[MS-FSWCU]:
WebAnalyzer/Crawler Utility Structure Specification

Revision Summary

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

Table of Contents

1 Introduction

This document specifies that enable serializing and de-serializing information associated with an octet stream.

1.1 Glossary

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

ASCII
little-endian
network byte order
Unicode
UTF-8

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

dictionary

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.

[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

[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 Structure Overview (Synopsis)

This structure specifies how to serialize and de-serialize simple and complex types, such as numerical values, text, arrays, to an octet stream. Serialization is used mainly in two settings: while transmitting information over the network, or while storing information to a file. In either case, this structure enables processes to represent internal values and structures so that other processes can read the information. This is still true even if the processes are written in different programming languages or if they have different internal representations of information values and structures.

1.4 Relationship to Protocols and Other Structures

This structure does not depend on any other protocols or structures. Protocols that depend on this structure transmit information between applications on different hardware architectures, transmit information between applications implemented in different programming languages, or some combination thereof.

1.5 Applicability Statement

This protocol is designed for serializing information to transmit over a network, or to write to files.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

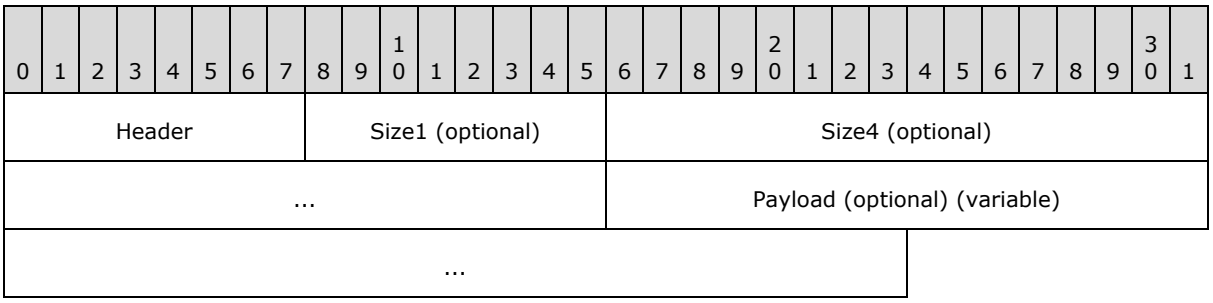
None.

2 Structures

The following table specifies the numerical values that are used in these structures, in addition to the computer-independent equivalent.

Data type	Format on the wire
double64_l	[IEEE754] double-precision (64-bit) floating point. Representation is **little-endian**.
int32_l	32-bit signed integer with two-complement signed number representation. Representation is little-endian.
int16_l	16-bit signed integer with two-complement signed number representation. Representation is little-endian.
uint8	8-bit unsigned integer.

The serialized representation of any data type MUST follow the format that is specified in the following table.



Structure fields:

Header (1 byte): Indicates which data type this packet contains.

Size1 (1 byte): Optional **uint8** field that either specifies the size of the **payload** field, depending on the **header** field, or is not used.

Size4 (4 bytes): An **int32_l** field that specifies either the value of the data, the size of the **payload** field, or is not used. The **header** field specifies whether this field is associated with the data or with the **payload** field

Payload (variable): Contains a byte stream that is the serialized data. This field MUST be present only for the appropriate headers.

2.1 Simple types

Simple types are types that cannot contain other types, such as strings and number values.

2.1.1 Floating type

This section specifies how floating type values are serialized. The values MUST meet the standard specified in [IEEE754]. Only values within a double-precision 64-bit range are supported. The format of the serialized value MUST be the following:

Header: %x66 (**ASCII** value 'f')

Size1: The size of the **payload** field in bytes. This field MUST be a **uint8** value.

Size4: Not applicable.

Payload: The value MUST be represented using an ASCII String that meets the following lexical specification.

```
floatnumber   ::=  pointfloat | exponentfloat
pointfloat    ::=  [intpart] fraction | intpart "."
exponentfloat ::=  (intpart | pointfloat) exponent
intpart       ::=  digit+
fraction      ::=  "." digit+
exponent      ::=  ("e" | "E") ["+" | "-"] digit+
digit         ::=  "0"..."9"
```

2.1.2 Integer type

This section specifies how integer values are serialized. The values MUST be within the range of a 32-bit signed integer; values outside that range are specified in section 2.1.3. The value is serialized in the following format:

Header: %x69 (ASCII value 'i')

Size1: Not applicable.

Size4: An **int32_l** field that represents the value of the integer.

Payload: Not applicable.

2.1.3 Long type

This section specifies how integer values outside the range of a 32-bit signed integer MUST be serialized. The format of the serialized value MUST be the following:

Header: %x6C (ASCII value 'l')

Size1: Not applicable.

Size4: An **int32_l** value that specifies the size of the **payload** field. This field represents the number of **int16_l** values that the **payload** field contains. This value MUST be negative if the value of the input is negative.

Payload: The numerical value MUST be represented in this field, divided into **int16_l** values that occur in **network byte order**. Each **int16_l** value represents 15 bits of the total value. This field MUST NOT be used to specify whether the input value is negative.

2.1.4 None type

This specifies how a **None** type serializes variables that have no value. The format of the serialized value MUST be the following:

Header: %x4E (ASCII value 'N')

Size1: Not applicable.

Size4: Not applicable.

Payload: Not applicable.

2.1.5 String type

The serialized representation of a string that is not **Unicode** MUST be in the following format. If the string is empty and the size is 0, then the **payload** field MUST be empty. The format of the serialized value MUST be the following:

Header: %x73 (ASCII value 's')

Size1: Not applicable.

Size4: The size of the **payload** field. This field MUST be an **int32_l** value and MUST give the number of bytes that the **payload** field contains.

Payload: The string value.

2.1.6 Unicode string type

This specifies how to serialize a Unicode string. The format of the serialized value MUST be the following:

Header: %x75 (ASCII value 'u')

Size1: Not applicable.

Size4: An **int32_l** value that represents the size of the **payload** field, specified in bytes.

Payload: The string value encoded as **UTF-8**.

2.2 Complex types

Complex types are container types that can contain other complex types or simple types.

2.2.1 Array type

The **array** type contains only the types that are specified in section 2. The format of the serialized value MUST be the following:

Header: %x5B (ASCII value '[')

Size1: Not applicable.

Size4: An **int32_l** value that specifies the number of elements that the **payload** field contains.

Payload: All values in the array. The order in which they are serialized MUST be the same before and after serializing.

2.2.2 Dictionary type

The **dictionary** type is a key/value based container type. The keys MUST always be one of the simple types that are specified in section 2.1, or the **tuple** type that is specified in section 2.2.3. A **tuple** MUST NOT be a key if it contains an array or a dictionary.

The **value** field in the dictionary entry MUST be one of the types specified in section 2. It is serialized in the following format:

Header: %x7B (ASCII value '{')

Size1: Not applicable.

Size4: Not applicable.

Payload: The last value MUST be followed by the ASCII character '0', or "%x30". If the dictionary is empty the **payload** field MUST contain only "%x30".

2.2.3 Tuple type

The **tuple** type MUST contain only types that are specified in section 2. The format of the serialized value MUST be the following:

Header: %x28 (ASCII value '(')

Size1: Not applicable.

Size4: An **int32_l** value that specifies the number of elements that the payload field contains.

Payload: All values in the **tuple**. The order in which the values are serialized MUST be the same before and after serializing.

3 Structure Examples

3.1 Simple types

3.1.1 Floating type

This section has three examples that specify how floating types are serialized. In the following table, the three input values are in the left column, and the serialized values are in the right column.

Input	Output (Hex)
1.0	%x66 %x03 %x31 %x2e %x30
2e20	%x66 %x06 %x32 %x65 %x2B %x30 %x32 %x30
2e-20	%x66 %x17 %x31 %x2E %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x39 %x65 %x2D %x30 %x32 %x30

3.1.2 Integer type

This section has four examples that specify how integer types are serialized. In the following table, the input values are in the left column, and the serialized values are in the right column.

Input	Output (Hex)
1	%x69 %x01 %x00 %x00 %x00
-1	%x69 %xFF %xFF %xFF %xFF
2147483647	%x69 %xFF %xFF %xFF %x7F
-2147483648	%x69 %x00 %x00 %x00 %x80

3.1.3 Long type

This section has four examples that specify how long types are serialized. In the following table, the input values are in the left column, and the serialized values are in the right column.

Input	Output (Hex)
1	%x6C %x01 %x00 %x00 %x00 %x01 %x00
-1	%x6C %xFF %xFF %xFF %xFF %x01 %x00
2147483648	%x6C %x03 %x00 %x00 %x00 %x00 %x00 %x00 %x00 %x02 %x00
-2147483649	%x6C %xFD %xFD %xFD %xFD %x01 %x00 %x00 %x00 %x02 %x00

3.1.4 None type

The following table specifies how **None** types are serialized. The input value is in the left column, and the serialized value is in the right column.

Input	Output (Hex)
None	%x4e

3.1.5 String type

This section has two examples that specify how non-Unicode strings are serialized. In the following table, the input values are in the left column, and the serialized values are in the right column.

Input	Output (Hex)
"hello world"	%x73 %x0B %x00 %x00 %x00 %x68 %x65 %x6C %x6C %x6F %x20 %x77 %x6F %x72 %x6C %x64
""	%x73 %x00 %x00 %x00 %x00

3.1.6 Unicode string type

This section has two examples that specify how Unicode strings are serialized. In the following table, the input values are in the left column, and the serialized values are in the right column.

Input	Output (Hex)
"hello world"	%x75 %x0B %x00 %x00 %x00 %x68 %x65 %x6C %x6C %x6F %x20 %x77 %x6F %x72 %x6C %x64
""	%x75 %x00 %x00 %x00 %x00
"æøå"	%x75 %x06 %x00 %x00 %x00 %xc3 %xa6 %xc3 %xb8 %xc3 %xa5

3.2 Complex types

3.2.1 Array type

In this example, an **array** with three entries is serialized. The entries and their corresponding values are specified in the following table, and the serialized output is specified after the table.

Input:

Value	Type
1	integer
hello world	string
2147483648	long

Output (Hex):

```
%x5B %x03 %x00 %x00 %x00 %x69 %x01 %x00 %x00 %x00 %x73 %x0B %x00 %x00 %x00 %x68 %x65 %x6C
%x6C %x6F %x20 %x77 %x6F %x72 %x6C %x64 %x6C %x03 %x00 %x00 %x00 %x00 %x00 %x00 %x00 %x02
%x00
```

3.2.2 Dictionary type

In this example, a **dictionary** with three key-value pairs is serialized. The input shown in the following table contains both the keys and values and their corresponding types. The serialized output is specified after the table.

Input:

Key	Key type	Value	Value type
integer	string	1	integer
1	integer	integer	string
hello	string	world	string

Output (Hex):

```
%x7B %x69 %x01 %x00 %x00 %x00 %x73 %x07 %x00 %x00 %x00 %x69 %x6E %x74 %x65 %x67 %x65 %x72
%x73 %x05 %x00 %x00 %x00 %x68 %x65 %x6C %x6C %x6F %x73 %x05 %x00 %x00 %x00 %x77 %x6F %x72
%x6C %x64 %x73 %x07 %x00 %x00 %x00 %x69 %x6E %x74 %x65 %x67 %x65 %x72 %x69 %x01 %x00 %x00
%x00 %x30
```

3.2.3 Tuple type

In this example a **tuple** with three entries is serialized. The entries and their corresponding values are specified in the following table, and the serialized output is specified after the table.

Input:

Value	Type
1	integer
hello world	string
2147483648	long
[1,2]	Array containing two integers

Output (Hex):

```
%x28 %x04 %x00 %x00 %x00 %x69 %x01 %x00 %x00 %x00 %x73 %x0B %x00 %x00 %x00 %x68 %x65 %x6C
%x6C %x6F %x20 %x77 %x6F %x72 %x6C %x64 %x6C %x03 %x00 %x00 %x00 %x00 %x00 %x00 %x00 %x02
%x00 %x5B %x02 %x00 %x00 %x00 %x69 %x01 %x00 %x00 %x00 %x69 %x02 %x00 %x00 %x00
```

4 Security Considerations

None.

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

S

T

U

V