[MS-FSNC]:
Node Controller Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	Editorial	Revised and edited 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.

Table of Contents

1 Introduction

This document specifies the Node Controller Protocol. This protocol handles communication between a protocol client and a node controller protocol server. There is a system that consists of a set of processes and files, and it exists on the same computer as the node controller protocol server. The node controller protocol server monitors and controls that system. A protocol client is typically part of an application that requests information about the system or requests modifications to the system.

1.1 Glossary

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

base64
Coordinated Universal Time (UTC)
XML

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

HTTP GET
TCP/IP

The following terms are specific to this document:

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

1.2 References

1.2.1 Normative References

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

[MC-RegEx] Microsoft Corporation, "Regular Expression Language Elements", http://msdn.microsoft.com/en-us/library/az24scfc(VS.80).aspx

[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

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

1.2.2 Informative References

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

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

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

1.3 Protocol Overview (Synopsis)

This protocol specifies the communication between a protocol client and a node controller protocol server. The protocol server supports methods for controlling the node on which it runs and verifying current states. These methods perform the following operations:

- Manage a configured set of processes on the node. This includes verifying the current state of processes and starting or stopping processes.
- Manage a subset of the files available on the node. This includes uploading configuration files, downloading files, retrieving status for disk usage on the node, or retrieving information about individual files or directories.

The following figure describes the message exchange between the protocol client and the protocol server.

Figure 1: Client-server message exchange

1.4 Relationship to Other Protocols

This protocol uses XML-RPC over HTTP as shown in the following layering diagram.

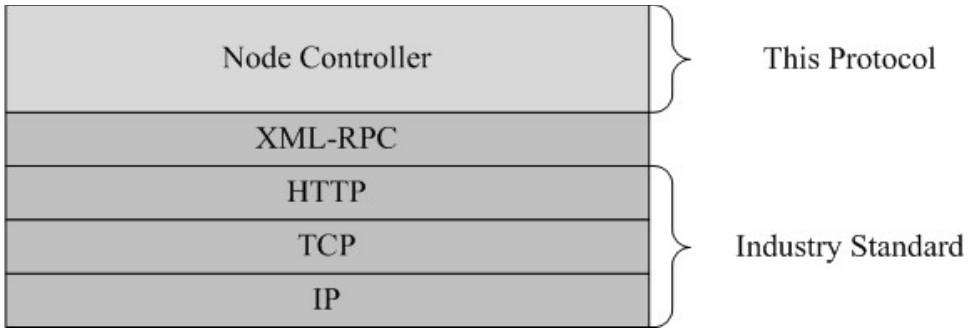

Figure 2: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

This protocol requires that the protocol client know the hostname and port of the protocol server prior to establishing a connection over **TCP/IP**.

1.6 Applicability Statement

This protocol is designed for controlling and monitoring a set of predefined processes on a remote node. It also retrieves or stores small configuration files on a node whose directory is within a predefined set of directories.

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] as transport mechanism over TCP/IP.

2.2 Message Syntax

The **HTTP GET** path, as specified in [RFC2616], MUST be "/RPC2". The request and response messages in the following sections have been specified with method signatures that map to XML body requests and responses as specified in [MS-FSXTAPI]. Examples of how these signatures map to XML body requests and responses are in section 4.

2.2.1 Common Data Types

This protocol uses data structures. Each of these data structures is either a composite data type, or one of the basic data types with additional limitations. For more information about **int**, **string**, **array**, and **struct** data types, see [XML-RPC].

The following table summarizes the data structures that this protocol uses.

Name	Description
RelativePath	A file path that is relative to the root of the install directory, as specified in section 2.2.1.1.
ProcessState	The state of a process, as specified in section 2.2.1.2.
ProcessInfo	Information about a process, as specified in section 2.2.1.3.
ProcessName	The name of the process, as specified in section 2.2.1.4.
ProcessList	A list of processes and their state, as specified in section 2.2.1.5.
DirectoryDiskUsage	Disk usage status for a directory, as specified in section 2.2.1.6.
NodeDiskUsage	Disk usage status for the entire node, as specified in section 2.2.1.7.
NodeState	The current status of the node, as specified in section 2.2.1.8.
OperationResult	Status for a process operation, as specified in section 2.2.1.9.
FileInfo	Information about a file, as specified in section 2.2.1.10.

2.2.1.1 RelativePath Data Type

This is a string that represents the relative path that starts with a directory. The protocol server MUST use the root of its install directory as the base directory for the relative path. Both slash (/) and backslash (\) are acceptable as path separators.

The following list specifies directories that can begin a valid relative path:

- etc
- templates
- var

- resources
- data
- lib/python2.5/processors

2.2.1.2 ProcessState Data Type

This specifies the state of a process. It MUST be of type **int** and contain a value that is specified in the following table.

Value	State	Description
1	Dead	The process is not running, and neither the system nor the user has requested that it be stopped.
2	Suspended_auto	The process was stopped by the system.
3	Suspended_user	The process was stopped by the user.
4	Running	The process is running.
5	Stopping	The protocol server initiated stopping the process, but it has not received confirmation that the **StopProcess** method has finished.
6	Restarting	The protocol server initiated a restart of the process, but it has not received confirmation that the current process has terminated.

2.2.1.3 ProcessInfo Data Type

This contains information about a process, represented as a **struct** that contains all of the elements that are specified in the following table.

Name	Type	Description
state	**ProcessState**, as specified in section 2.2.1.2.	The current state of the process.
pid	**int**	The process identifier. MUST be 0 if the process is not running.
name	**string**	A description of the process.

2.2.1.4 ProcessName Data Type

This contains the name of the process. It is a string that matches a name in the list of configured processes maintained by the node controller protocol server, as specified in section 3.2.1.

2.2.1.5 ProcessList Data Type

This contains a list of the processes on a node, in addition to the state associated with each process. It is represented as a **struct**, where each member has a name of type **ProcessName**, as specified in section 2.2.1.4, and a value of type **ProcessInfo**, as specified in section 2.2.1.3.

2.2.1.6 DirectoryDiskUsage Data Type

This specifies the disk usage in a directory. It is represented as a **struct** that contains all of the elements that are specified in the following table.

Name	Type	Description
dir	string	The absolute path to the directory. Both slash (/) and backslash (\) MUST be accepted as the path separator.
mountpoint	string	The root of the drive on which this directory is located. For example "C:\".
disktotal	string	A string representation of an integer that specifies the size of this file system in bytes.
diskfree	string	A string representation of an integer that specifies the number of bytes that are available on this file system.

2.2.1.7 NodeDiskUsage Data Type

This contains information about the disk usage for a node. It is represented as a **struct** that contains all of the elements that are specified in the following table.

Name	Type	Description
data	**DirectoryDiskUsage** as specified in section 2.2.1.6.	Disk status for the **data** element.
fixml	**DirectoryDiskUsage** as specified in section 2.2.1.6.	Disk status for the **fixml** element.
index	**DirectoryDiskUsage** as specified in section 2.2.1.6.	Disk status for the **index** element.
var	**DirectoryDiskUsage** as specified in section 2.2.1.6.	Disk status for the **var** element.
disktotal	string	A string representation of an integer that specifies the total size in bytes for the file system.
diskfree	string	A string representation of an integer that specifies the number of bytes available on the file system.

2.2.1.8 NodeState Data Type

This contains information about a node. It is of type **struct** and contains all of the elements that are specified in the following table.

Name	Type	Description
df	**NodeDiskUsage**, as specified in section 2.2.1.7.	Disk usage status for the node.
FASTSEARCH	**string**	Absolute path to the root of the install directory for the node controller protocol server. Both slash (/) and backslash (\) MUST be accepted as the path separator.

2.2.1.9 OperationResult Data Type

This is an array that contains three values that represent status information associated with starting or stopping a process. Values are specified in the following table.

Index	Type	Description
0	**ProcessName**, as specified in section 2.2.1.4	The name of the process.
1	**int**	Status code for the operation. Values are specified in the following table.
2	**string**	Text description of the status for the operation.

Values for the status code field when a process is started are specified in the following table.

Status value	Description
0	Cannot start the process.
1	Process started successfully or process was already running.
5	Process is not started because it is in the Stopping state, as specified in section 2.2.1.2. The process will start when the current process ends.
6	Process did not start because it is in the Restarting state, as specified in section 2.2.1.2. The process will start when the current process ends.

Values for the status code field when a process is stopped are specified in the following table.

Status value	Description
0	The protocol server did not stop the process.
1	The protocol server issued a **StopProcess** request. The text description of the status specifies whether it stopped or if the protocol server is waiting for the process to stop.

2.2.1.10 FileInfo Data Type

This contains the size and modification time of a file. It is represented as an array that contains the two values specified in the following table.

Index	Type	Description
0	**int**	The size of the file in bytes.
1	**int**	The number of seconds elapsed between midnight of January 1, 1970, not counting leap seconds, and the time the file was last modified. This amount is specified in **UTC**.

2.2.2 Fault Message

This protocol supports fault messages to report errors to the protocol client. The fault message contains a fault code and descriptive fault text.

```
struct faultMessage {
    int code;
    string description;
}
```

In this protocol, the fault code is an **int** that contains the value 1. The **string** that contains the description of the fault is specified in the following format.

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

The **ErrorText** element MUST be a text description of the error, while the **ExceptionType** element MUST be a value specified in the following table.

Value	Description
GeneralNCError	General node controller error.
ProcessManagerError	Error related to process management or file permissions.
ConfigParseError	Can not parse the configuration file.
InterfaceError	Unexpected errors while processing the request.

For more information about fault messages, see [XML-RPC].

2.2.3 GetDataBase64

This retrieves data from a file and encodes it using **base64** encoding, as specified in [RFC4648]. The signature of this method is specified as follows.

string: GetDataBase64(string filepath, int startingOffset=0, int size=0)

filepath: Relative path to the file. It is of type **RelativePath**, as specified in section 2.2.1.1.

startingOffset: Optional. Specifies the number of bytes to skip from the beginning of the file before reading the file. The default value is 0.

size: Optional. Specifies the number of bytes to read.

Return value: The protocol server MUST return the file contents as a string that is encoded as **base64**, as specified in [RFC4648].

2.2.4 GetDirList

This lists the contents in a directory. The signature of this method is specified as follows.

```
array GetDirList(string path, string regexp="")
```

path: A relative path to the directory. It is of type **RelativePath**, as specified in section 2.2.1.1.

regexp: Optional regular expression, as specified in [MC-RegEx].

Return value: The protocol server MUST return an array of type **RelativePath** containing the name of a file or directory, as specified in section 2.2.1.1.

2.2.5 GetProcessList

This queries the protocol server for the status of registered processes. The signature of this method is specified as follows.

```
struct GetProcessList()
```

return value: The protocol server MUST return an element of type **ProcessList**, as specified in section 2.2.1.5.

2.2.6 GetQuickFileInfo

This returns information about the size and modification time of a file. The signature of this method is specified as follows.

```
array GetQuickFileInfo(string filepath)
```

filepath: A relative path to the file. It is of type **RelativePath**, as specified in section 2.2.1.1.

Return value: The protocol server MUST return an array of type **FileInfo** as specified in section 2.2.1.10.

2.2.7 GetStatus

This queries the protocol server for the status information that is specified in a **NodeState** element, as specified in section 2.2.1.8. The signature of this method is specified as follows.

```
struct GetStatus()
```

return value: The protocol server MUST return a value of type **NodeState**, as specified in section 2.2.1.8.

2.2.8 RestartProcess

This restarts a list of processes and is equivalent to calling the **StopProcess** method, as specified in section 2.2.11, followed by the **StartProcess** method, as specified in section 2.2.10, for each process. The signature of this method is specified as follows.

```
array RestartProcess(array processes)
```

processes: A list of processes to restart. This is an array of type **ProcessName**, as specified in section 2.2.1.4.

Return value: This is an array of type **OperationResult**, as specified in section 2.2.1.9. For each process in the *processes* parameter there are two **OperationResult** elements in the response, where the first contains a status code for the **StopProcess** method and the second contains a status code for the **StartProcess** method.

2.2.9 SaveConfigFile

This stores a configuration file on the node. The signature of this method is specified as follows.

```
int SaveConfigFile(string filepath, string data)
```

filepath: A relative path to the configuration file. It is of type **RelativePath**, as specified in section 2.2.1.1.

data: The information to save in the file.

Return value: The protocol server MUST return a value of 1.

2.2.10 StartProcess

This starts a list of processes. The signature of this method is specified as follows.

```
array StartProcess(array processes)
```

processes: A list of processes to start. The *processes* parameter MUST be an array of type **ProcessName**, as specified in section 2.2.1.4.

Return value: The protocol server MUST return an array of type **OperationResult** elements, as specified in section 2.2.1.9. Each **OperationResult** element is associated with a process in the *processes* parameter. The value of the status code in an **OperationResult** element MUST be set as specified in section 2.2.1.9.

2.2.11 StopProcess

This stops a list of processes. They MUST remain stopped until they are explicitly restarted. The signature of this method is specified as follows.

```
array StopProcess(array processes, int kill=0)
```

processes: A list of processes to stop. This is an array of type **ProcessName**, as specified in section 2.2.1.4.

kill: Optional. Contains a value of either 0 or 1.The default value is 0.

Return value: The protocol server MUST return an array of elements of type **OperationResult**. Each **OperationResult** element is associated with a process in the *processes* parameter. The value of the **status** element in an **OperationResult** element MUST be set as specified in section 2.2.1.9.

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

The protocol client MUST establish a TCP/IP connection to the protocol server using the hostname and port of the protocol server.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Faults

The protocol client MUST handle the fault messages that the protocol server sends in response to any request, as specified in section 2.2.2. This is implementation-specific.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

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

The protocol server MUST maintain the following states:

node_config: A list of all processes that the protocol server manages. Each item contains a process name that is identical to the **ProcessName** data type, specified in section 2.2.1.4. Each item also contains a description of the process, and a **process_state** item.

process_state: A data structure that tracks which processes are in *Suspended_user*, *Suspended_auto*, *Stopping*, and *Restarting* states, as specified in section 2.2.1.2. The two other states, *Running* and *Dead*, MUST be verified periodically to determine whether the state is current.

subdirs: A set of installation subdirectories in which the protocol server stores, browses, and reads files. The subdirectories are specified in section 2.2.1.1.

diskusagedirs: Represents the directories for which the protocol server verifies disk usage. Contains the location for all directories that are specified in the **NodeDiskUsage** data type, specified in section 2.2.1.7.

3.2.2 Timers

The protocol server MUST use the following timers:

process_transition: Verifies processes that are in transitional states, *Stopping* or *Restarting*, as specified in section 2.2.1.2. The default value for this timer is five seconds.

process_alive: Verifies that running processes are still alive. The default value for this timer is 30 seconds.

Both timers are continuous, therefore they MUST reset when they expire. This restarts the timer countdown.

3.2.3 Initialization

The protocol server starts the XML-RPC implementation as soon as it can process incoming requests. It registers with the Configuration Service, as specified in [MS-FSCX], and implements the following methods that are required by that protocol: **ConfigurationChanged**, **ReRegister**, and **ping**. When registering, the protocol server MUST specify "NodeControl" as the module type and "NodeControl" as the module name.

The protocol server MUST initialize the following items: the timers, the **node_config** by reading it from a configuration file, and the **process_state** by retrieving the current state of all specified processes. The **node_config** data type is specified in section 2.2.1.4, and the **process_state** data type is specified in section 2.2.1.2. For more information about XML-RPC implementations, see [XML-RPC].

3.2.4 Higher-Layer Triggered Events

There are no higher-layer triggered events.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Fault Message

The protocol server returns a fault message if it can not handle a request or if the request parameters were not created as specified in this document. The format of the fault message is specified in section 2.2.2.

The protocol server returns a fault message that contains the **InterfaceError** exception type if an internal exception occurs, as specified in section 2.2.2. All other faults that the protocol server returns are specified for each method.

3.2.5.2 GetDataBase64

The protocol server MUST verify that the requested file is located within a directory stored in the **subdirs** set. It begins reading the file at the specified offset, and then reads the amount of data specified. If the starting offset is not provided, the protocol server reads from the beginning of the file. If the amount of information to read is not specified or is set to 0, then the protocol server reads the remaining information in the file. The protocol server encodes the information using base64 encoding as specified in [RFC4648].

The protocol server returns a fault message that contains the **ProcessManagerError** exception type, as specified in section 2.2.2, in the following cases:

- The *filepath* parameter is not a file.
- The **subdirs** element does not contain the *filepath* parameter.
- An error occurred while reading from the file.

3.2.5.3 GetDirList

The protocol server MUST verify that the specified directory is located within a directory stored in the **subdirs** element. The protocol server sends a list of the files and directories that the specified directory contains. Only the files and directories whose names match the specified regular expression are returned. If the regular expression is omitted or is an empty string, the protocol server sends all files and directories in the specified directory.

The protocol server returns a fault message that contains the **GeneralNCError** exception type, as specified in section 2.2.2, in the following cases:

- The *regexp* parameter contains an invalid regular expression format.
- The **subdirs** element does not contain the *path* parameter, or the directory specified in the *path* parameter does not exist.
- The protocol server can not list the contents in the directory.

3.2.5.4 GetProcessList

The protocol server sends information about the processes and their processing state as specified in section 2.2.1.2. The protocol server iterates through the processes that are specified in **node_config**. If the state of the process is specified in **process_state**, it can be returned directly. All other processes MUST be verified as running.

The protocol server MUST find the process identifiers for all processes whose process state is not "Suspended_auto", "Suspended_user", or "Dead". The value 0 is used as a process identifier for the processes that are not running.

The protocol server returns a fault message that contains the **ProcessManagerError** exception type, as specified in section 2.2.2, if it can not return a complete list of processes and processing states.

3.2.5.5 GetQuickFileInfo

The protocol server sends the size and modification time of the specified file. It returns a fault message that contains the **GeneralNCError** exception type as specified in section 2.2.2 in the following cases:

- The *filepath* parameter does not contain a file.
- The **subdirs** element does not contain the *filepath* parameter.
- The protocol server can not retrieve information about the file.

3.2.5.6 GetStatus

The protocol server retrieves a set of directories from **diskusagedirs**. It verifies the amount of disk space that is available for these directories, both the total size and the amount of available space. The protocol server MUST also return the root of its install directory.

The protocol server returns a fault message that contains the **ConfigParseError** exception type, as specified in section 2.2.2, if it can not retrieve the location of the directories for which it MUST verify disk usage from **diskusagedirs**.

3.2.5.7 RestartProcess

The **RestartProcess** method restarts a list of processes.

The protocol server performs the equivalent of a **StopProcess** method, as specified in section 3.2.5.10, with the *kill* parameter set to 0, followed by a **StartProcess** method as specified in section 3.2.5.9, where the only difference is that the **RestartProcess** method does not return a fault message when a process is in the "Suspended_auto", "Suspended_user", "Stopping", or "Restarting" state in **process_state**. It instead returns 0 in the status code for the stop operation of this process, as specified in section 2.2.1.9.

The protocol server returns a fault message that contains the **ProcessManagerError** exception type, as specified in section 2.2.2, if an error occurs while starting or stopping a process.

3.2.5.8 SaveConfigFile

The protocol server verifies that the file is located within a directory in the **subdirs** element and stores the configuration information in the specified file.

The protocol server returns a fault message that contains the **ProcessManagerError** exception type, as specified in section 2.2.2, in the following cases:

- The **subdirs** element does not contain the *filepath* parameter.
- The protocol server can not write the file.

3.2.5.9 StartProcess

The **StartProcess** method starts a list of processes. For each process the protocol server MUST take the following actions:

- If the state of the process contains the enumeration "Stopping", as specified in section 2.2.1.2, the protocol server MUST change the state to "Restarting", as specified in section 2.2.1.2. The protocol server MUST not start the process in this case because the other instance of it MUST be stopped first.
- If the state of the process is "Restarting", as specified in section 2.2.1.2, the protocol server MUST do nothing.

- If the state of the process is "Suspended_user" or "Suspended_auto", as specified in section 2.2.1.2, the protocol server MUST remove it from **process_state** and start the process.
- If the process is running, the protocol server MUST do nothing.
- If the process is dead, the protocol server MUST restart it.

The protocol server returns a fault message that contains the **ProcessManagerError** exception type, as specified in section 2.2.2, if an error occurs before the protocol server starts the processes.

3.2.5.10 StopProcess

The **StopProcess** method stops a list of processes.

For each process, the protocol server MUST take the following actions:

- If the *kill* parameter is set, the protocol server stops the process immediately.
- If the *kill* parameter is not set, the protocol server allows time for the process to shut down. If the process is still running when the protocol server sends the response, the protocol server updates the **process_state** to "Stopping", as specified in section 2.2.1.2.

If the protocol server stops the process, it MUST update the state of the process to a value of "Suspended_user", as specified in section 2.2.1.2.

The protocol server returns a fault message that contains the **ProcessManagerError** exception type, as specified in section 2.2.2, in the following cases:

- A process is in the "Suspended_auto", "Suspended_user", "Stopping", or "Restarting" state and the *kill* parameter is 0 or omitted.
- An error occurs while stopping the process.

3.2.6 Timer Events

3.2.6.1 The process_transition Timer

When the **process_transition** timer expires, the protocol server MUST verify whether all the processes that are in the "Stopping" or "Restarting" states, as specified in section 2.2.1.2, were stopped. If a process has been in the same state for longer than a specified amount of time, the protocol server MUST kill the process.

If a process that is in the "Stopping" state is killed or the protocol server determines that it is not running, the protocol server changes the state in **process_state** from "Stopping" to "Suspended_user".

If a process that is in the "Restarting" state is killed or the protocol server determines that it is not running, the protocol server removes the process from **process_state**. The process will be started by the **process_alive** timer as specified in section 3.2.6.2.

3.2.6.2 The process_alive Timer

When the **process_alive** timer expires, the protocol server MUST verify all processes that are specified in the **node_config** element. If a process is not in **process_state**, the protocol server MUST verify that the process exists and start it if the process is not running.

3.2.7 Other Local Events

None.

4 Protocol Examples

The examples in this section contain only the **XML** body for each message. For an example of an HTTP header, see [XML-RPC].

4.1 GetStatus

The following example shows the request that the protocol client sends to the protocol server to call the **GetStatus** method.

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

    <params>
  </params>
</methodCall>
```

The following example shows the response from the protocol server.

```
<?xml version='1.0'?>
<methodResponse>
  <params>
    <param>
      <value><struct>
        <member>
          <name>df</name>
          <value><struct>
            <member>
              <name>index</name>
              <value><struct>
                <member>
                  <name>mountpoint</name>
                  <value><string>c:\\</string></value>
                </member>
                <member>
                  <name>disktotal</name>
                  <value><string>268433354752</string></value>
                </member>
                <member>
                  <name>dir</name>
                  <value>
                    <string>c:\\SPSearchExtended\\data\\data_index</string>
                  </value>
                </member>
                <member>
                  <name>diskfree</name>
                  <value><string>246887350272</string></value>
                </member>
              </struct></value>
            </member>
            <member>
              <name>diskfree</name>
              <value><string>246887350272</string></value>
            </member>
            <member>
```

```
  <name>fixml</name>
  <value><struct>
    <member>
      <name>mountpoint</name>
      <value><string>c:\\</string></value>
    </member>
    <member>
      <name>disktotal</name>
      <value><string>268433354752</string></value>
    </member>
    <member>
      <name>dir</name>
      <value>
        <string>c:\\SPSearchExtended\\data\\data_fixml</string>
      </value>
    </member>
    <member>
      <name>diskfree</name>
      <value><string>246887350272</string></value>
    </member>
  </struct></value>
</member>
<member>
  <name>disktotal</name>
  <value><string>268433354752</string></value>
</member>
<member>
  <name>var</name>
  <value><struct>
    <member>
      <name>mountpoint</name>
      <value><string>c:\\</string></value>
    </member>
    <member>
      <name>disktotal</name>
      <value><string>268433354752</string></value>
    </member>
    <member>
      <name>dir</name>
      <value><string>c:\\SPSEAR~1\\var</string></value>
    </member>
    <member>
      <name>diskfree</name>
      <value><string>246887350272</string></value>
    </member>
  </struct></value>
</member>
<member>
  <name>data</name>
  <value><struct>
    <member>
      <name>mountpoint</name>
      <value><string>c:\\</string></value>
    </member>
    <member>
      <name>disktotal</name>
      <value><string>268433354752</string></value>
    </member>
    <member>
```

```
                <name>dir</name>
                <value><string>c:\\SPSEAR~1\\data</string></value>
              </member>
              <member>
                <name>diskfree</name>
                <value><string>246887350272</string></value>
              </member>
            </struct></value>
          </member>
        </struct></value>
      </member>
      <member>
        <name>FASTSEARCH</name>
        <value><string>c:\\SPSEAR~1</string></value>
      </member>
    </struct></value>
  </param>
 </params>
</methodResponse>
```

4.2 StartProcess

The following example shows a **StartProcess** request that starts three processes: **sprel**, **webanalyzer**, and **qrserver**.

```
<?xml version='1.0'?>
<methodCall>
  <methodName>StartProcess</methodName>
  <params>
    <param>
      <value><array><data>
        <value><string>sprel</string></value>
        <value><string>webanalyzer</string></value>
        <value><string>qrserver</string></value>
      </data></array></value>
    </param>
  </params>
</methodCall>\n"
```

The following response shows that the protocol server returned a status code of 1 for all three processes, but the text description shows that **sprel** and **webanalyzer** were started successfully, and **qrserver** was already running.

```
<?xml version='1.0'?>
<methodResponse>
  <params>
    <param>
      <value><array><data>
        <value><array><data>
          <value><string>sprel</string></value>
          <value><int>1</int></value>
          <value><string>Started sprel (PID: 360)</string></value>
        </data></array></value>
        <value><array><data>
          <value><string>webanalyzer</string></value>
          <value><int>1</int></value>
```

```
            <value><string>Started webanalyzer (PID: 932)</string></value>
          </data></array></value>
          <value><array><data>
            <value><string>qrserver</string></value>
            <value><int>1</int></value>
            <value><string>Process qrserver is already running</string></value>
          </data></array></value>
        </data></array></value>
      </param>
    </params>
  </methodResponse>\n"
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: 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.

7 Change Tracking

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

8 Index