[MS-FSCHT]:
Cheetah Data Structure

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	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	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
01/20/2012	1.5	Minor	Clarified the meaning of the technical content.
04/11/2012	1.5	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	1.5	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Cheetah Data Format Protocol. This protocol enables the serialization and deserialization of data structures. The serialized form is a compact stream of bytes for persistent storage or network transfer. This protocol specifies only the serialization and deserialization of data structures; network transport mechanisms and storage protocols are specified in other documents.

Sections 1.7 and 2 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. All other sections and examples in this specification are informative.

1.1 Glossary

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

attribute
big-endian
object
UTF-8

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

Cheetah checksum
Cheetah entity

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.

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://ieeexplore.ieee.org/servlet/opac?punumber=2355

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

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

1.2.2 Informative References

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

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

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

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

1.3 Structure Overview (Synopsis)

The purpose of this protocol is to enable the serialization of user-defined data structures. A **Cheetah entity** is a named user-defined type that contains any number of **attributes** and Cheetah collections. Attributes and Cheetah collections can be atomic types such as **int**, **float**, **bool**, **string**, or **bytearray**; or user-defined types such as enumerations and entities. An entity can inherit from another entity. Cheetah entities are used to create, serialize, and deserialize complex data structures as parameters in remote method invocations.

1.4 Relationship to Protocols and Other Structures

The Cheetah format is used by the Middleware Protocol, as described in [MS-FSMW].

1.5 Applicability Statement

This specification is applicable for remote method invocation protocols or persisted **object** trees. Its compact octet stream representation makes it applicable to wire protocols. The Cheetah binary file format is not human readable. The format does not include additional information for error detection or corruption prevention.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

None.

2 Structures

2.1 Cheetah Specifications

Cheetah entities are data types that are specified in the following Augmented Backus-Naur Form (ABNF):

```
specification : 1*definition
definition    : enum_dcl ";" / entity_dcl ";"
enum_dcl      : "enum" identifier "{" enum_list "}"
enum_list     : identifier  *( "," identifier )
entity_dcl    : "entity" identifier (inheritance)? entity_body
entity_body   : "{" *export "}"
export        : ( coll_dcl ";" ) / ( attr_dcl ";" )
inheritance   : ":" identifier
attr_dcl      : "attribute" attr_type identifier
attr_type     : atomic_type / identifier
atomic_type   : "int" / "float" / "bool" / "string" / "bytearray" / "longint"
coll_dcl      : "collection"  attr_type  identifier
identifier    : ( ALPHA / '_' ) *( ALPHA / DIGIT / '_' )
```

Instances of this grammar specify the Cheetah entities that other protocols employ. Sections 2.2 and 2.3 specify how to map a Cheetah entity to its serialized form.

For more information about the Augmented Backus-Naur Form (ABNF), see [RFC5234].

2.2 Common Data Types

The following sections specify common structures that are used to serialize Cheetah data structures. There is no fixed alignment limitation, such as a 4-byte alignment, in a Cheetah structure; all fields are contiguous regardless of length. The format supports the following primitive types as specified in [MS-DTYP].

- **INT32**
- **INT64**

The byte-ordering of **INT32** and **INT64** MUST be **big-endian**. The format supports the following type as specified in [IEEE754].

- **Float**

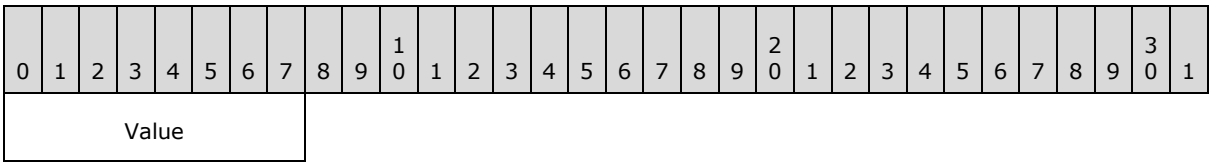
The byte-ordering of **Float** MUST be big-endian. The following table specifies the mapping between the types in the *atomic_type* rule in section 2.1 and the types that are specified in this section.

Cheetah atomic type	Computer-independent equivalent
bool	**Bool** as specified in section 2.2.1.
int	**INT32** as specified in [MS-DTYP].
longint	**INT64** as specified in [MS-DTYP].
float	**Float** as specified in [IEEE754].
string	**LengthPrefixString**, as specified in section 2.2.3.

Cheetah atomic type	Computer-independent equivalent
bytearray	**LengthPrefixByteArray**, as specified in section 2.2.2.

2.2.1 Bool

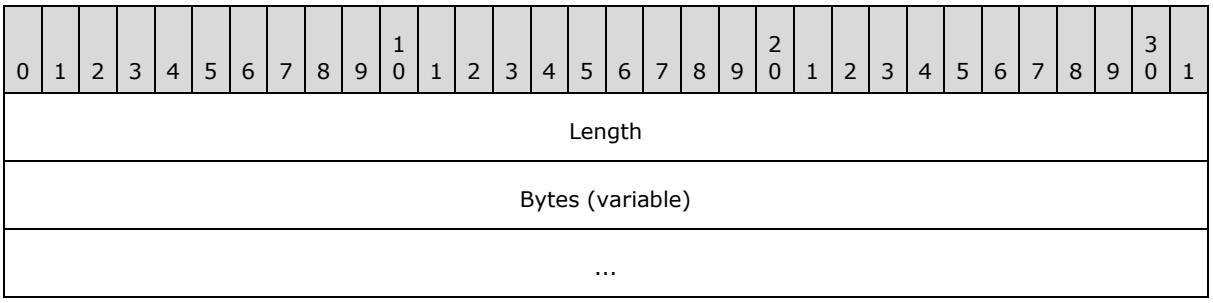
The Bool type represents a Boolean value.



Value (1 byte): A Boolean value. If **true**, the value MUST NOT equal 0. If **false**, the value MUST equal 0.

2.2.2 LengthPrefixByteArray

The **LengthPrefixByteArray** type represents an array of bytes, as specified in the following table.



Length (4 bytes): A variable of type **INT32** that represents the length of the following array of bytes.

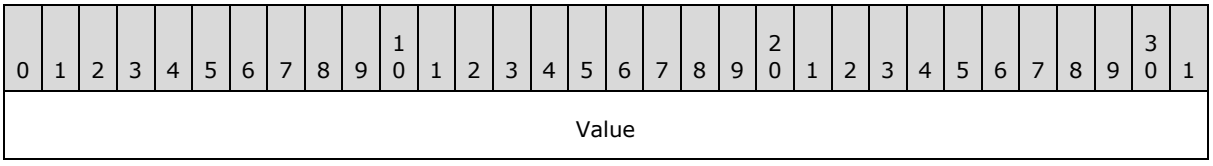
Bytes (variable): A series of bytes. The number of bytes MUST be equal to the value that is specified in the **Length** field.

2.2.3 LengthPrefixString

The **LengthPrefixString** type is the same as a **LengthPrefixByteArray**, as specified in section 2.2.2. This attribute is used instead of the **LengthPrefixByteArray** type when the encoding MUST be **UTF-8**.

2.2.4 Enum

The format of this variable is specified in the following table.



Value (4 bytes): A variable of type **INT32** that specifies the order of the enumerations.

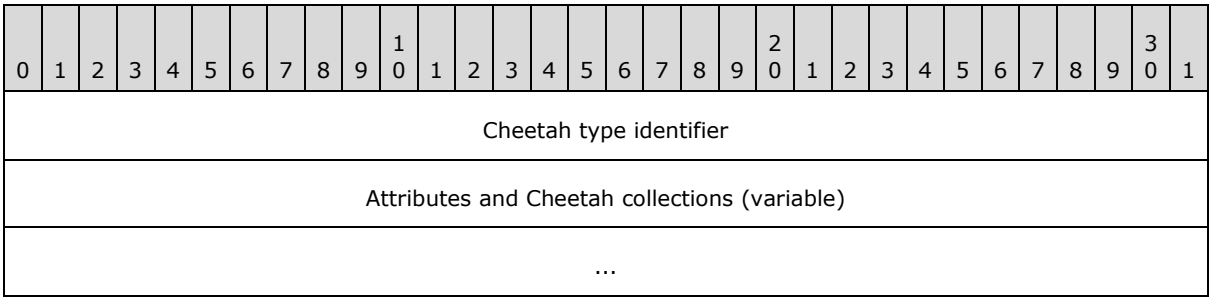
2.3 Cheetah Entity Definitions

An entity is a user-defined data type that is composed of contiguous serialized attributes and Cheetah collections. The order of the attributes in the serialized form is the same as the order in which they occur in the entity. The following table specifies the mapping between the rules in the grammar in section 2.1 and the names that are used in this section.

Cheetah grammar rule	Computer-independent equivalent
entity_dcl	Entity as specified in 2.3.1
attr_dcl	Attribute as specified in 2.3.2
coll_dcl	Cheetah Collection as specified in 2.3.3

2.3.1 Entity

Each entity type is assigned a Cheetah type identifier that specifies the type. This number MUST be supplied for each entity that is used by a protocol that employs the Cheetah protocol. The format of a Cheetah entity is specified in the following table.



Cheetah type identifier (4 bytes): MUST be a variable of type **INT32** that specifies the Cheetah type identifier of the associated entity.

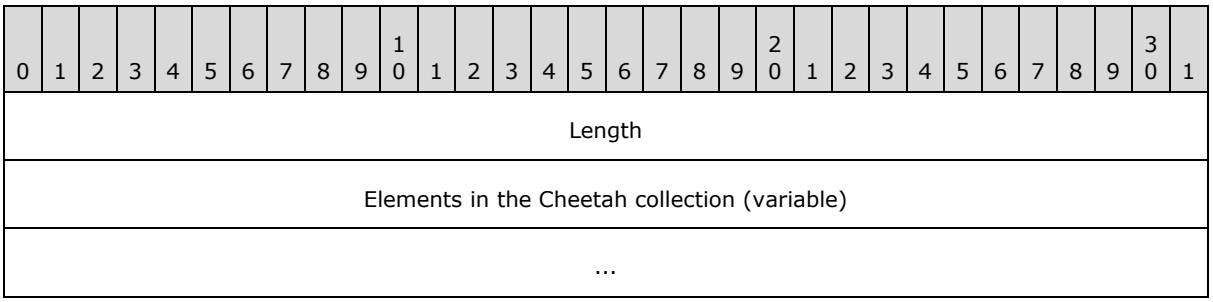
Attributes and Cheetah collections (variable): The series of attributes and Cheetah collections for this entity. The attributes and Cheetah collections in the binary format MUST appear in the same order as in the entity.

2.3.2 Attribute

An attribute MUST be of one of the types that are specified in section 2.1, or it MUST be a user-defined entity. All the attributes that are specified in the entity MUST be included in the serialization of the entity.

2.3.3 Cheetah Collection

A Cheetah collection is, in its serialized form, a length-prefixed series of atomic types or entities that are specified in the following table.

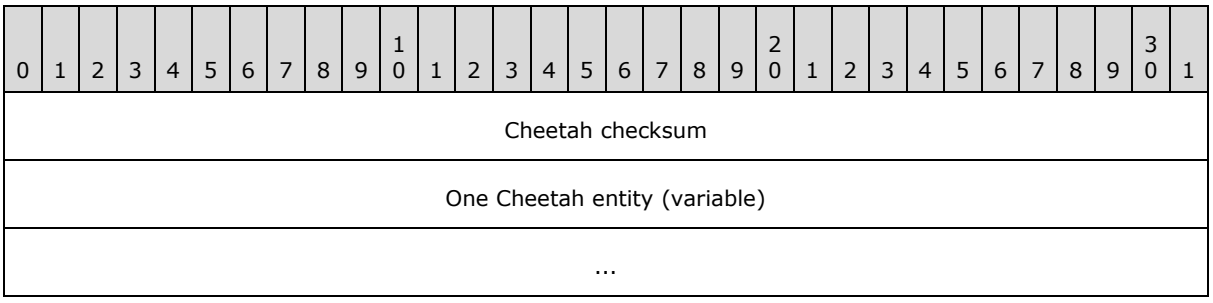


Length (4 bytes): A variable of type **INT32** that represents the number of elements in the Cheetah collection. If the length is 0, the Cheetah collection is empty and no elements follow.

Elements in the Cheetah collection (variable): A series of elements. Each element is represented according to its type. The number of elements MUST be equal to the value specified in the **Length** field.

2.4 Serialization

A serialized Cheetah entity contains the **Cheetah checksum** and the entity. This Cheetah checksum MUST be supplied by the protocol and applications in the Cheetah data format. The serialized form MUST be as specified in the following table.



Cheetah checksum (4 bytes): A variable of type **INT32** that contains the checksum. It MUST be supplied by the implementation that uses this protocol.

One Cheetah entity (variable): Exactly one Cheetah entity MUST be serialized, as specified in section 2.3.

3 Structure Examples

3.1 Cheetah Entity

3.1.1 Empty Entity

This is an example of an entity that contains no data:

```
entity my_empty_entity {
};
```

3.1.2 Adding Attributes to an Entity

Either the atomic types that are specified in section 2.1 or user-defined entities MUST be used to append attributes to an entity, as specified in the following example:

```
entity my_entity {
  attribute int my_int;
  attribute string my_name;
};
```

The *my_entity* entity in the preceding example has two members, the integer attribute *my_int* and the string attribute *my_name*. The members of a serialized entity are written to the file in the order in which they are specified. A Cheetah entity can inherit from another Cheetah entity. Section 2.4 specifies how the different types are serialized.

3.1.3 Inheritance

In the following example, the *my_inherited_entity* entity contains three members: the *my_int* and *my_name* members are inherited from the *my_entity* variable previously specified, and the *my_additional* variable. The members of this entity are written to the file in the order in which they are specified: *my_int*, *my_name,* and then *my_additional*.

```
entity my_inherited_entity : my_entity {
  attribute int my_additional;
};
```

3.1.4 Adding Cheetah Collections to an Entity

Cheetah collections of atomic types and entities are used to build tree structures, as specified in the following example:

```
entity my_tree_entity {
  collection int lots_of_ints;
  collection my_entity all_my_big_entities;
};
```

The binary file contains only one Cheetah object. Cheetah entities can contain Cheetah collections of other Cheetah entities. In the preceding example, the *my_tree_entity* entity is serialized by writing the *lots_if_ints* Cheetah collection, followed by the *all_my_big_entities* Cheetah collection. The *all_my_big_entities* Cheetah collection is of type *my_entity* and contains entities of one or both of

the following types: *my_inherited_entity* and *my_bigger_entities*. Section 2.4 specifies how the different types are serialized.

3.2 Serialized Cheetah Entity

This sample specifies how to serialize a Cheetah structure, and shows the resulting serialized form.

```
entity my_entity {
  attribute string name;
  attribute int number;
  attribute string some_text;
  attribute longint big_number;
};

entity my_list {
  collection my_entity a_list;
};
```

In this example, the *my_entity* entity is associated with a Cheetah type identifier of 0 and the *my_list* entity is associated with a Cheetah type identifier of 1. The Cheetah checksum is implementation-specific and included in the documentation that specifies that the Cheetah protocol is used. For these entities, the Cheetah checksum is 1234567.

3.3 Serialized Cheetah Collection

Building on the example described in section 3.2, the following entity of type *my_list* is named *the_list*. This entity contains a Cheetah collection of type *my_entity* that is named *a_list.* The *a_list* collection contains three entries:

```
the_list {
  a_list = {
    { name="name", number=32, some_text="this is text", big_number=1}
    { name="name", number=0, some_text="this is text", big_number=60365344270}
    { name="strange", number=-32, some_text="less text", big_number=1}
  }
}
```

The following stream of bytes represents the serialized version of the *the_list* object:

```
0000: 0012 D687 0000 0001 0000 0003 0000 0000  ................
0010: 0000 0004 6E61 6D65 0000 0020 0000 000C  ....name... ....
0020: 7468 6973 2069 7320 7465 7874 0000 0000  this is text....
0030: 0000 0001 0000 0000 0000 0004 6E61 6D65  ............name
0040: 0000 0000 0000 000c 7468 6973 2069 7320  ........this is
0050: 7465 7874 0000 000E 0E0E 0E0E 0000 0000  text............
0060: 0000 0007 7374 7261 6E67 65FF FFFF E000  ....strange.....
0070: 0000 096C 6573 7320 7465 7874 0000 0000  ...less text....
0080: 0000 0001                                ....
```

To send and receive the *the_list* object, these bytes are sent over the wire using this protocol.

3.4 Annotated Cheetah File Structure

The bytes in the section 3.3 example are mapped to the following logical structure.

```
Cheetah checksum: 1234567 (0x0012D687, INT32)
Cheetah type identifier: 1 (0x00000001, INT32)
  Length of Cheetah collection: 3 (0x0000003, INT32)
    1. element:
      Cheetah type identifier: 0 (0x00000000, INT32)
        name, LengthPrefixString:
          Length: 4 (0x0000004, INT32)
          String value: "name" (0x6E616D65, 4xbyte)
        number: 32 (0x00000020, INT32)
        some_text, LengthPrefixString:
          Length: 12 (0x000000C, INT32)
          String value: "This is text" (0x746869732069732074657874, 12*byte)
        big_number: 1 (0x0000000000000001, INT64)
    2. element:
      Cheetah type identifier:0 (0x00000000, INT32)
        name, LengthPrefixString:
          Length: 4 (0x0000004, INT32)
          String value: "name" (0x6E616D65, 4xbyte)
        number: 0 (0x00000000, INT32)
        some_text, LengthPrefixString:
          Length: 12 (0x000000C, INT32)
          String value: "This is text" (0x746869732069732074657874, 12*byte)
        big_number: 60365344270 (0x0000000E0E0E0E0E, INT64)
    3. element:
      Cheetah type identifier:0 (0x00000000, INT32)
        name, LengthPrefixString:
          Length: 7 (0x0000007, int32)
          String value: "strange" (0x737472616E6765, 7xbyte)
        number: -32 (0xFFFFFFE0, INT32)
        some_text, LengthPrefixString:
          Length: 9 (0x0000009, INT32)
          String value: "less text" (0x6C6573732074657874, 9xbyte)
        big_number: 1 (0x0000000000000001, INT64)
```

The first element in this serialized stream is the Cheetah checksum that contains the value 1234567. The next element is the Cheetah type identifier for the *my_list* entity, which contains a value of 1.

As described in section 3.2, this *my_list* object has only one Cheetah collection, the *a_list* Cheetah collection of type *my_entity*. When this Cheetah collection is serialized, the length is the first element that appears in the serialization. The length indicates that there are three elements in the *my_list* object.

After the length, the three elements of the Cheetah collection are included. Entities of type *my_entity* begin with the entity Cheetah type identifier. In this instance, the Cheetah type identifier contains a value of 0.

Next, the four elements of the *my_entity* type are serialized. The serialized form of all *my_entity* entities are identical, except for the actual data. The order is the Cheetah type identifier, followed by the four attributes of the entity. The attributes are a **LengthPrefixString** attribute, a variable of type **INT32**, a second **LengthPrefixString** attribute, and finally a variable of type INT64. The string elements are prefaced with the length of the string.

There is no alignment limitation. For example, the last 3 bytes of the word "strange" in element 2 occupy the first 3 bytes of a 4-byte area. Consequently, the next **INT32** variable does not align to any 4-byte limitation that is common in fixed-format file specifications.

4 Security Considerations

When deserializing a Cheetah data structure, the length of a **LengthPrefixByteArray**, **LengthPrefixString** or a Cheetah collection is verified against implementation-specific limits before allocating memory. There is no code in the serialized form; consequently, implementations and network transports are responsible for minimizing the possibility of denials of service and other security issues.

5 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.

6 Change Tracking

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

7 Index