[MS-FSWASDS]:
WebAnalyzer/SPRel Data Serving 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	Minor	Updated the technical content
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the WebAnalyzer/SPRel Data Serving Protocol (WASDS), used to improve search relevance by retrieving metadata about search items before they are indexed. Protocol clients use this lookup protocol to retrieve information from a remote protocol server where the meta information is associated with a unique key.

1.1 Glossary

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

big-endian
marshal
unmarshal

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

abstract object reference (AOR)
base port
Cheetah checksum
data distribution function
FAST Search Interface Definition Language (FSIDL)
name server
server interface

The following terms are specific to this document:

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

1.2 References

1.2.1 Normative References

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

[MS-FSCDCFG] Microsoft Corporation, "Component Distribution Configuration File Format Specification", November 2009.

[MS-FSCHT] Microsoft Corporation, "Cheetah Data Structure", November 2009.

[MS-FSMW] Microsoft Corporation, "Middleware Protocol Specification", November 2009.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt

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

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary", June 2008.

1.3 Protocol Overview (Synopsis)

This protocol is a remote method lookup protocol that receives requests from protocol clients to query a remote protocol server for records in a database table. Records contain metadata that is defined in a schema, and each record is identified by a unique key.

For a large number of keys or a large body of metadata, the database can be partitioned across multiple protocol servers. The protocol client chooses the appropriate protocol server based on a prioritized **data distribution function**. To reduce the number of requests and responses, the protocol client can request several keys in one request. The following overview diagram shows a setup with one protocol client and four protocol servers, where the servers are split into two partitions (partition 0 and 1).

For failure resilience and performance, one backup protocol server can be associated with each partition. To the protocol client, the master and backup server are fully interchangeable, and the client can choose which of the two servers to contact within a partition, for example by alternating between the two for load-balancing purposes. In addition, if a particular server in a partition becomes faulty or unreachable, the client can choose to not contact that server.

The partitioning and replication in this protocol is very simplistic: the servers do not detect each other, and only the client stores the full set of server nodes. As shown in the following diagram, the client has contacted the primary node (node 0_0) in partition 0 and the backup node (node 1_1) in partition 1. It might have contacted the backup node if, for example, the primary node in partition 1 (node 1_0) was unresponsive.

This protocol cannot update the data on the servers. The data on the servers is populated out of band by applying the data distribution function on input data and starting each server with its own copy of the partition data; therefore, populating data on the servers is not covered in this protocol.

The schema of the data is implementation specific, but note that there are reserved attributes, defined in section 2.2.

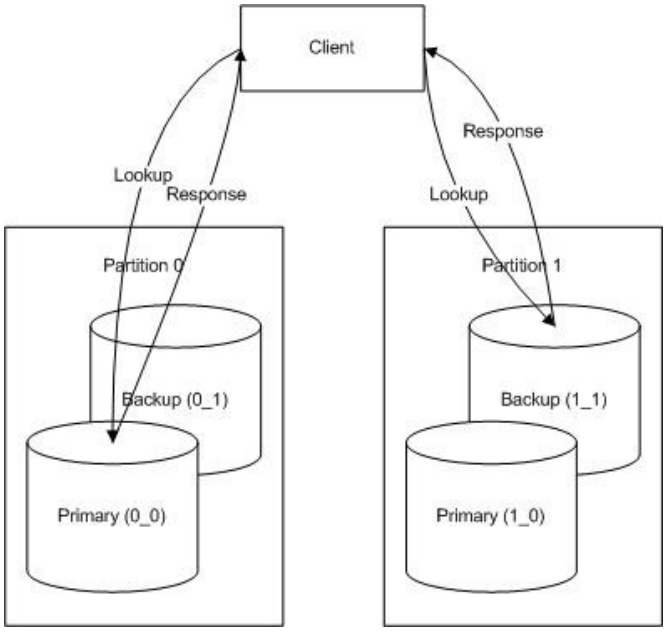

Figure 1: Overview – A protocol client querying two replicated partitions

1.4 Relationship to Other Protocols

Interfaces in this protocol are written in the **FAST Search Interface Definition Language (FSIDL)** described in [MS-FSMW]. This protocol relies on the Cheetah Data Format Specification, as described in [MS-FSCHT], for serializing custom data types and the Middleware Protocol as described in [MS-FSMW] for transport, binding, and name resolution.

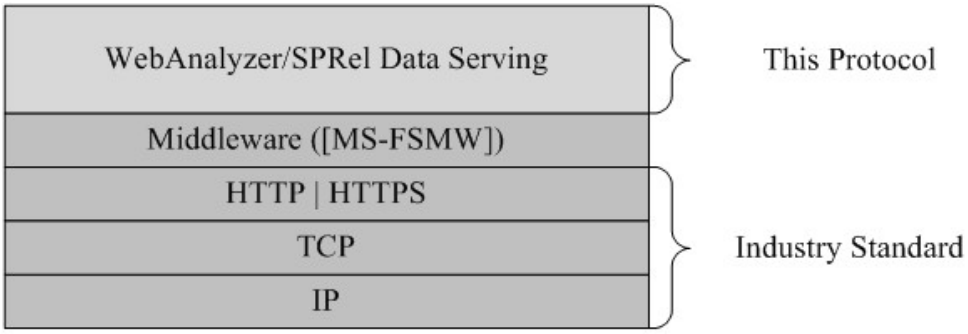

Figure 2: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

This protocol depends on the protocol described in [MS-FSMW]. The initialization steps are described in section 3.1.3.7.

1.6 Applicability Statement

This protocol is applicable for retrieving static attribute data stored on a remote protocol server. The protocol is suitable for retrieval of static attribute data that is not updated frequently and where a

simplistic data distribution scheme and failure resiliency can be applied. This protocol has no provisions for updates, transactions, or synchronization.

1.7 Versioning and Capability Negotiation

This specification covers versioning issues in the following areas:

- Version information: Versioning is described in [MS-FSMW], but for **marshaled** types the **Cheetah checksum** and type identifiers are described in section 2.2. Versioning is determined by the **server interface** name and server interface version strings that are registered with the **name server** by the protocol client and protocol server, as described in the full FSIDL in section 6.
- Capability negotiation: There are no provisions for capability negotiation in this protocol.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

This protocol MUST use the Middleware Protocol, as specified in [MS-FSMW], to transmit method requests and responses.

Transport security is provided in the Middleware Protocol. All security related parameters are controlled by the Middleware Protocol.

Byte ordering is defined in the [MS-FSMW] and [MS-FSCHT], unless explicitly stated otherwise.

2.2 Common Data Types

Data marshaling and **unmarshaling (1)** MUST be performed as specified in [MS-FSCHT]. The FSIDL and Cheetah entities for this protocol are specified in section 6 section 7.

The FSIDL data types are specified in [MS-FSMW]. The mapping of FSIDL types to native types is also specified in [MS-FSMW].

Custom data types and Cheetah entities MUST be as specified in the Cheetah Data Format Specification [MS-FSCHT] section 2. The version number for the Cheetah messages MUST contain the constant value of **-465339615**. The type identifier for the Cheetah entities MUST be as specified in the following table.

Cheetah entity	Type identifier
db_field	0
db_record	1
db_recordset	2
db_recordsetlist	3
db_keylist	4
db_stringfield	12

The data served in this protocol contains sets of records, as defined by the **db_recordsetlist** data type defined in 2.2.4. The schema of the data served is implementation specific. There is, however, a set of record attribute names that are reserved, as specified in the following table.

Reserved attribute names
Status

Note that exceptions do not have type identifiers, and are marshaled as defined in [MS-FSMW] section 3.1.4.10.

2.2.1 db_field

This data type contains the name of an attribute in a record. The structure of this data type is specified as follows:

```
entity db_field {
    attribute string name;
};
```

name: This MUST be a string as defined in [MS-FSCHT] section 2.2

2.2.2 db_record

This data type contains the attributes that define a record. The structure of this data type is specified as follows:

```
entity db_record {
    collection db_field fields;
};
```

fields: This MUST be a **db_field** collection. The **db_field** data type is defined in section 2.2.1; the collection is defined in [MS-FSCHT] section 2.3.3.

2.2.3 db_recordset

This data type contains a key-records relationship. If the key exists in the database, the key MUST map to only one **db_recordset** data type, where the **db_recordset** contains one or more records. The structure of this data type is specified as follows:

```
root entity db_recordset {
    attribute string key;
    collection db_record records;
};
```

key: This MUST be a string, as specified in [MS-FSCHT] section 2.2.

records: This MUST be a **db_record** collection. The **db_record** data type is defined in section 2.2.2; the collection is defined in [MS-FSCHT] section 2.3.3.

2.2.4 db_recordsetlist

This data type contains a collection of **db_recordset** entities. The structure of this data type is specified as follows:

```
root entity db_recordsetlist {
    collection db_recordset recordsets;
};
```

recordsets: This MUST be a **db_recordset** collection. The **db_recordset** data type is defined in section 2.2.3; the collection is defined in [MS-FSCHT] section 2.3.3.

2.2.5 db_keylist

This data type contains multiple keys in a collection. The structure of this data type is specified as follows:

```
root entity db_keylist {
```

```
    collection string keys;
};
```

keys: This MUST be a string collection. The string data type is defined in [MS-FSCHT] section 2.2; the collection is defined in [MS-FSCHT] section 2.3.3.

2.2.6 db_stringfield

This data type represents a string value of an attribute. Note that the attribute name is inherited from the **db_field** data type, defined in section 2.2.1. All attributes communicated in this protocol MUST be of this type. The structure of this data type is specified as follows:

```
entity db_stringfield : db_field {
    attribute string value;
};
```

value: This MUST be a string as defined in [MS-FSCHT] section 2.2.

2.2.7 internal_error

This data type represents an exception that implies that an internal error occurred. The structure of this data type is specified as follows:

```
exception internal_error {
    string error;
    string traceback;
};
```

error: This MUST be a string as defined in [MS-FSCHT] section 2.2.

traceback: This MUST be a string as defined in [MS-FSCHT] section 2.2.

2.2.8 unknown_table_error

This data type represents an exception that implies that an attempt was made to open a database table that does not exist. The structure of this data type is specified as follows:

```
exception unknown_table_error {
    string table;
};
```

table: This MUST be a string as defined in [MS-FSCHT] section 2.2.

2.2.9 key_not_found

This data type represents an exception that an attempt was made to look up a key that did not exist in the database. The structure of this data type is specified as follows:

```
exception key_not_found {
    string key;
};
```

key: This MUST be a string as defined in [MS-FSCHT] section 2.2.

3 Protocol Details

3.1 storageservice::cache_manager Server Details

3.1.1 Abstract Data Model

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

This protocol is stateless in the sense that no state is preserved across RPC method calls. The state of the protocol server is specified by the following:

Schema: The database sends data in a fixed schema that is specified in section 3.1.3.1.

Database table: The database sends data from a database table whose name is available to both the protocol client and the protocol server. The database table names are specified in section 3.1.3.2.

Partition identifier: If the dataset is split into multiple disjoint subsets, a dedicated protocol server is associated with each partition by the unique partition identifier for the partition. Partition identifiers are specified in section 3.1.3.3.

Replication identifier: If data is replicated within a partition, the replication identifier differentiates the master from the backup protocol server. Replication identifiers are specified in section 3.1.3.4.

Data distribution function: When there are multiple partitions, data is partitioned between protocol servers according to a fixed data distribution function. The data distribution function is specified in section 3.1.3.5.

3.1.2 Timers

None.

3.1.3 Initialization

3.1.3.1 Schema

The metadata schema used for a request/response pair is implementation specific and MUST be identifiable by the protocol client and protocol server.

3.1.3.2 Database Tables

Protocol servers MUST have one or more named database tables from which to send data. The database name is implementation specific, and MUST be identifiable by both the protocol server and protocol clients. The database name MUST be a string as defined in [MS-FSCHT] section 2.1.

3.1.3.3 Partition Identifier

If the dataset is large, it can be partitioned with a data distribution function so that multiple protocol servers can send data from their own partitions to the protocol client. Partitioning the protocol

servers creates an ordered finite set of partitions. Each partition MUST be assigned an integer value that specifies its position in the ordered set, where **0** represents the first partition. The partition identifier MUST be **0** if there is only one partition. The diagram in section 1.3 earlier in this document shows a setup with two partitions, and they have the partition identifier **0** and **1**, respectively.

The number of partitions and partition identifiers are defined by the contents of the Component Distribution Configuration file, defined in [MS-FSCDCFG]. The set of partitions is defined by the set of <CT_host> elements containing <CT_webanalyzer> elements, defined in [MS-FSCDCFG] section 2.3.2 and [MS-FSCDCFG] section 2.3.8, respectively. Partitions are numbered in the order they appear in the Component Distribution Configuration file from zero to *n*, where *n* is the number of partitions. The name of the host that a partition belongs to is defined by the **name** attribute of the respective <CT_host> element.

3.1.3.4 Replication Identifier

Replication is enabled when the Component Distribution Configuration file, defined in [MS-FSCDCFG], has the **redundant-lookup** attribute set in the <CT_webanalyzer> elements. When replication is used, a primary protocol server exists for each partition in addition to a backup protocol server. Primary protocol servers MUST have the replication identifier **0** (zero), and backup protocol servers MUST have the replication identifier **1**. The diagram in section 1.3 earlier in this document shows a setup with two replicated partitions, where the primary and backup protocol servers have the replication identifier **0** and **1**, respectively.

Note that the primary and backup protocol servers are always located on different physical hosts, to provide independent failure. As explained in section 3.1.3.3, a protocol server hosts a given numbered partition *p* from a set of *n* partitions. When replication is enabled, a backup protocol server hosting partition *p+1 modulo n* MUST exist on the same physical host.

3.1.3.5 Data Distribution Function

The protocol client MUST determine where information resides when multiple partitions are associated with protocol servers, given the definitions. The protocol client associates a partition with a key by applying a distribution function to the key for the metadata to retrieve. This function MUST also be used when distributing data to the server nodes.

Input parameters for the distribution function MUST be the key and the ordered set of protocol servers. The function MUST select an instance in the set of protocol servers. The number of protocol servers and their ordering are defined by their partitions and partition identifiers, as defined in section 3.1.3.3.

The data distribution function MUST first produce an MD5 digest of the key. MD5 digests are defined in [RFC1321]. The digest MUST be interpreted as a **big-endian** 128-bit unsigned integer. An array index value MUST then be produced by calculating the value of the checksum modulo the number of protocol servers in the server set. Lastly, the protocol server is chosen by using the array index value to select a server from the ordered set, where the first array index is zero.

The pseudocode for the data distribution function is specified in the following code. Inputs are important to resolving and ordering the set of protocol servers in an array:

```
-- Initialize values
SET database_key TO the key that should be looked up
SET servers TO ordered array of available servers
SET number_of_servers TO servers.length

-- Obtain the MD5 checksum
```

```
CALL md5_init RETURNING md5_state
CALL md5_append WITH md5_state, database_key RETURNING md5_state
CALL md5_finish WITH md5_state RETURNING raw_digest

-- Convert the MD5 checksum to a 128 bit unsigned integer
CALL read_uint128be_from_buffer WITH raw_digest RETURNING uint128_digest

-- Obtain array index
CALL modulo WITH uint128_digest, number_of_servers RETURNING array_index


-- Index into array to obtain server that hosts the key
SET server TO value at array_index in servers array
```

3.1.3.6 Registered Server Object Name

The WebAnalyzer protocol server registers its server object in the Middleware Protocol name server with a fixed-name prefix, a partition identifier, and a replication identifier. The fixed-name prefix MUST be in the following format: **fds/walookupdb**

Partitioning is specified in section 3.1.3.3, and replication in section 3.1.3.4. The partition identifier and the replication identifier MUST be a fixed-name prefix, separated by an underscore, as specified in the following format: *<partition ID>*_*<replication ID>*.

The number of protocol servers and their names is obtained by the file format specified in [MS-FSCDCFG], as defined in sections 3.1.3.3 and 3.1.3.4. Protocol servers MUST be differentiated by their partition identifiers. The assembled protocol server name MUST contain the partition and replication identifier in the following format: **fds/walookupdb***<partition ID>*_*<replication ID>*. If there is only one protocol server, the assembled protocol server name MUST be **fds/walookupdb0_0**. The full FSIDL for naming server objects is specified in [MS-FSMW].

3.1.3.7 Middleware

The protocol server MUST use the Middleware Protocol bind method to register its **storageservice::cache_manager** server object in the Middleware Protocol name server. This is achieved by calling the bind method on the Middleware Protocol name server, as specified in [MS-FSMW] section 3.4.4.2. The parameter to this method is constructed by creating an **abstract object reference** that encapsulates the following parameters.

Correspondingly, the protocol client can obtain the abstract object reference for a protocol server. This is achieved by calling the resolve method on the Middleware Protocol name server, as specified in [MS-FSMW] section 3.4.4.1. The parameters to this method are the **name**, **interface_type** and **version** parameters.

host: This MUST be a string that contains the host name of the server object on the protocol server. The value is implementation specific.

port: This MUST be an integer value that contains the port number of the server object on the protocol server. The value is **base port** + 390.

interface_type: This MUST be a string that contains the literal **storageservice::cache_manager**.

interface_version: This MUST be a string that contains the literal **5.1**.

object_id: This MUST be an integer that is unique for each server object.

name: This MUST be a string that contains the literal **fds/walookupdb<P>_<R>**, where *P* is the partition identifier specified in section 3.1.3.3 and *R* is the replication identifier, as specified in section 3.1.3.4.

3.1.4 Message Processing Events and Sequencing Rules

Message processing MUST be performed as defined in the Middleware Protocol defined in [MS-FSMW], with custom data types marshaling as defined in [MS-FSCHT].

The protocol server interface includes the methods specified in the following table.

Method	Description
access_factory	Performed by the protocol client to initiate the protocol.
get_random_read	Instructs the protocol server to open the named database table.
get_list	Instructs the protocol server return the set of records identified by the given set of keys.

The client MUST initiate the protocol by obtaining the abstract object reference for the protocol server by issuing a resolve command to the Middleware Protocol name server.

The protocol client MUST then send an access_factory message, defined in section 3.1.4.1, to obtain an object with which it can request a database to be opened.

The protocol client MUST then send a get_random_read message, defined in section 3.1.4.2, which specifies the which database the protocol server MUST open.

Lastly, the protocol client proceeds to send a sequence of get_list messages, defined in section 3.1.4.3.

Generic Middleware Protocol exceptions can be thrown from any method to return error messages, as specified in [MS-FSMW] section 3.1.1. Otherwise, any custom exceptions raised are specified for each message.

3.1.4.1 Receiving an access_factory Message

The **access_factory** method MUST initiate the protocol by creating a new **access_factory** server object for the protocol client. The returned object is used to open the database.

```
db_access_factory access_factory();
```

Return Values: This MUST be an instance of a **db_access_factory** object, which is used to open the database.

Exceptions Thrown: This MUST NOT throw any exceptions beyond those thrown by the underlying Middleware Protocol, as specified in [MS-FSMW] section 3.

3.1.4.2 Receiving a get_random_read Message

The **get_random_read** method in the **db_access_factory** interface opens the database to be queried. This method MUST return a new **db_random_read** server object identified by the table name.

```
db_random_read db_access_factory::get_random_read(in string table)
```

table: This MUST contain the name of the table to open. It MUST be a string, as specified in [MS-FSCHT] section 2.1.

Return Values: This MUST be an instance of a **db_random_read** interface that is used to query a database.

Exceptions Thrown: If the named database table does not exist, the protocol server MUST throw the **unknown_table_error** exception. The **table** attribute in the exception MUST be set to an informative message, and MUST NOT be parsed. The definition of the **unknown_table_error** exception is specified in section 2.2.8.

3.1.4.3 Receiving a get_list Message

The **get_list** method in the **db_random_read** interface queries the database on the protocol server for a sequence of keys. The protocol client MUST verify that the keys are associated with the protocol server if there are multiple partitions.

```
db_recordsetlist get_list(in db_keylist keys)
```

keys: This MUST be a set of keys to retrieve from the database. The set MUST contain one or more keys, as specified in section 2.2.5.

Return Values: This MUST be a set of records that was retrieved from the database. The set MUST contain one **db_recordset** for all keys in the *keys* parameter. The format of the **db_recordsetlist** structure is specified in section 2.2.4. The schema of the records is implementation specific, although the **status** attribute name is reserved, as defined in section 2.2. When a key is found in the database, the **status** attribute of the corresponding record in the **db_recordsetlist** data type MUST be set to the literal **'ok'**. Conversely, when a key is not found, an empty dummy record MUST be added to the **db_recordsetlist** data type with its **status** attribute set to the literal **'not found'**.

Exceptions Thrown: The protocol server MUST NOT throw the **key_not_found** exception, defined in section 2.2.9, if one or more keys do not exist. Instead the **status** attribute of each record of the **db_recordsetlist** data type defines whether a key was present, as described in the preceding paragraph.

The protocol server MUST throw the **internal_error** exception, defined in section 2.2.7, if it cannot perform the lookups requested. The protocol server MUST distinguish between the error states defined in the following table, and set the **error** and **traceback** attributes accordingly.

Error condition	Error attribute value	Traceback attribute value
The table contains no rows.	**The table being served is empty**	****empty****
A data structure in the database was empty.	**The data structure is empty**	****ds_empty****
Any other condition.	**An unexpected error occurred.**	unspecified

For the last error condition listed in this table, note that there is a spelling error in the **error** attribute value ("occured" is used, rather than "occurred"). Also note that the **traceback** attribute value MUST be set to a stack trace that describes the error condition, but that does not further define its formatting. The **traceback** attribute value MUST NOT be parsed.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

4 Protocol Examples

4.1 Sequence Diagram

The following sequence diagram shows a protocol client, a name service, and two protocol servers. Each server serves its own partition of the full set, identified by its name (fds/walookupdb0_0 and fds/walookupdb0_1).

The protocol servers initiate by registering themselves under a name denoting their partition and replication identifier. The protocol client then resolves the server names and contacts the servers. The **access_factory** and **get_random_read** messages open the database to query, and the **get_list** message retrieves the records the protocol client queries for.

In this diagram, the protocol client completes all communication with the first protocol server before contacting the other, but this is not a requirement of this protocol.

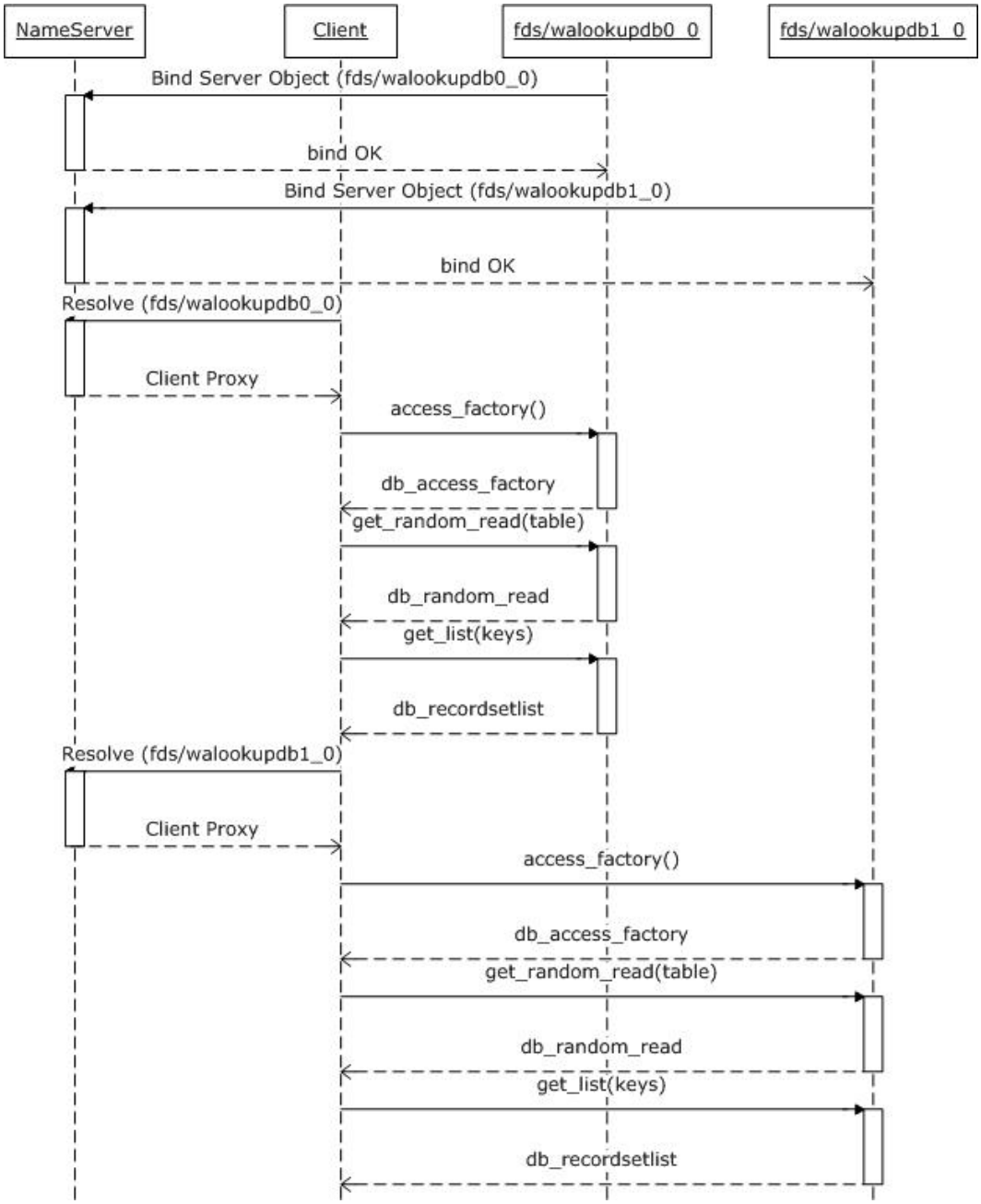

Figure 3: Name server and two protocol server partitions

4.2 Client-Side Lookup

This example describes the **get_list** method of the **db_random_read** interface, as specified in section 3.1.4.3. The protocol client sends a set of keys to the database protocol server, which responds with a set of records it retrieved. This example contains only one protocol server; if there were more protocol servers, they would register unique server names as specified in section 3.1.3.6.

The protocol client selects the appropriate protocol server by mapping the keys to protocol server names using the data distribution function specified in section 3.1.3.5.

The implementation-specific location of the shared name server and the symbolic name of the WebAnalyzer server object are specified as a part of developing the implementation. When the implementation is invoked, the protocol server creates a server object that implements the **cache_manager** interface and registers it with the name server as specified in section 3.1.3.7. The protocol client then acquires a client proxy for this **cache_manager** interface by resolving the server object with the name server, as described in section 3.1.3.7.

Next, the protocol client calls the **access_factory** method on the **cache_manager** object, which returns a client proxy of a factory object that is used to open the database. The protocol client opens the database by calling a **get_random_read** method on the factory object with the table name it wishes to query. The protocol server returns a **db_random_read** client proxy object that the protocol client uses to perform queries, in association with the **get_list** method described in section 3.1.4.3.

4.3 Protocol Server Initialization

The following pseudocode describes how a protocol server creates an abstract object reference that it registers in the Middleware Protocol name server with the Middleware Protocol bind method.

```
SET server_object_instance TO INSTANCE OF storageservice::cache_manager SERVER OBJECT
SET server_object_host TO "myserver.contoso.com"
SET server_object_port TO "1234"
SET server_object_interface_type TO "storageservice::cache_manager"
SET server_object_interface_version TO "5.1"
SET server_object_name TO "fds/walookupdb0_0"
SET server_object_aor TO server_object_host, server_object_port,
server_object_interface_type, server_object_interface_version AND server_object_name
CALL nameserver.bind WITH server_object_instance AND server_object_aor
```

4.4 Protocol Client Initialization

The following pseudocode describes how a protocol client obtains an abstract object reference from the Middleware Protocol name server that represents the protocol server.

```
SET server_object_name TO "fds/walookupdb0_0"
SET server_object_type TO "storageservice::cache_manager"
SET server_object_version TO "5.1"
SET table_name TO "default"
SET keys TO set_of_keys_to_look_up
CALL nameserver.resolve WITH server_object_name, server_object_type AND server_object_version
RETURNING server_object_aor
```

4.5 Protocol Client Database Query Message

The following pseudocode describes how a protocol client uses the abstract object reference obtained in section 4.4 to open a database on the server. The database name is contained in the **table_name** variable. The client has a set of keys to look up in the **set_of_keys_to_look_up** variable, and concludes by printing the values returned by the server for this set.

```
CALL server_object_aor.access_factory RETURNING db_access_factory_client_proxy
CALL db_access_factory_client_proxy.get_random_read WITH table_name RETURNING
db_random_read_client_proxy
CALL db_random_read_client_proxy.get_list WITH set_of_keys_to_look_up RETURNING
db_recordsetlist
FOR db_recordset in db_recordsetlist
    print db_recordset
END FOR
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full FSIDL

For ease of implementation, the full FSIDL is provided below. The syntax uses the IDL syntax extensions described in [MS-FSMW].

```
module cht {
    module storageservice {
        typedef sequence<octet> cheetah;
        typedef cheetah db_recordsetlist;
        typedef cheetah db_keylist;
    };
};

module interfaces {
    module storageservice {
        interface db_random_read;
        interface db_access_factory;
        interface cache_manager;

        exception internal_error {
        string error;
        string traceback;
     };

     exception unknown_table_error {
        string table;
     };

     exception key_not_found {
            string key;
     };

     interface db_random_read {
        #pragma version db_random_read 5.1
        cht::storageservice::db_recordsetlist get_list(
            in cht::storageservice::db_keylist keys)
            raises (key_not_found, internal_error);
     };

     interface db_access_factory {
         #pragma version db_access_factory 5.1
         storageservice::db_random_read get_random_read(
             in string table)
             raises (unknown_table_error);
     };

     interface cache_manager {
         #pragma version cache_manager 5.1
         storageservice::db_access_factory access_factory();
     };
    };
};
```

7 Appendix B: Full Cheetah Specification

For ease of implementation, the full all Cheetah entities are provided below. The syntax is described in [MS-FSCHT].

```
entity db_field {
    attribute string name;
};

entity db_record {
    collection db_field fields;
};

root entity db_recordset {
    attribute string key;
    collection db_record records;
};

root entity db_recordsetlist {
    collection db_recordset recordsets;
};

root entity db_keylist {
    collection string keys;
};

entity db_stringfield : db_field {
    attribute string value;
};
```

8 Appendix C: Product Behavior

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

- Microsoft® FAST™ Search Server 2010

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

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

9 Change Tracking

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

10 Index