[MS-FSCX]:
Configuration XML-RPC Protocol Specification

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	1.05	Minor	Clarified the meaning of the technical content.
03/18/2011	1.05	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.05	No change	No changes to the meaning, language, or formatting of the technical content.
01/20/2012	1.6	Minor	Clarified the meaning of the technical content.
04/11/2012	1.6	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	1.6	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Configuration XML-RPC Protocol between a protocol server, which is the Configuration Component, and protocol clients. A protocol server stores configuration and system information, and allows protocol clients to update or query this information. Protocol clients can register for alerts to receive notifications about relevant changes. The protocol server also regularly contacts components in the system to assure their availability.

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-OFCGLOS]:

backup indexer node
base64 encoding
content collection
content source
index column
indexer row
indexing component
master indexer node

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-FSCMW] Microsoft Corporation, "Configuration Middleware Protocol Specification".

[MS-FSO] Microsoft Corporation, "FAST Search System Overview".

[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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, http://www.ietf.org/rfc/rfc4648.txt

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

1.2.2 Informative References

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

1.3 Protocol Overview (Synopsis)

This protocol specifies the communication between the Configuration protocol server and other protocol entities. Protocol entities that implement a set of methods that the Configuration protocol server can call are referred to as modules and can in some cases also act as servers. The protocol server performs the following:

- Manages configuration, including uploading and downloading configuration information. There are two types of configuration. Universal configuration is divided into sections and is shared between all protocol clients. Configuration files are associated with a module and a path, and generate alerts when they are updated.
- Manages modules, including registering and unregistering modules, and querying for existing modules.
- Updates and queries a model of the system. In addition to the registered modules, this system model also keeps track of **content collections** and **indexing components**.
- Manages alerts and sends notification to modules when changes occur, if the modules subscribed to those alerts when they registered.

The following figure is an example of network traffic between the protocol server and protocol entities.

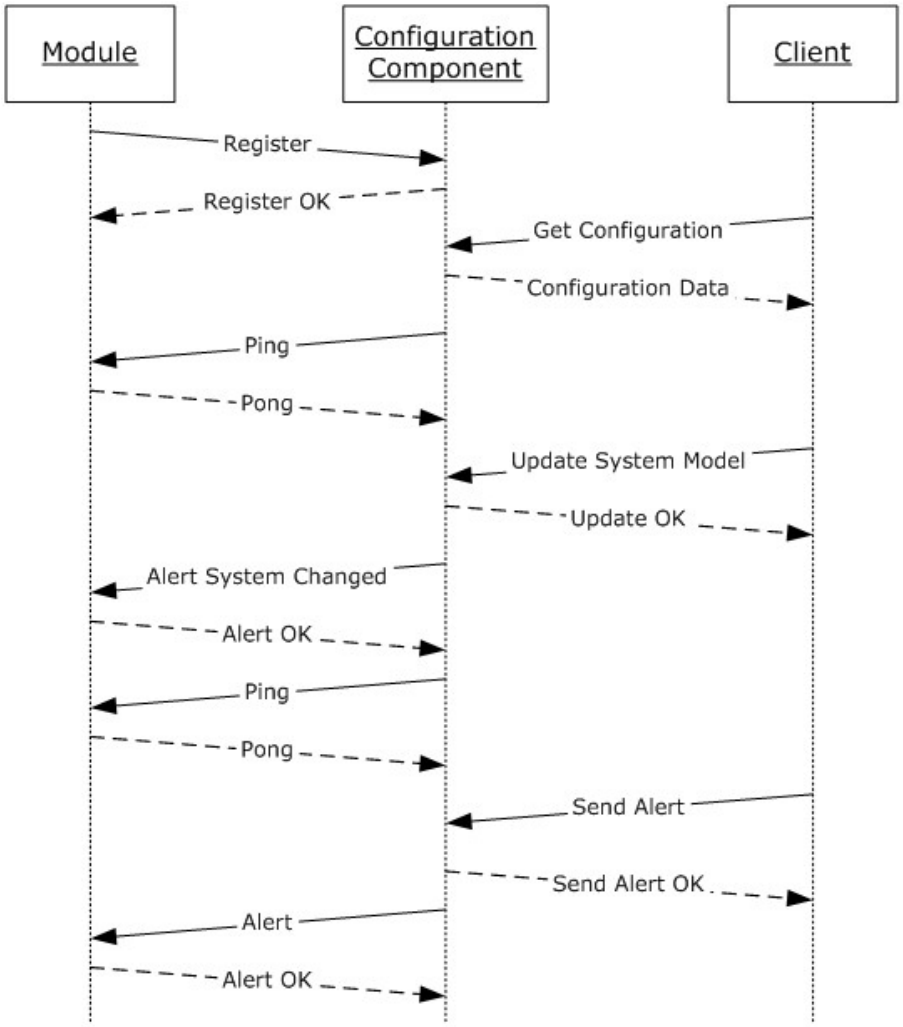

Figure 1: Network activity

For more information about the abstract data model, see section 3.

1.4 Relationship to Other Protocols

The Configuration XML-RPC Protocol uses XML-RPC over HTTP as described in [XML-RPC].

The following diagram shows the underlying messaging and transport stack that the protocol uses:

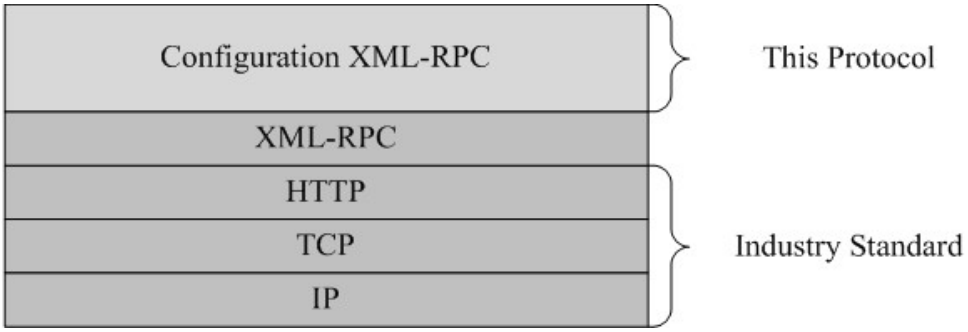

Figure 2: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

To use this protocol, the protocol client establishes a connection to the protocol server over TCP/IP. The host name and port associated with the protocol server are known to the protocol client prior to initiating the connection.

1.6 Applicability Statement

This protocol is used in a distributed system in which protocol clients share small configuration files. Protocol clients can register to receive notifications about changes or to be visible to other protocol clients.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The protocol MUST use HTTP as specified in [RFC2616] for the transport mechanism over TCP/IP.

2.2 Message Syntax

The format of the XML body requests and responses is described in [MS-FSXTAPI]. The HTTP POST path, as specified in [RFC2616], MUST contain the value "RPC2".

2.2.1 Common Data Types

In addition to the basic int, double, string, array and struct data types described in [XML-RPC], this protocol uses the data structures specified in the following table. Each of these data structures is either a basic data type with additional restrictions, or the specification of a composite data type.

Name	Description
CollectionName	The name of a content collection. For details see section 2.2.1.1.
ModuleAddress	An array that contains the location of a module. For details see section 2.2.1.2.
ModuleRegister	Information used to register a module. For details see section 2.2.1.3.
AlertType	The type of alert to which a protocol client subscribes. For details see section 2.2.1.4.
ModuleInfo	Information about a module. For details see section 2.2.1.5.
AllModuleInfo	Module information for all modules in the system. For details see section 2.2.1.6.
ModuleType	The type of module to query on the protocol server. For details see section 2.2.1.8.
CollectionDetails	Attributes of a collection. For details see section 2.2.1.8.
UniversalConfig	Configuration information. For details see section 2.2.1.9.

2.2.1.1 CollectionName Data Type

The name field associated with a content collection MUST contain a string that contains only the characters in [a-zA-Z0-9] and contains a maximum of 16 characters. The lower-case version of the *CollectionName* field MUST be unique.

2.2.1.2 ModuleAddress Data Type

This MUST contain the address of the module. It is represented as an array whose values are specified in the following table.

Index	Type	Description
0	string	The host name where the module is located
1	int	The port number used by the protocol server to contact the module

2.2.1.3 ModuleRegister Data Type

This contains information about modules that register with the protocol server. It is represented as a structure which contains all of the elements specified in the following table. Each module can specify additional elements.

Name	Type	Description
port	integer	The port number of the module.
type	string	The type of the module, as specified in section 2.2.1.6
name	string	The name of the module
version	string	The version of the module
alerts	array of strings	The alerts to which the module subscribed

2.2.1.4 Alert Type

An alert MUST be a known type, as specified in the following table.

Alert types
collection
configfile
cluster_assigned
clusters
index
pipeline_added
pipeline_modified
pipeline_removed
pipelines
processorserver_added
processorserver_removed
search_mode_change
stages

2.2.1.5 ModuleInfo Data Type

This contains all of the elements of the *ModuleRegister* structure, except for the *port* and *type* elements. It also contains a *middleware* element whose value MUST be set to "xml-rpc" for all modules that registered using this protocol. If the module uses the Configuration Middleware Protocol as specified in [MS-FSCMW], then the value of the *middleware* element MUST be set to "mw". The additional elements specified in the *ModuleRegister* structure MUST also be present in the corresponding *ModuleInfo* structure.

Name	Type	Description
name	string	The name of the module
version	string	The version of the module
alerts	array of strings	The alerts to which the module subscribed
middleware	string	This MUST be either "xml-rpc" or "mw"

2.2.1.6 AllModuleInfo Data Type

This contains the information about the modules that registered with the protocol server. It is represented as a structure in which the module type is used as the name of the associated element. The array of structures specified in the following table contains the values for each element. All elements in the table MUST be present in each *AllModuleInfo* structure.

Name	Type	Description
name	string	The name of the module
version	string or int	The version of the module
hostname	string	The host name where the module is located
port	int	The port through which to contact the module
middleware	string	This MUST be either "xml-rpc" or "mw"
responding	int	If the module responded to the last **ping** method request from the protocol server, this contains the value 1, otherwise 0.

2.2.1.7 Module Type

The following module types MUST always be available for query on the protocol server, even if no modules are currently registered for the module type. In that case, the value for that type MUST be an empty array.

ModuleType
ContentDistributor
ProcessorServer
DataSource
FilterInterface
FilterAlerter
Search Engine

2.2.1.8 CollectionDetails Data Type

This contains the details about a content collection. It is represented as a structure which contains the elements specified in the following table. All elements in the table MUST be present in the structure.

Name	Type	Description
name	string	The name of the content collection
cluster	string	This MUST be "webcluster"
description	string	A description of the content collection
pipeline	string	This MUST be "Office14 (webcluster)"
datasources	array of *ModuleAddress* structures (section 2.2.1.2)	This is a list of **content sources** associated with the content collection. The array MUST contain 0 or more elements.
created	string	The date the collection was created
cleared	string	The date that the content in the collection was last cleared

2.2.1.9 UniversalConfig Data Type

This contains configuration information for one section in the universal configuration file. The section specifies an *attrib* field that contains an array of name-value pairs. The *name* attribute of an attrib is a string. The value of an attribute is of type string, int, double, array or struct. If the value is an array, each value in the array is of type string, int or double. If the value is a structure, each member has a name and values that are of type string, int or double.

```
struct UniversalConfig {
  array attrib;
}

struct attrib {
  attribute string name;
  attribute string|int|double value;
}
```

2.2.2 Fault Response

The XML-RPC protocol supports a special message, known as a fault response, to report errors to the protocol client. The fault contains a fault code and a fault string as described in [XML-RPC].

In this protocol the fault code MUST be an integer with the value 1.

The fault string MUST use the following format.

```
&lt;class '<ExceptionType>'&gt;: <ErrorText>
```

The <ErrorText> element contains a textual description of the error, and the <ExceptionType> element MUST contain a value specified in the following table.

Value	Description
ConfigServerException	The base class used for high level faults
ConfigServerException.GeneralError	A general error that does not fall into any of the following specific error categories
ConfigServerException.PipelineError	An error related to the *pipelines* alert type, as specified in section 2.2.1.3.
ConfigServerException.CollectionError	An error related to a content collection
ConfigServerException.ConfigError	An error related to configuration of the system
ConfigServerException.SearchClusterError	An error related to a search *cluster* element
ConfigServerException.ModuleError	An error related to a module
ConfigServerException.DataSourceError	An error related to a content source

2.2.3 AddCollection

This creates a new content collection. The signature of this method is as follows.

```
int AddCollection(string collection, string description, string cluster, string pipeline);
```

collection: The name of the content collection. It is a *CollectionName* string as specified in section 2.2.1.1.

description: This specifies what the content collection can contain.

cluster: Contains the value "webcluster".

pipeline: Contains the value **"**Office14 (webcluster)".

return value: Contains a value of 1.

2.2.4 ConfigurationChanged

This sends an alert to a module. The signature of this method is as follows.

```
int ConfigurationChanged(string alerttype, alertarg);
```

alerttype: This is the type of alert, as specified in section 2.2.1.3.

alertarg: These are additional arguments that are associated with the alert. The *alertarg* parameter can be of any data type.

return value: MUST contain a value of 1.

2.2.5 GetActiveModuleList

This returns a list of all active modules of a specified type. The signature of this method is specified as follows.

```
array GetActiveModuleList(string moduletype);
```

moduletype: This is the module type.

return value: MUST return an array in which each element is a *ModuleAddress* array as specified in section 2.2.1.2.

2.2.6 GetAllModuleDetails

This queries the protocol server for information about all known modules. The signature of this method is as follows.

```
struct GetAllModuleDetails();
```

return value: MUST return an *AllModuleInfo* structure as specified in section 2.2.1.6.

2.2.7 GetClusterCollections

This queries the protocol server for a list of all content collections. The signature of this method is as follows.

```
array GetClusterCollections(string cluster);
```

cluster: Contains the value "webcluster".

return value: MUST return an array in which each element is a *CollectionName* string as specified in section 2.2.1.1.

2.2.8 GetClusters

Returns an array that contains a string whose value is "webcluster". The signature of this method is as follows.

```
array GetClusters();
```

return value: MUST contain an array of one string, whose value MUST be "webcluster".

2.2.9 GetClusterSearchColumns

This queries the protocol server for a list of all **master indexer nodes**. The signature of this method is as follows.

```
array GetClusterSearchColumns(string cluster, int docapi=0, int static=1);
```

cluster: This MUST be "webcluster".

docapi: Optional. If specified, the value MUST be 0. If omitted, the parameter MUST be processed as though it contained a value of 0.

static: Optional. If specified, the value MUST be 1. If omitted, the parameter MUST be processed as though it contained a value of 1.

return value: MUST return an array of *ModuleAddress* arrays as specified in section 2.2.1.2.

2.2.10 GetCollection

This queries the protocol server for the attributes of a content collection. The signature of this method is as follows.

```
struct GetCollection(string collection);
```

collection: The name of the content collection. This MUST be a *CollectionName* string as specified in section 2.2.1.1.

return value: MUST return a *CollectionDetails* structure as specified in section 2.2.1.8 or an empty structure.

2.2.11 GetCollectionCleared

This returns a string that represents the date the content collection was last cleared of content. The signature of this method is as follows.

```
string GetCollectionCleared(string collection);
```

collection: A **String** that contains the name of a content collection.

return value: MUST return a string that represents a date formatted as specified in [ISO-8601] or an empty string.

2.2.12 GetCollectionCluster

This returns a string that contains either the value "webcluster" or the empty string. The signature of this method is as follows.

```
string GetCollectionCluster(string collection);
```

collection: The name of the content collection. This MUST be a *CollectionName* string as specified in section 2.2.1.1.

return value: MUST return the value "webcluster" or an empty string.

2.2.13 GetCollectionCreated

This returns a string that represents the date the content collection was created. The signature of this method is as follows.

```
string GetCollectionCreated(string collection);
```

collection: A string that contains the name of the content collection.

return value: MUST return a string that represents a date formatted as specified in [ISO-8601] or an empty string.

2.2.14 GetCollectionDataSourceList

This queries the protocol server for which content sources are associated with a content collection. The signature of this method is as follows.

```
array GetCollectionDataSourceList(string collection);
```

collection: The name of the content collection. It MUST be a *CollectionName* string as specified in section 2.2.1.1.

return value: MUST return an array with 0 or more elements. Each element in the array is a *ModuleAddress* array as specified in section 2.2.1.2.

2.2.15 GetCollectionList

This queries the protocol server for all content collections. The signature of this method is as follows.

```
array GetCollectionList();
```

return value: MUST return an array in which each item is a *CollectionName* string as specified in section 2.2.1.1.

2.2.16 GetCollectionPipeline

This returns a string that contains the value "Office14 (webcluster)". The signature of this method is as follows.

```
string GetCollectionPipeline(string collection);
```

collection: The name of the content collection. It MUST be a *CollectionName* string as specified in section 2.2.1.1.

return value: MUST return the value "Office14 (webcluster)".

2.2.17 GetConfig

This queries the protocol server for a section of the universal configuration file. The signature of this method is as follows.

```
UniversalConfig GetConfig(string section);
```

section: The name of a section from the universal configuration file as specified in section 2.2.1.9.

return value: A *UniversalConfig* structure.

2.2.18 GetIndexerRowID

This queries the protocol server to determine which **indexer row** an indexing component belongs to. The signature of this method is as follows.

```
int GetIndexerRowID(string hostname, int port);
```

hostname: This is the host name on which the indexing component runs.

port: This is the port through which the indexing component is available.

return value: An integer that specifies the row that contains the specified indexing component. The value is greater than or equal to 0.

2.2.19 GetModuleInfo

This queries the protocol server for information about a module. The signature of this method is as follows.

```
struct GetModuleInfo(string hostname, int port, string module="");
```

hostname: This is the host name of the computer that runs the module.

port: This is the port through which the module is available.

module: Optional. If specified, this MUST be an empty string.

return value: MUST return a *ModuleInfo* structure as specified in section 2.2.1.5.

2.2.20 GetModuleList

This queries the protocol server for modules of a specified type. The signature of this method is as follows.

```
array GetModuleList(string moduletype);
```

moduletype: This is the type of module.

return value: MUST return an array with 0 or more elements in which each element is a *ModuleAddress* array as specified in section 2.2.1.2.

2.2.21 GetSearchBackupList

This queries the protocol server to determine which **backup indexer nodes** are associated with a master indexer node. The signature of this method is as follows.

```
List GetSearchBackupList(string host, int port, int docapi=0);
```

host: This is a string that contains the host name of the master indexer node.

port: This is an integer that represents the port that is associated with the master indexer node.

docapi: Optional. If specified, the parameter MUST contain a value of 0.

return value: MUST return a list that contains zero or more elements in which each element is a *ModuleAddress* array as specified in section 2.2.1.2.

2.2.22 GetSearchColumnCluster

This returns a string that contains "webcluster" if the specified indexing component exists. The signature of this method is as follows.

```
string GetSearchColumnCluster(string host, int port);
```

host: This is the host name on which the indexing component runs.

port: The port number through which the indexing component is available.

return value: MUST return a string with the value "webcluster" or an empty string.

2.2.23 GetSearchColumnFTMode

This queries the protocol server to determine whether an indexing component is in an **index column** that is running in fault-tolerant mode. The signature of this method is as follows.

```
int GetSearchColumnFTMode(string host, int port);
```

host: This is the host name on which the indexing component runs.

port: This is the port number that is associated with an indexing component.

return value: MUST return 0 or 1.

2.2.24 GetSearchColumnPartitionID

This returns the column number that is associated with an indexing component. The signature of this method is as follows.

```
int GetSearchColumnPartitionID(string host, int port);
```

host: This is a string that contains the host name on which the indexing component runs.

port: This is an integer that represents the port number that is associated with an indexing component.

return value: An integer that specifies a column number. Its value is greater than or equal to 0.

2.2.25 IsValidCollection

This verifies whether a content collection exists. The signature of this method is as follows.

```
int IsValidCollection(string collection);
```

collection: This is the name of the content collection. This MUST be a *CollectionName* string as specified in section 2.2.1.1.

return value: MUST return 0 or 1.

2.2.26 LoadConfigFile

This retrieves a configuration file from the protocol server. The signature of this method is as follows.

```
string LoadConfigFile(string module, string filepath);
```

module: This is the name of the module for which this configuration file is stored

filepath: This is the relative path to the configuration file.

return value: The content of the configuration file.

2.2.27 LoadConfigFileBase64

This retrieves a configuration file from the protocol server. The information the file is encoded using **base64 encoding** as specified in [RFC4648]. The signature of this method is as follows.

```
string LoadConfigFileBase64(string module, string filepath);
```

module: This is the name of the module for which this configuration file is stored.

filepath: This is the relative path to the configuration file.

return value: The content of the configuration file encoded using base64 encoding as specified in [RFC4648].

2.2.28 LoadOptionalConfigFile

This retrieves an optional configuration file from the protocol server. The signature of this method is as follows.

```
string LoadOptionalConfigFile(string module, string filepath);
```

module: This is the name of the module for which this configuration file is stored.

filepath: This is the relative path to the configuration file.

return value: The content of the configuration file or an empty string.

2.2.29 LoadOptionalConfigFileBase64

This retrieves an optional configuration file from the protocol server. The information in the file returned is encoded using base64 encoding as specified in [RFC4648]. The signature of this method is as follows.

```
string LoadOptionalConfigFileBase64(string module, string filepath);
```

module: This is the name of the module for which this configuration file is stored.

filepath: This is the relative path to the configuration file.

return value: The content of the configuration file encoded using base64 encoding as specified in [RFC4648] or an empty string.

2.2.30 ping

This specifies whether registered modules are available. The signature of this method is as follows.

```
string ping();
```

return value: MUST return the value "pong".

2.2.31 RegisterModule

This registers a module with the protocol server. The signature of this method is as follows.

```
string RegisterModule(struct moduleinfo);
```

moduleinfo: This MUST be a *ModuleRegister* structure as specified in section 2.2.1.3.

return value: This is the host name from which the module registered.

2.2.32 RegisterProcessorPipelines

This specifies that a document processing component is ready to process content. The signature of this method is as follows.

```
RegisterProcessorPipelines(array processorserver, array pipelines);
```

processorserver: The document processing component. This MUST be a *ModuleAddress* array as specified in section 2.2.1.2.

pipelines: This MUST be an array that contains the string "Office14 (webcluster)".

return value: MUST return 1.

2.2.33 RemoveCollection

This removes a content collection. The signature of this method is as follows.

```
int RemoveCollection(string collection, int force=0);
```

collection: This is the name of the content collection. This MUST be a *CollectionName* string as specified in section 2.2.1.1.

force: Optional. If specified, this MUST have the value 0 or 1.

return value: MUST return 1.

2.2.34 ReRegister

This requests that a registered module resubmit its module information. The signature of this method is as follows.

```
struct ReRegister();
```

return value: MUST return a *ModuleRegister* structure as specified in section 2.2.1.3

2.2.35 SaveConfigFile

This uploads a configuration file to the protocol server. The signature of this method is as follows.

```
int SaveConfigFile(string module, string filepath, string data, int sendAlert=1);
```

module: This is the name of the module for which this configuration file is stored.

filepath: This is the relative path to the configuration file.

data: This is the content to store in the configuration file.

sendAlert: Optional. If specified, this MUST have the value 0 or 1.

return value: MUST return 1.

2.2.36 SaveConfigFileBase64

This uploads a configuration file using base64 encoding to the protocol server. The signature of this method is as follows.

```
int SaveConfigFileBase64(string module, string filepath, string data, int sendAlert=1);
```

module: This is the name of the module for which this configuration file is stored.

filepath: This is the relative path to the configuration file.

data: This is the content to store in the configuration file using base64 encoding as specified in [RFC4648].

sendAlert: Optional. If specified, this MUST have the value 0 or 1.

return value: MUST return 1.

2.2.37 SendAlert

This sends an alert of the specified type to all modules that requested notification about this type of alert. The signature of this method is as follows.

```
int SendAlert(string alerttype, alertarg=None);
```

alerttype: This is the type of alert. See section 2.2.1.3 for known alert types.

alertarg: Optional. The protocol server MUST accommodate an *alertarg* parameter of any data type and any value.

return value: MUST return 1.

2.2.38 SetConfig

This sets a configuration section in the universal configuration. The signature of this method is as follows.

```
int SetConfig(string section, struct config);
```

section: A **String** that contains the name of the configuration section to set in the universal configuration file, as specified in section 2.2.1.9.

config: The configuration to store in this section. This MUST be a *UniversalConfig* structure as specified in section 2.2.1.9.

return value: MUST return 1.

2.2.39 UnregisterModule

This unregisters a module from the protocol server. The signature of this method is as follows.

```
int UnregisterModule(struct moduleinfo);
```

moduleinfo: This MUST be a *ModuleRegister* structure as specified in section 2.2.1.3.

return value: MUST return 1.

2.2.40 UpdateCollection

This creates or updates a content collection. The signature of this method is as follows.

```
int UpdateCollection(string collection, string description, string cluster, string pipeline,
string cleared, array datasources=None);
```

collection: This is the name of the content collection. This MUST be a *CollectionName* string as specified in section 2.2.1.1.

description: This is a description of the content collection.

cluster: Contains the value "webcluster".

pipeline: Contains the value "Office14 (webcluster)".

cleared: This is a string that contains the formatted date, as specified in [ISO-8601], of the last content clearance.

datasources: Optional. If specified, this MUST be a *ModuleAddress* array as specified in section 2.2.1.2.

return value: MUST return 1.

3 Protocol Details

3.1 Configuration Component 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.

3.1.1.1 Universal Configuration

This contains configuration information that is grouped in sections, rather than associated with a module and a local path. The protocol server MUST store this information in the following data structure.

universal configuration: This is a set of section names associated with a data structure. The contents of the data structure MUST NOT contain arrays or structures inside arrays or structures. The valid configuration types are specified in section 2.2.1.9.

3.1.1.2 Content Collections

The protocol server MUST maintain information about content collections in the following data structure.

content collections**:** This is a list of content collections. For each content collection the protocol server MUST store the associated name, description, date of creation, and date of last content clearance, and list of content sources.

3.1.1.3 Modules

The protocol server MUST maintain both the modules that register with the **RegisterModule** method specified in section 2.2.31, and the modules that are tracked as part of the Configuration Middleware Protocol as specified in [MS-FSCMW]. Modules are maintained in two separate lists based on the state the protocol server maintains:

active modules: These are modules that respond to the **ping** request specified in section 2.2.30 and section 3.1.5.2.

inactive modules: These are modules that did not respond to the ping request specified in section 2.2.30 and section 3.1.5.2. They are maintained in a list for a specified time before the protocol server deletes them from the list.

Each module contains the information for the *ModuleInfo* structure, as specified in section 2.2.1.5.

The protocol server sends a **ping** request to determine whether or not modules are still available. It MUST maintain a list of all the pending **ping** requests.

ping requests: This is the list of **ping** requests that have not received a reply.

3.1.1.4 Indexing components

The indexing service is specified in [MS-FSO]. A set of modules act as indexing components that are arranged in rows and columns so that the protocol can scale to the appropriate size and support fault-tolerance. Each column contains a master indexer node and zero or more backup indexer nodes.

The protocol server MUST maintain the following data structure:

indexers: This is a list of indexing components. For each indexing component, the protocol server stores the role of the indexing component, in addition to the row and column position.

3.1.2 Timers

The protocol server MUST use the following timers:

ping_modules: This timer sends **ping** method requests to all registered modules at regular intervals. The default value for this timer is 60 seconds.

inactive_module: This timer removes a module from the inactive modules list if it has not responded to **ping** for a specified amount of time. The default value for this timer is 600 seconds.

3.1.3 Initialization

The protocol server MUST start its XML-RPC application as soon as it can process incoming requests.

The protocol server MUST set up the Configuration Middleware Protocol as specified in [MS-FSCMW], and add those modules to the *active modules* list.

The protocol server MUST initialize the **ping_modules** timer.

3.1.4 Message Processing Events and Sequencing Rule

3.1.4.1 AddCollection

This method uses messages as specified in section 2.2.3.

The protocol server adds the specified content collection to content collections. The protocol server sets the creation and clearance dates to the current time as specified in [ISO-8601].

The protocol server calls the **ConfigurationChanged** method with one of the following combinations of parameters:

- *alerttype* string that contains the value "index" and no additional alert parameters
- *alerttype* string that contains the value "clusters" and no additional alert parameters
- *alerttype* string that contains the value "collection" and the name of the collection as an additional alert parameter

The protocol server MUST return a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.2 in the following cases:

- If the *collection* parameter does not follow the requirements for a *CollectionName* string as specified in section 2.2.1.1

- If a content collection with the same name already exists

If the *pipeline* parameter does not contain "Office14 (webcluster)", the protocol server MUST return a fault message with the <ExceptionType> element that contains the value "ConfigServerException.PipelineError" as specified in section 2.2.2.

If the *cluster* parameter does not contain "webcluster", the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.SearchClusterError" as specified in section 2.2.2.

3.1.4.2 ConfigurationChanged

This method uses messages as specified in section 2.2.4.

The protocol server traverses all the modules in the active modules list to determine which modules registered for alert notifications from the protocol server sends, based on the value of the *alerttype* string. The protocol server creates a connection for every module using the host name and port with which the module is registered, and calls the **ConfigurationChanged** method as specified in section 2.2.4 with the *alerttype* string and any additional alert parameters.

If no active modules subscribed to the specified alert, the protocol server MUST do nothing.

3.1.4.3 GetActiveModuleList

This method uses messages as specified in section 2.2.5.

The protocol server MUST return the host name and port number for all modules in the active modules list that match the specified module type**.** The protocol server returns an empty array if no modules of the specified type exist in the list.

3.1.4.4 GetAllModuleDetails

This method uses messages as specified in section 2.2.6.

The protocol server MUST traverse the active modules list and the inactive modules list to return the details about these modules from the *AllModuleInfo* structure as specified in section 2.2.1.6.

If the protocol server cannot retrieve the module details, the protocol server returns a fault message that contains the <ExceptionType> element that contains the value "ConfigServerException.ModuleError" as specified in section 2.2.2.

3.1.4.5 GetClusterCollections

This method uses messages as specified in section 2.2.7.

The protocol server MUST return the name of all content collections in content collections.

If the *cluster* parameter does not contain "webcluster", the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.SearchClusterError" as specified in section 2.2.2.

3.1.4.6 GetClusters

This method uses messages as specified in section 2.2.8.

The protocol server MUST return an array that contains the string "webcluster".

3.1.4.7 GetClusterSearchColumns

This method uses messages as specified in section 2.2.9.

The protocol server MUST traverse the *indexers* list and return an array with the address of the master indexer node for each index column.

If the specified *cluster* element does not contain "webcluster", the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.SearchClusterError" as specified in section 2.2.2.

3.1.4.8 GetCollection

This method uses messages as specified in section 2.2.10.

If the protocol client sets the *collection* parameter to the name of a non-existent content collection, the protocol server MUST return an empty structure.

3.1.4.9 GetCollectionCleared

This method uses messages as specified in section 2.2.11.

If the *cleared* attribute of the specified collection is not set, the protocol server MUST return an empty string.

If the content collection specified in the *collection* parameter does not exist, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.2.

3.1.4.10 GetCollectionCluster

This method uses messages as specified in section 2.2.12.

If the protocol client sets the *collection* parameter to the name of a non-existent content collection, the protocol server MUST return an empty structure.

3.1.4.11 GetCollectionCreated

This method uses messages as specified in section 2.2.13.

If the *created* attribute of the specified collection is not set, the protocol server MUST return an empty string.

If the content collection specified in the *collection* parameter does not exist, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.2.

3.1.4.12 GetCollectionDataSourceList

This method uses messages as specified in section 2.2.14.

The protocol server MUST return all content sources associated with the specified content collection in content collections.

If the content collection specified in the *collection* parameter does not exist, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.2.

3.1.4.13 GetCollectionList

This method uses messages as specified in section 2.2.15.

The protocol server MUST return the name of all content collections that exist in content collections.

3.1.4.14 GetCollectionPipeline

This method uses messages as specified in section 2.2.16

The protocol server MUST return the string "Office14 (webcluster)".

If the content collection specified in the *collection* parameter does not exist, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.2.

3.1.4.15 GetConfig

This method uses messages as specified in section 2.2.17.

The protocol server looks up the section in universal configuration file and returns it. If the section does not exist, the protocol server returns an empty structure.

3.1.4.16 GetIndexerRowID

This method uses messages as specified in section 2.2.18.

The protocol server MUST find the indexing component that matches the specified host name and port in the *indexers* list and return the row number of the associated indexing component.

If no match is found, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.GeneralError" as specified in section 2.2.2.

3.1.4.17 GetModuleInfo

This method uses messages as specified in section 2.2.19.

The protocol server MUST find the module with the specified host name and port in the active modules list or the inactive modules list. If the module is found, the protocol server returns the registered information for this module. If no match is found, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ModuleError" as specified in section 2.2.2.

3.1.4.18 GetModuleList

This method uses messages as specified in section 2.2.20.

The protocol server MUST look up the module type in both the active modules list and the inactive modules list, and return all modules that match the specified criteria. The protocol server returns an empty array if there are no matches.

3.1.4.19 GetSearchBackupList

This method uses messages as specified in section 2.2.21.

The method MUST traverse the *indexers* list and find all indexing components that act as a backup. These components are in the same index column as the master indexer node, as specified with the *host name* and *port* parameters. If no components are found, the method returns an empty array.

3.1.4.20 GetSearchColumnCluster

This method uses messages as specified in section 2.2.22.

If an indexing component that matches the specified *host* and *port* parameters is found in the *indexers* list, the method returns a string that contains "webcluster". Otherwise it returns an empty string.

3.1.4.21 GetSearchColumnFTMode

This method uses messages as specified in section 2.2.23.

The method MUST return an integer with the value 1 if the index column that contains the indexing component that matches the specified *host* and *port* parameters is running in fault-tolerant mode. If the index column is not running in fault-tolerant mode, the method returns a value of 0.

If the specified indexing component is not found in the *indexers* list, the method returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.SearchClusterError" as specified in section 2.2.2.

3.1.4.22 GetSearchColumnPartitionID

This method uses messages as specified in section 2.2.24.

The protocol server MUST return the column number of the indexing component that registered in the *indexers* list with the specified *host name* and *port* parameters.

If the indexing component specified by the *host name* and *port* parameters is not found, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.SearchClusterError" as specified in section 2.2.2.

3.1.4.23 IsValidCollection

This method uses messages as specified in section 2.2.25.

The protocol server MUST return 1 if a content collection exists in content collections that is associated with the name specified in the *collection* parameter. If no content collection with this name exists, the method returns a value of 0.

3.1.4.24 LoadConfigFile

This method uses messages as specified in section 2.2.26.

The protocol server opens the file specified by the *module* and *filepath* parameters, read the information from this configuration file, and return it. This method is not suited for the transfer of binary files, use LoadConfigFileBase64 instead.

The protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ConfigError" as specified in section 2.2.2 in the following cases:

- If the *module* parameter is an absolute path
- If the path to the configuration file specified by the *module* and *filepath* parameters is not located in the configuration file directory that the protocol server uses for the specified module
- If there is no configuration file that matches the specified *module* and *filepath* parameters

3.1.4.25 LoadConfigFileBase64

This method uses messages as specified in section 2.2.27.

The protocol server opens the file specified by the *module* and *filepath* parameters, and read the information from this configuration file. It encodes the information using base64 encoding as specified in [RFC4648] and returns it.

The protocol server MUST return a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ConfigError" as specified in section 2.2.2 in the following cases:

- If the *module* parameter is an absolute path
- If the path to the configuration file specified by the *module* and *path* parameters is not located in the directory where the protocol server stores the configuration files for the specified module
- If there is no configuration file that matches the specified *module* and *filepath* parameters
- If the protocol server cannot encode the information using base64 encoding as specified in [RFC4648]

3.1.4.26 LoadOptionalConfigFile

This method uses messages as specified in section 2.2.28.

The protocol server opens the file that matches the specified *module* and *path* parameters, read the information from this configuration file, and return it.

If there is no configuration file that matches the parameters, the protocol server returns an empty string.

3.1.4.27 LoadOptionalConfigFileBase64

This method uses messages as specified in section 2.2.29.

The protocol server MUST open the file that matches the specified module and *filepath* parameters, and read the information from this configuration file. It encodes the information using base64 encoding as specified in [RFC4648] and return it.

If there is no configuration file that matches the parameters, the protocol server returns an empty string.

3.1.4.28 ping

This method uses messages as specified in section 2.2.30.

When the protocol server calls the **ping** method, it adds the request to the list of **ping** requests as specified in section 3.2.4.2.

When receiving a response from a **ping** request, the protocol server removes the request from the list of **ping** requests. If the module that sent the response is in the inactive modules list, the protocol server calls the **ReRegister** method as described in section 3.2.4.3, removes the module from the inactive modules list, and deletes the **inactive_module** timer for the module.

3.1.4.29 RegisterModule

This method uses messages as specified in section 2.2.31.

The protocol server adds the module and all information specified in the *moduleinfo* parameter to the active modules list. The *moduleinfo* parameter includes address information that the protocol server uses to contact the module regularly to verify continued availability. The protocol client subscribes to alerts by specifying at least one *alerts* parameter in the *ModuleRegister* structure as specified in section 2.2.1.3.

The protocol server calls the **ConfigurationChanged** method as specified in section 3.2.4.1. If the protocol server sends an alert to the protocol client, it specifies additional parameters for the *alerttype* string and the host name and port of the module. The host name and port are in a *ModuleAddress* array as specified in section 2.2.1.2.

If the *moduleinfo* parameter was not correctly formatted, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ModuleError" as specified in section 2.2.2.

3.1.4.30 RegisterProcessorPipelines

This method uses messages as specified in section 2.2.32.

If the specified **ProcessorServer** already has a collection of *pipelines* arrays associated with it, the protocol server adds the specified *pipelines* array to the existing collection.

If the specified **ProcessorServer** is not an active document processor server, the protocol server MUST do nothing.

3.1.4.31 RemoveCollection

This method uses messages as specified in section 2.2.33.

If the *force* parameter contains the value 1, the protocol server removes the content collection from the list of content collections, and calls the **ConfigurationChanged** method with the *alerttype* string that contains the value "collection" and the name of the collection that was removed. The protocol server then returns a value of 1.

If the *force* parameter is 0, the protocol server calls the **ConfigurationChanged** method as specified in section 3.2.4.1 with one of the following combinations of parameters:

- *alerttype* string that contains the value "index" and no additional alert parameters
- *alerttype* string that contains the value "collection" and the name of the content collection as additional *alert* parameter
- *alerttype* string that contains the value "clusters" and no additional alert parameters

The protocol server then removes the content collection from content collections. It calls the **ConfigurationChanged** method as specified in section 3.2.4.1 with one of the following combinations of parameters:

- *alerttype* string that contains the value "collection" and the name of the content collection as an additional *alert* parameter
- *alerttype* string that contains the value "index" and no additional *alert* parameters

If the content collection does not exist, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.22.

3.1.4.32 ReRegister

This method uses messages as specified in section 2.2.34. The protocol server establishes communication with the module, and calls the **ReRegister** method as specified in section 2.2.33.

After that, when the protocol server receives a response from a module, it adds the module to the active modules list, and includes additional information in the *ModuleRegister* structure as specified in section 2.2.1.3.

The protocol server calls the **ConfigurationChanged** module and specifies additional parameters for the *alerttype* string and the host name and port of the module. The host name and port MUST be in a *ModuleAddress* array as specified in section 2.2.1.2.

3.1.4.33 SaveConfigFile

This method uses messages as specified in section 2.2.35.

The protocol server MUST store the file in the path specified in the *filepath* parameter within a separate directory reserved for the module specified in the *module* parameter.

Unless the optional parameter *sendAlert* is set to 0, the protocol server calls the **ConfigurationChanged** method as specified in section 3.2.4.1 with the *alerttype* string that contains the value "configfile" and an array that contains the *module* and *filepath* parameters. This method is not suited for the transfer of binary files, use SaveConfigFileBas64 instead.

The protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ConfigError" as specified in section 2.2.2 in the following cases:

- If the *module* parameter is an absolute path
- If the path to the configuration file specified by the *module* and *filepath* parameters is not located in the directory in which the protocol server stores the configuration files for the specified module
- If the protocol server cannot store the configuration file

3.1.4.34 SaveConfigFileBase64

This method uses messages as specified in section 2.2.36.

The protocol server MUST decode the information as specified in [RFC4648], and store it in the path specified in the *filepath* parameter within a separate directory reserved for the module specified in the *module* parameter.

Unless the optional parameter *sendAlert* is set to 0, the protocol server calls the **ConfigurationChanged** method as specified in section 3.2.4.1 with the *alerttype* string that contains the value "configfile" and an array that contains the *module* and *filepath* parameters.

The protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ConfigError" as specified in section 2.2.2 in the following cases:

- If the *module* parameter is an absolute path
- If the path to the configuration file specified by the *module* and *filepath* parameters is not located in the directory in which the protocol server stores the configuration files for the specified module
- If the protocol server cannot store the configuration file
- If the protocol server cannot decode the information encoded using base64 encoding

3.1.4.35 SendAlert

This method uses messages as specified in section 2.2.37.

Alerts are triggered by changes to the system model, updated configuration files, or are explicitly called by protocol clients. The protocol server MUST call the **ConfigurationChanged** method as specified in section 3.2.4.1 with the *alerttype* and *alert* parameters that the protocol client specified when it called the **SendAlert** method.

3.1.4.36 SetConfig

This method uses messages as specified in section 2.2.38.

The protocol server adds the configuration to the specified section in the universal configuration file.

If the *config* parameter does not follow the restrictions for the *UniversalConfig* structure as specified in section 2.2.1.9, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ConfigError" as specified in section 2.2.2.

3.1.4.37 UnregisterModule

This method uses messages as specified in section 2.2.39.

The protocol server traverses the active modules list and the inactive modules list to remove the registered information about the module.

The protocol server calls the **ConfigurationChanged** method with the *alerttype* string set to the type with which the module registered, and the location of the module as a *ModuleAddress* array as additional *alert* parameter.

If the module registered with the *alerttype* string set to "ProcessorServer", the protocol server calls the **ConfigurationChanged** method again, with the *alerttype* string set to "processorserver_removed" and the location of the module as a *ModuleAddress* array as an additional *alert* parameter, as specified in section 2.2.1.2.

If the *moduleinfo* parameter does not contain the required information, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ModuleError" as specified in section 2.2.2.

3.1.4.38 UpdateCollection

This method uses messages as specified in section 2.2.40.

If no content collection whose name matches the specified *collection* parameter exists, the protocol server adds a new content collection to content collections with the name, description, and content sources that are specified in the parameters.

If the content collection already exists, the protocol server updates the description and content sources for this content collection in content collections. If the value of the *description* parameter begins with the value "base64_", the remainder of the string is encoded using base64 encoding as specified in [RFC4648]. The protocol server decodes the remainder of the string before updating the *description* field with this value.

If there is a content source specified in the *datasources* parameter that is not associated with a content collection in content collections, then the protocol server associates the content source with the content collection. If there is a content source that is not specified in the *datasources* parameter, the protocol server removes the association in content collections for that content source.

The protocol server calls the **ConfigurationChanged** method as specified in section 3.2.4.1 with one of the following combinations of parameters:

- *alerttype* string that contains the value "index" and no additional alert parameters
- *alerttype* string that contains the value "clusters" and no additional alert parameters
- *alerttype* string that contains the value "collection" and the name of the content collection as an additional alert parameter

If the *collection* parameter does not follow the requirements for a *CollectionName* string as specified in section 2.2.1.1, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.CollectionError" as specified in section 2.2.2.

If the *pipeline* parameter does not contain "Office14 (webcluster)", the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.PipelineError" as specified in section 2.2.2.

If the *cluster* parameter does not contain "webcluster", the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.SearchClusterError" as specified in section 2.2.2.

If the *datasources* parameter specifies a content source that is associated with another content collection in content collections, the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.DataSourceError" as specified in section 2.2.2.

If the *datasources* parameter specifies a content source that is not registered in the active modules list with a module type that contains the value "DataSource", the protocol server returns a fault message with the <ExceptionType> element that contains the value "ConfigServerException.ModuleError" as specified in section 2.2.2.

3.1.5 Timer Events

3.1.5.1 ping_module Timer

When the **ping_module** timer expires, the protocol server MUST determine whether all modules in **active modules** and **inactive modules** are available.

First, the protocol server tests the **ping** request list for modules that did not receive a reply. Modules that did not receive a reply are moved to the inactive modules list, and then the protocol server starts the **inactive_module** timer for those modules.

The protocol server traverses the active module and inactive module lists and does the following for each module:

- If the type of the module is *DataSource* and the name of the module is *Enterprise Crawler*, or if the module registered using port number 0, the module is ignored.
- If the *middleware* element of the module contains the value "mw", the protocol server MUST determine whether the module is available as specified in [MS-FSCMW]. If not, the protocol server moves the module from the active modules list to the inactive modules list.
- If the *middleware* element of the module contains the value "xml-rpc", as specified in section 2.2.1.5, the protocol server calls the **ping** method as specified in section 2.2.29.

3.1.5.2 inactive_module Timer

When the **inactive_module** timer expires, the protocol server removes the module from the inactive modules list.

3.1.6 Other Local Events

None.

3.2 Module 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.

Modules MUST maintain the following state:

module_info: This is a data structure that contains the name, type, and version of the module. It also contains the port number on which the module listens for incoming calls to its XML-RPC methods, as well as a list of alerts from the protocol server to which the module subscribes. The module can also include additional information in this data structure.

Modules can also maintain the following state:

last_ping_time: This is the time that the protocol server last called the **ping** method as specified in section 2.2.30 for this module.

3.2.2 Timers

Modules can use the following timers:

last_pinged_check: This timer determines whether the protocol server is still available. If it is available, the protocol server regularly sends a ping request, and this timer determines whether the ping method as specified in section 2.2.30 was called recently. The default value for this timer is 30 seconds.

3.2.3 Initialization

A module sets up a port to listen for XML-RPC requests. It MUST save the port number in the *module_info* structure.

The module sets up a TCP/IP connection to the protocol server using the host name and port of the protocol server. This connection is used to call the **RegisterModule** method as specified in section 2.2.31. The module uses the information stored in the *module_info* structure to set the parameters of this method as specified.

If the module uses the **last_pinged_check** timer it MUST initialize the timer.

3.2.4 Message Processing Events and Sequencing Rules

3.2.4.1 ConfigurationChanged

When the **ConfigurationChanged** method is called, the module takes whatever action is appropriate based on the parameters.

3.2.4.2 ping

When the **ping** method is called, the module MUST reply with a value of "pong". The module updates **last_ping_time** if the module uses this state.

3.2.4.3 ReRegister

When the **ReRegister** method is called, the module MUST send the contents of **module_info** in the reply.

3.2.4.4 UnregisterModule

When a module is terminated it calls the **UnregisterModule** method specified in section 2.2.39 using information stored in the **module_info** xxx as parameters. A module can be terminated without calling this method, or modules can ignore calling this method. The protocol server marks these modules as inactive and eventually removes them from its list.

3.2.5 Timer Events

If the module uses the **last_pinged_check** timer, it reads the **last_ping_time** variable to determine whether the timer expired between the previous **ping** request and the current **ping** request. If the module was not pinged for 600 seconds, it MUST re-register with the protocol server by calling the **RegisterModule** method as specified in section 2.2.31.

3.2.6 Other Local Events

None.

4 Protocol Examples

These examples contain only the XML body for each message. An example of how the HTTP headers are constructed is described in [XML-RPC].

4.1 RegisterModule

The following is an example of how a module can register itself.

```
<?xml version='1.0'?>
<methodCall>
  <methodName>RegisterModule</methodName>
  <params>
    <param>
      <value><struct>
        <member>
          <name>hostname</name>
          <value><string>example.host.com</string></value>
        </member>
        <member>
          <name>name</name>
          <value><string>SPRel</string></value>
        </member>
        <member>
          <name>alerts</name>
          <value><array><data>
            <value><string>collection</string></value>
            <value><string>myalert</string></value>
          </data></array></value>
        </member>
        <member>
          <name>version</name>
          <value><string>fs14.200</string></value>
        </member>
        <member>
          <name>type</name>
          <value><string>SPRel</string></value>
        </member>
        <member>
          <name>port</name>
          <value><int>20002</int></value>
        </member>
      </struct></value>
    </param>
  </params>
</methodCall>
```

The following is an example reply from the server.

```
<?xml version='1.0'?>
<methodResponse>
  <params>
    <param>
      <value><string>example.host.com</string></value>
    </param>
  </params>
```

```
</methodResponse>
```

4.2 SendAlert and ConfigurationChanged

The following is an example of how a protocol client sends an alert to all modules that subscribe to the alert specified with an *alerttype* string that contains the value "myalert".

```
<?xml version='1.0'?>
<methodCall>
  <methodName>SendAlert</methodName>
  <params>
    <param>
      <value><string>myalert</string></value>
    </param>
    <param>
      <value><array><data>
        <value><int>1</int></value>
        <value><string>parse</string></value>
      </data></array></value>
    </param>
  </params>
</methodCall>
```

The following is an example of a reply sent by the protocol server.

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

The protocol server also calls the **ConfigurationChanged** method for all modules that subscribe to the alert specified with an *alerttype* string that contains the value "myalert", as described in section 2.2.4.

```
<?xml version='1.0'?>
<methodCall>
  <methodName>ConfigurationChanged</methodName>
  <params>
    <param>
      <value><string>myalert</string></value>
    </param>
    <param>
      <value><array><data>
        <value><int>1</int></value>
        <value><string>parse</string></value>
      </data></array></value>
    </param>
  </params>
</methodCall>
```

Each module then sends the following reply.

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

4.3 GetCollection

The following is an example that shows the request to call the **GetCollection** method as specified in section 2.2.10.

```
<?xml version='1.0'?>
<methodCall>
  <methodName>GetCollection</methodName>
  <params>
    <param>
      <value><string>sp</string></value>
    </param>
  </params>
</methodCall>
```

The following is an example of a response returned by the protocol server .

```
<?xml version='1.0'?>
<methodResponse>
  <params>
    <param>
      <value><struct>
        <member>
          <name>pipeline</name>
          <value><string>Office14 (webcluster)</string></value>
        </member>
        <member>
          <name>datasources</name>
          <value><array><data>
          </data></array></value>
        </member>
        <member>
          <name>description</name>
          <value><string>Default collection for SharePoint content</string></value>
        </member>
        <member>
          <name>cluster</name>
          <value><string>webcluster</string></value>
        </member>
        <member>
          <name>cleared</name>
          <value><string></string></value>
        </member>
        <member>
          <name>created</name>
```

```
            <value><string></string></value>
          </member>
          <member>
            <name>name</name>
            <value><string>sp</string></value>
          </member>
        </struct></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

F

G

I

L

M

N

O

P

R

S

T

U

V

X