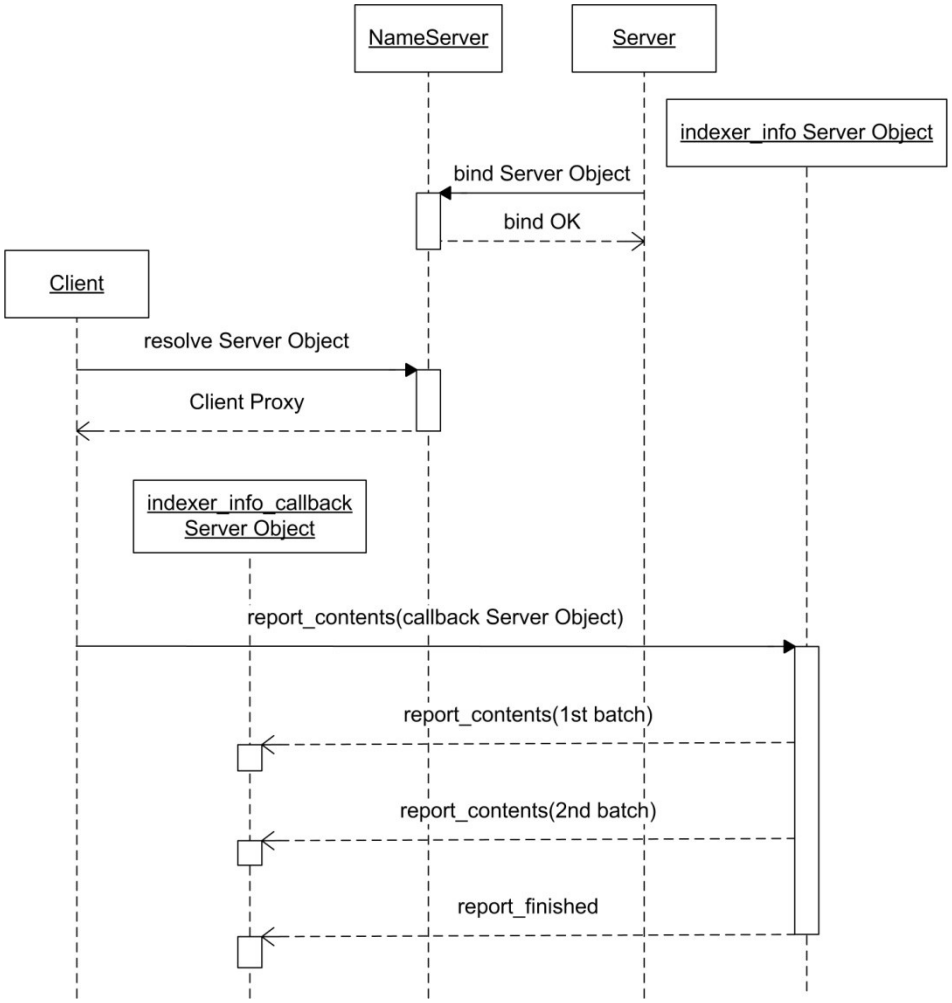

Figure 3: Protocol example

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full FSIDL

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

6.1 FSIDL

```
module interfaces {
  module indexingengine {
    exception invalid_input_exception {
      string what;
    }

    exception unknown_collection_error {
      string what;
    }

    typedef sequence<octet> cheetah;
    typedef cheetah document_id;
    typedef cheetah document_id_set;
    typedef sequence<string> collection_list;
    typedef sequence<string> search_controller_list;
    typedef sequence<string> index_id_set_t;

    interface indexer_admin {
      void suspend_indexing();
      void resume_indexing();
      void reset_index();
      void enable_tracemode();
      void disable_tracemode();
      void enable_debug_logging();
      void disable_debug_logging();
      boolean request_reindex(in long partition);
      void self_check(in indexer_info_callback callback);
    }

    interface indexer_info {
      long get_percentage_full();
      long get_apiqueue_size();
      long get_sparequeue_size();
      boolean get_apiqueue_full();
      boolean get_sparequeue_full();
      long get_total_num_docs();
      long get_num_docs_in_collection(in string collection)
        raises (unknown_collection_error);
      collection_list get_collection_names();
      long long get_last_input_time(in string collection);
      void report_contents(in indexer_info_callback callback,
                           in long batch_size);
      void report_collection_contents(in indexer_info_callback callback,
                                      in long batch_size,
                                      in string collection);
      float get_fixml_fill_rate();
      search_controller_list get_search_controllers();
      string get_column_role();
      string get_document(in cht::documentmessages::document_id doc_id,
                          in string collection);
      boolean has_document(in string internal_id)
```

```
        raises (invalid_arguments_exception);
      index_id_set_t get_active_index_set();
      string status();
    }

    interface indexer_info_callback {
      void report_contents(in long column_id,
                           in long row_id,
                           in cht::documentmessages::document_id_set set,
                           in long batch_num);
      void report_collection_contents(in long column_id,
                                      in long row_id,
                                      in string collection,
                                      in cht::documentmessages::document_id_set set,
                                      in long batch_num);
      void report_finished(in long column_id,
                           in long row_id,
                           in long num_batches,
                           in long num_docs);
      void report_unindexed_document(in long column_id,
                                     in long row_id,
                                     in string combined_id,
                                     in string message);
      void report_duplicate_document(in long column_id,
                                     in long row_id,
                                     in string combined_id,
                                     in string message);
    }
  }
}
```

6.2 Cheetah

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

root entity document_id_set {
  collection document_id doc_ids;
}
```

7 Appendix B: XML Schema for status method

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="column_role">
    <xs:complexType>
      <xs:attribute name="backups" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="state" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="document_api">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="queue_size" />
        <xs:element ref="operations_processed" />
      </xs:sequence>
      <xs:attribute name="number_of_elements" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="status_updates" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="frequence" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="load" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="last_sequence" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="documents">
    <xs:complexType>
      <xs:attribute name="size" type="xs:NMTOKEN" use="optional" />
      <xs:attribute name="total" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="indexed" type="xs:NMTOKEN" use="optional" />
      <xs:attribute name="active" type="xs:NMTOKEN" use="optional" />
      <xs:attribute name="not_indexed" type="xs:NMTOKEN" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="index_frequence">
    <xs:complexType>
      <xs:attribute name="max" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="min" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="indexer">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="documents" />
        <xs:element ref="column_role" />
        <xs:element ref="index_frequence" />
        <xs:element ref="partition" maxOccurs="unbounded" />
        <xs:element ref="document_api" />
      </xs:sequence>
      <xs:attribute name="exclusion_interval" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="preferred_master" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="heartbeat" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="time_since_last_index" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="connected_feeds" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="port" type="xs:NMTOKEN" use="required" />
```

```
        <xs:attribute name="column" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="factory_type" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="status" type="xs:string" use="required" />
        <xs:attribute name="hostname" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="cluster" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="row" type="xs:NMTOKEN" use="required" />
      </xs:complexType>
    </xs:element>

    <xs:element name="operations_processed">
      <xs:complexType>
        <xs:attribute name="api" type="xs:NMTOKEN" use="required" />
      </xs:complexType>
    </xs:element>

    <xs:element name="partition">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="documents" />
        </xs:sequence>
        <xs:attribute name="pending_exclusionlist" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="status" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="indexed_per_second" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="range" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="timestamp_indexed" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="type" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="index_id" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="id" type="xs:NMTOKEN" use="required" />
      </xs:complexType>
    </xs:element>

    <xs:element name="queue_size">
      <xs:complexType>
        <xs:attribute name="current" type="xs:NMTOKEN" use="required" />
      </xs:complexType>
    </xs:element>

  </xs:schema>
```

8 Appendix C: 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.

9 Change Tracking

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

10 Index

T

V