[MS-FSSPRADM]:
SPRel Administration and Status Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Revision Summary

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

Table of Contents

1 Introduction

This document specifies the SPRel Administration and Status Protocol for transmitting status and configuration options between a protocol client and a protocol server. This protocol enables the protocol client to control the protocol server, as well as to query it for status information. This protocol is a pure client/server protocol.

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

1.1 Glossary

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

Augmented Backus-Naur Form (ABNF)
Coordinated Universal Time (UTC)
Hypertext Transfer Protocol (HTTP)
UTF-8

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

associated query
base port
crawl collection
document identifier
search clickthrough
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

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

1.2.1 Normative References

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

[ISO-8601] International Organization for Standardization, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO/IEC 8601:2004, December 2004, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30

Note There is a charge to download the specification.

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

[MS-FSXTAPI] Microsoft Corporation, "XML-RPC Translatable API Structure 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

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.rfc-editor.org/rfc/rfc5234.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)

This protocol allows a protocol client to obtain different kinds of status information from the protocol server and to manage the process that the protocol server uses for extraction and analysis. For purposes of this protocol, a protocol server is an application that extracts **associated queries** based on **search clickthrough** logs.

This protocol uses XML-RPC, a remote procedure call protocol that uses XML to encode methods and responses, and **Hypertext Transfer Protocol (HTTP)** to serve as a transport mechanism.

The sequence of communication is as follows:

1. The application that is running on the protocol client sends a request to the protocol server.

2. The protocol server sends a response to the protocol client.

The protocol server does not initiate communication with the protocol client. The protocol client has information about the host name and port of the protocol server before it initiates communication with the protocol server.

1.4 Relationship to Other Protocols

This protocol uses XML-RPC to format requests and responses. This protocol transmits these messages by using the HTTP protocol, as shown in the following diagram:

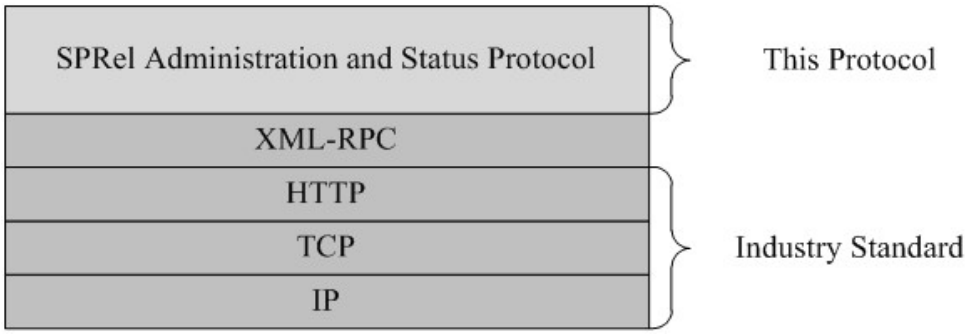

Figure 1: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

The protocol client is required to obtain the host name and port of the protocol server before this protocol is initiated.

1.6 Applicability Statement

This protocol is designed for transmitting status and configuration options between the protocol client and the protocol server.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

This protocol uses the transport protocol that is specified in [XML-RPC]. The syntax used to specify the XML-RPC methods in this specification is translated to xml as specified in [MS-FSXTAPI].

2.2 Common Data Types

The format of the XML body requests and responses is specified in [XML-RPC]. The HTTP POST path, as specified in [RFC2616], MUST be "/RPC2". The protocol server and the protocol client MUST support both HTTP version 1.0 and HTTP version 1.1.

Implementers MUST encode the following data types, as specified in [XML-RPC]:

- **array**
- **boolean**
- **double**
- **int**
- **string**
- **struct**

This protocol also specifies a **dynamic** type that implementers use to define arguments of type **boolean**, **double**, **int**, or **string**. The size of the dynamic type varies based on which [XML-RPC] type the **dynamic** type represents. All strings MUST use **UTF-8** encoding.

Some messages contain **double** values that are encoded as **string** values. Such a value MUST consist of an ASCII string that abides by the following **Augmented Backus-Naur Form (ABNF)** ([RFC5234]) rules:

```
floatnumber   =  pointfloat / exponentfloat
pointfloat    =  ([intpart] fraction) / (intpart ".")
exponentfloat =  intpart / pointfloat exponent
intpart       =  <1>*<17>DIGIT
fraction      =  "." <1>*<16>DIGIT
exponent      =  "e" ["-"] <1>*<3>DIGIT
```

2.2.1 Configuration Options

This section specifies all the configuration options that MUST be supported.

2.2.1.1 Global Configuration Options

The following table describes all the configuration options that are global for the protocol server.

Option name	Type	Description
batch_size	**int**	Specifies the number of operations that can be contained in each batch that the processing nodes submit to the indexing engine.

Option name	Type	Description
callback_timeout	**int**	Specifies the number of seconds that the processing nodes are required to wait for a response after sending the final batch of operations to the indexing engine.
concurrent_feeds	**int**	Specifies the number of processes that can simultaneously send operations to the indexing engine. If the value of the **cpus** configuration option, described later in this table, is less than that of **concurrent_feeds**, the **cpus** option MUST be used to determine the number of processes used.
cpus	**int**	Specifies the number of operations that can simultaneously run on each processing node in the system.
force_collections	**boolean**	A Boolean value that indicates whether the data contained in the **crawl collections** is to be updated. A value of **true** indicates that all the crawl collections MUST be updated. A value of **false** indicates that the existing data MUST be used.
keep_clicklogs	**int**	The number of days for which search clickthrough logs are kept. This value MUST be larger than 0 and MUST NOT be lower than the **use_clicklogs** value. For information about the **use_clicklogs** option, see the following.
pollwalsr_interval	**int**	Specifies the number of seconds between times that the protocol server checks for new data.
run_partial_update	**boolean**	A Boolean value that specifies whether partial update operations MUST be submitted for all changed documents in the **search index**. A value of **true** indicates that partial update operations MUST be submitted for all changed documents in the search index.
sort_buffer	**int**	Specifies the amount of memory, in megabytes, to use when sorting data. This value is specified per task running in the system.
use_clicklogs	**int**	The number of search clickthrough logs to be included in the analysis. This value MUST be greater than 0.

2.2.1.2 FAST Distributed Make Configuration Options

The following table describes all the FAST Distributed Make (FDM) configuration options.

Option name	Type	Description
disk_free	**int**	The minimum amount of available disk space needed on each node. This is a global value, and all nodes where the amount is smaller than this limit MUST NOT participate in the analysis. The value MUST be in megabytes and MUST NOT be lower than 1.
verbose	**boolean**	If **true**, verbose logging MUST be enabled for the processing framework. This is not the same as the log levels in section 2.2.8. This kind of logging MUST only contain information about the analysis.

2.2.2 Status Structure

The following table specifies the content of the structure that is returned by the **GetStatus** method.

Member name	Type	Description
clicklog_dates	**array**	If the protocol server extracted search clickthrough logs, this array MUST contain two dates as specified in section 2.2.3.1. If the protocol server did not extract search clickthrough logs, the array MUST be empty.
collections	**array**	This array MUST contain all the crawl collections in the system. Every element in the array MUST be a structure, as specified in section 2.2.3.2.
end_factor	**int**	This value represents the number of partitions into which the output of the relevance analysis is divided. This value MUST always equal the number of elements contained in the **lookupdbs** array, defined later in this table.
lookupdbs	**array**	This array contains the web analyzer lookup database nodes. The array MUST contain one entry for each web analyzer lookup database entry. Each array entry MUST be a two-element array in which the first element, of type **string**, specifies a host name, and the second element is of type **int**.
proc_generation	**int**	The number of times that the analysis was run.
procstatus	**array**	This array contains processing status information, as specified in section 2.2.3.3.
runtimes	**struct**	This structure contains run statistics, as specified in section 2.2.3.4.
schedule	**array**	This array MUST be a schedule array, as specified in section 2.2.7.
split_factor	**int**	This value represents the number of partitions into which the data is divided.
systemstatus	**array**	This array MUST be a system status array, as specified in section 2.2.3.5.
workers	**array**	This array contains the web analyzer worker nodes. The array MUST contain one entry for each web analyzer worker node entry. Each array entry MUST be a two-element array in which the first element, of type **string**, specifies a host name, and the second element, of type **string**, specifies a path.

2.2.3 Nested Status Arrays and Structures

This section specifies all the arrays and structures that are contained within the top-level status structures as specified in section 2.2.2.

2.2.3.1 clicklog_dates Array

The **clicklog_dates** array is returned as part of the status structure as specified in section 2.2.2. The following table specifies the content of the **clicklog_dates** array.

Element	Type	Description
0	**string**	The oldest search clickthrough log that is kept by the protocol server. The value MUST be a date that is formatted as specified in [ISO-8601].
1	**string**	The newest search clickthrough log that is kept by the protocol server. The value MUST be a date that is formatted as specified in [ISO-8601].

2.2.3.2 Crawl Collection Status Structure

The crawl collection status structure is returned as part of the status structure, as specified in section 2.2.2. The following table specifies the content of the crawl collection status structure.

Member name	Type	Description
cleared	**string**	This value specifies the date the crawl collection was last cleared of content. The value MUST be formatted as specified in [ISO-8601]. If the crawl collection was not cleared, this value MUST be the same as the **created** value, defined later in this table.
created	**string**	This value specifies the date the crawl collection was created. The value MUST be formatted as specified in [ISO-8601].
generation	**string**	This value specifies the number of times that the crawl collection was processed.
last_time_finished	**string**	This value specifies the last time that the crawl collection finished a processing run. The value MUST be a date formatted as specified in [ISO-8601] or an empty string.
last_time_started	**string**	This value specifies the last time that the crawl collection began a processing run that finished. The value MUST be a date formatted as specified in [ISO-8601] or an empty string.
name	**string**	The name of the crawl collection.
status	**int**	This value specifies the status of the crawl collection. The value MUST be one or more of the values specified in section 2.2.5.
timestamp	**string**	This value specifies the last time that the crawl collection received new data. The value MUST be a date formatted as specified in [ISO-8601] or an empty string.

2.2.3.3 procstatus Array

The **procstatus** array is contained in the view status structure, as specified in section 2.2.2. The following table specifies the content of the **procstatus** array.

Element	Type	Description
0	**string**	This value specifies the overall processing status for the protocol server: ▪ If the analysis is running, the value MUST be "running". ▪ If the analysis is running but a stop command has been issued, the value MUST be "stopping". ▪ If the analysis is stopped, the value MUST be "stopped". ▪ If the analysis is running but a pause command has been issued, the value MUST be "pausing". ▪ If the analysis is paused, the value MUST be "paused". ▪ If none of the preceding conditions apply, the value MUST be "ready".

Element	Type	Description
1	**string**	This value specifies the stage that the analysis is in. For more details, see section 2.2.4. The value MUST be encoded as a string.
2	**string**	This value specifies the percentage of the current stage that has finished. The value MUST be a **double** value that is encoded as a **string**.

2.2.3.4 runtimes Structure

The **runtimes** structure is returned as part of the status structure, as specified in section 2.2.2. The **runtimes** structure contains statistics about the time that the protocol server takes to process search clickthrough logs. The following table specifies the content of the **runtimes** structure.

Member name	Type	Description
count	**int**	This value specifies the number of analysis runs that the protocol server finished.
lastn	**array**	This array MUST contain a value that indicates the number of seconds it took to finish the last N processing runs. The array MUST NOT contain run times for more than the 5 last runs.
max	**int**	This value specifies the maximum number of seconds that it took to finish a run. If no analysis runs are complete, this value MUST be 2147483648.
min	**int**	This value specifies the minimum number of seconds that it took to finish a run. If no analysis runs finished, this value MUST be 2147483647.
sum	**int**	This value specifies the total amount of time that was spent on processing search clickthrough logs.

2.2.3.5 systemstatus Array

The **systemstatus** array is contained in the status structure, as specified in section 2.2.2. The following table specifies the content of the **systemstatus** array.

Element	Type	Description
0	**string**	If the protocol server is starting for the first time after installation finished, this value MUST be "Bootstrap". Otherwise, this value MUST be "Running".
1	**string**	If one or more worker processes do not respond, this value MUST be "WorkerFailure". If another error occurs, this value MUST be "WAError". In all other cases, this value MUST be "NoError".
2	**string**	If Element 1 contains a value of "NoError", this value MUST be "No error". Otherwise, this value MUST be a string that describes the error.

2.2.4 Analysis Stages

The analysis that the protocol server performs MUST be divided into several stages. The following table specifies the name and description of each stage of the analysis.

Stage name	Description
spcidcollapser	The protocol server collects any new **document identifiers (3)** that are associated with each crawl collection and updates the list of current document identifiers (3).
spanalysis	The protocol server generates a set of associated queries and weights them for each document identifier (3).
db_switch	The protocol server deploys the output databases.
sppartialupdate	The protocol server updates the search index.
cleanup	After the analysis finishes, the protocol server performs cleanup tasks.
nothing	The protocol server is not performing analysis. This value is a placeholder that MUST be used only when no analysis is underway.

2.2.5 Crawl Collection Status

The crawl collection status MUST be specified as a bitmask from the following table.

Value	Description
0x00000000	The crawl collection is ready.
0x00000001	The crawl collection is being updated.
0x00000002	The crawl collection is to be deleted.
0x00000004	The crawl collection is being deleted.
0x00000008	The crawl collection is to be cleared.
0x00000016	The crawl collection is being cleared.

2.2.6 URL Relevance Structure

The following table specifies the content of the structure that is returned by the **GetURIRelevanceData** method.

Member name	Type	Description
contentid	**string**	This value MUST be the document identifier (3) for the document that is described by this URL relevance structure.
queries	**Array**	This array of arrays contains all the associated queries for the document identifier (3). For more details, see section 2.2.6.1.

2.2.6.1 queries Array

The **queries** array, as specified in section 2.2.6, MUST relate to a single document identifier (3) and MUST comprise one or more arrays that MUST each conform to the format specified in the following table.

Element	Type	Description
0	**string**	The number of times that this query was associated with this document identifier (3).
1	**string**	The number of times that this query was associated with any document identifier (3).
2	**string**	A value that specifies how well the associated query describes this document identifier (3).
3	**string**	This value is the number of different document identifier (3) with which the query was associated.
4	**string**	The associated query.

2.2.7 Schedule array

The following table specifies the elements that a schedule array MUST contain.

Element	Type	Description
0	**int**	The time of the next processing run. This value MUST be of type **int**. This value MUST specify time, in seconds, relative to 00:00:00 1970-01-01 **UTC**.
1	**Int**	The interval, in seconds, between one processing run and the next. The value MUST be of type **int**.

2.2.8 Log levels

Log level codes MUST be specified as in the following table.

Log level name	Description
error	ERROR and CRITICAL log messages.
warning	The preceding level, in addition to WARNING messages.
info	The preceding level, in addition to INFO messages.
verbose	The preceding level, in addition to VERBOSE messages.
debug	The preceding level, in addition to DEBUG messages.

2.2.9 Error handling

The XML-RPC protocol supports a special message reply known as a fault. A fault is used to report errors back to the protocol client. Each fault MUST contain a fault code and a fault string, as specified in [XML-RPC].

Most of the errors that occur in the methods which are described in this specification generate faults. When a fault is generated, it MUST replace the return value of the method. This means that the return values that are specified in the following sections apply only to successful calls; every method MUST return a fault if the call is unsuccessful.

For this protocol, the fault code MUST be of type **int** and have a value of 1. The fault string MUST be specified according to the following Augmented Backus-Naur Form (ABNF) rules:

```
errormsg = prefix type delim errortxt

delim = %d39.38.103.116.59.58.32.32.60
prefix = %d38.108.116.59.116.121.112.101.32.39

type = %d101.120.99.101.112.116.105.111.110.46 (exception / attributeerror) %d46

exception = %d69.120.99.101.112.116.105.111.110
attributeerror = %d65.116.116.114.105.98.117.116.101.69.114.114.111.114
errortxt = 1*(VCHAR / SP)
```

exception: This value MUST be part of the fault message if the fault happened within the scope of the method.

attributeerror: This MUST be part of the fault message if an unknown method is called.

3 Protocol Details

3.1 Protocol Server Details

The protocol server MUST listen for incoming connections, process incoming XML-RPC requests, and respond to those requests in a timely manner. If an unexpected error occurs during processing, an XML-RPC fault MUST be generated, as specified in [XML-RPC].

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

The protocol server MUST NOT initially have any configured crawl collections.

The protocol client MUST start its XML-RPC server implementation as soon as it is able to process incoming requests.

The protocol server MUST register with the Configuration Service as specified in [MS-FSCX], and implement the following methods that are required by that protocol: **ConfigurationChanged**, **ReRegister**, and **ping**. When the protocol server registers itself, it MUST specify "SPRel" for both module type and module name, and the alerts array MUST contain the string "collection".

3.1.4 Message Processing Events and Sequencing Rules

ConfigurationChanged: When the **ConfigurationChanged** method is called with an alert type of "collection", the protocol client MUST update its crawl collection state. Crawl collections that are no longer valid MUST be deleted, and new crawl collections MUST be created.

3.1.4.1 Processing Management Methods

Every method that is specified in this section MUST have all of its arguments specified.

3.1.4.1.1 PauseProcessing

The **PauseProcessing** method pauses the current analysis. This method MUST return a fault if there is no analysis run underway.

```
int PauseProcessing()
```

Return value	Description
1	This method MUST return a value of 1 if the analysis run is paused.
2	This method MUST return a value of 2 if the analysis run is not paused but will pause eventually.

3.1.4.1.2 StartProcessing

The **StartProcessing** method schedules an analysis run. Depending on the argument to the Now parameter, a processing run MUST either begin immediately, or the processing status MUST be set to ready. Setting the processing status to ready ensures that the analysis run can begin at the scheduled time, if new data is detected within the system.

```
int StartProcessing(int Now)
```

Now: This value MUST be set to 0 or 1. If this value is 1, a processing run MUST begin immediately. If this value is 0, the processing status MUST be set to ready.

Return value	Description
1	This method MUST return a value of 1.

3.1.4.1.3 StopProcessing

The **StopProcessing** method sets the protocol server to a stopped mode. Depending on the arguments that are provided, any ongoing processing MUST be aborted, or the processing MUST finish before the status is set to stopped.

```
int StopProcessing(int Now)
```

Now: Specifies whether the analysis should be halted before its status is set to stopped. This value MUST be either 0 or 1. If this value is 1, any analysis that is underway MUST be halted. Otherwise, if this value is 0, that analysis MUST be allowed to finish.

Return value	Description
1	This method MUST return 1 if the processing was halted, if the protocol server was not running any processing, or if the **now** option was 0.
2	This method MUST return 2 if the **now** option was 1 but the processing could not be immediately halted. This does not mean that the processing continues to run until it finishes, only that it is halted later.

3.1.4.2 Configuration Methods

Every method that is specified in this section MUST have all of its arguments specified.

3.1.4.2.1 GetConfig

The **GetConfig** method returns all the global configuration options that are being used by the protocol server.

```
struct GetConfig()
```

Return value	Description
A structure that contains all the TEST	This method MUST return a structure that contains all the global

Return value	Description
global configuration options.	configuration options, as specified in section 2.2.1.1.

3.1.4.2.2 GetFDMConfig

The **GetFDMConfig** method returns all the processing-centric configuration options that are being used by the protocol server.

```
struct GetFDMConfig()
```

Return value	Description
A structure that contains all the processing-centric configuration options.	This method MUST return a structure that contains all the processing-centric configuration options, as specified in section 2.2.1.2.

3.1.4.2.3 GetLogLevel

The **GetLogLevel** method queries the protocol server for its current log verbosity level.

```
string GetLogLevel()
```

Return value	Description
The log level.	This method MUST return a mask of one or more log levels, as specified in section 2.2.8.

3.1.4.2.4 GetSchedule

The **GetSchedule** method queries the protocol server for the processing schedule.

```
array GetSchedule()
```

Return value	Description
An array containing the analysis schedule.	This method MUST return a schedule array, as specified in section 2.2.7.

3.1.4.2.5 SetConfig

The **SetConfig** method changes a specific global configuration value.

```
int SetConfig(string Keyword, dynamic Value)
```

Keyword: The name of a configuration option, as specified in section 2.2.1.1.

Value: A new value for the configuration option, as specified in section 2.2.1.1.

Return value: This method MUST return a value of 1.

3.1.4.2.6 SetFDMConfig

The **SetFDMConfig** method changes a specific FDM configuration value.

```
int SetFDMConfig(string Keyword, dynamic Value)
```

Keyword: The name of a configuration option, as specified in section 2.2.1.2.

Value: A new value for the configuration option, as specified in section 2.2.1.2.

Return value: This method MUST return a value of 1.

3.1.4.2.7 SetLogLevel

The **SetLogLevel** method sets the log verbosity level for the protocol server. As a result of this call, the protocol server MUST alter its log verbosity level to the specified level.

```
int SetLogLevel(string Level)
```

Level: A valid log level to be set on the protocol server. For more information, see section 2.2.8.

Return value: This method MUST return a value of 1.

3.1.4.2.8 SetSchedule

The **SetSchedule** method changes the processing schedule of the protocol server.

```
int SetSchedule(int When, int Interval)
```

When: Specifies the time of the next analysis run. The value MUST be an **int**. This method MUST specify the time of the processing run in seconds, relative to 00:00:00 1970-01-01 UTC.

Interval: Specifies the interval to elapse between analysis runs, in seconds. This value MUST be an **int**.

Return value: This method MUST return a value of 1.

3.1.4.3 Status Methods

Every method that is specified in this section MUST have all of its arguments specified.

3.1.4.3.1 GetStatus

The **GetStatus** method queries the protocol server for global status information.

```
struct GetStatus()
```

Return value	Description
A structure that contains global status information.	This method MUST return a structure that contains global status information, as specified in section 2.2.2.

3.1.4.3.2 GetURIRelevanceData

The **GetURIRelevanceData** method returns relevance data for a specific URL.

```
struct GetURIRelevanceData(string ID)
```

ID: The identifier of the document for which relevance data is being requested. This value MUST specify the identifier as a document identifier (3).

Return value	Description
A structure that contains URL relevance information.	This method returns a structure that contains URL relevance information, as specified in section 2.2.6.
An empty structure.	If the protocol server is not done with the analysis or if the document identifier is not found, this method MUST return an empty structure.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 Protocol Client Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

This protocol requires the setup of a TCP connection between the protocol client and the protocol server. The port number used for the connection MUST be **base port** plus 305. The connection MUST be initiated by the protocol client.

3.2.4 Message Processing Events and Sequencing Rules

None.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

4 Protocol Examples

The examples in this section contain only the XML body for each XML-RPC message.

4.1 GetSchedule

In this example, the protocol client uses the **GetSchedule** method to retrieve the current processing schedule. The protocol server responds with a schedule which indicates that the analysis runs every day at 04:00 and that the next scheduled run is at 2011-04-19T04:00:00 (Tue Apr 19 04:00:00 2011).

Request

```
<?xml version='1.0'?>
<methodCall>
    <methodName>GetSchedule</methodName>
        <params>
        </params>
</methodCall>
```

Response

```
<?xml version='1.0'?>
<methodResponse>
    <params>
        <param>
            <value><array><data>
                <value><int>1303182000</int></value>
                <value><int>86400</int></value>
            </data></array></value>
        </param>
    </params>
</methodResponse>
```

4.2 StopProcessing

In this example, the protocol client uses the **StopProcessing** method to instruct the protocol server to set the processing status to stopped and to discontinue any processing that is underway. To cause the protocol server to immediately discontinue processing, the protocol client calls the **StopProcessing** method with the "now" flag set to 1 to indicate a value of **true**.

Request

```
<?xml version='1.0'?>
<methodCall>
    <methodName>StopProcessing</methodName>
    <params>
        <param>
            <value><int>1</int></value>
        </param>
    </params>
</methodCall>
```

Response

```
<?xml version='1.0'?>
<methodResponse>
    <params>
        <param>
            <value><int>1</int></value>
        </param>
    </params>
</methodResponse>
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: XML Schema

For ease of implementation, the following XML-RPC Schema is provided.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="methodCall">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="methodName">
          <xsd:simpleType>
            <xsd:restriction base="ASCIIString">
              <xsd:pattern value="([A-Za-z0-9]|/|\.|:|_)*" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="params" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="param"  type="ParamType"
                           minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
          </xsd:complexType>
         </xsd:element>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="methodResponse">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name="params">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="param" type="ParamType" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="fault">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="value">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="struct">
                      <xsd:complexType>
                        <xsd:sequence>
                          <xsd:element name="member"
                                       type="MemberType" />
                          <xsd:element name="member"
                                       type="MemberType" />
                        </xsd:sequence>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
```

```
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="ParamType">
  <xsd:sequence>
    <xsd:element name="value" type="ValueType" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ValueType" mixed="true">
  <xsd:choice>
    <xsd:element name="i4"            type="xsd:int" />
    <xsd:element name="int"           type="xsd:int" />
    <xsd:element name="string"        type="ASCIIString" />
    <xsd:element name="double"        type="xsd:decimal" />
    <xsd:element name="Base64"        type="xsd:base64Binary" />
    <xsd:element name="boolean"       type="NumericBoolean" />
    <xsd:element name="dateTime.iso8601" type="xsd:dateTime" />
    <xsd:element name="array"         type="ArrayType" />
    <xsd:element name="struct"        type="StructType" />
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="StructType">
  <xsd:sequence>
    <xsd:element name="member" type="MemberType"
                 maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MemberType">
  <xsd:sequence>
    <xsd:element name="name"  type="xsd:string" />
    <xsd:element name="value" type="ValueType" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ArrayType">
  <xsd:sequence>
    <xsd:element name="data">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="value"  type="ValueType"
                       minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="ASCIIString">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="([ -~]|\n|\r|\t)*" />
  </xsd:restriction>
</xsd:simpleType>
```

```
    <xsd:simpleType name="NumericBoolean">
      <xsd:restriction base="xsd:boolean">
        <xsd:pattern value="0|1" />
      </xsd:restriction>
    </xsd:simpleType>

  </xsd:schema>
```

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

N

O

P

Q

R

S

T

U

V

X