Standard Specification for the **Uncomplicated Binary Exchange Format (UBEXF)**

Draft 1: Revision 05

ionogu@acm.org

ionogu@acm.org

Table of Contents

ionogu@acm.org

1. GENERAL

1.1. INTRODUCTION

1.1.1. General Information

1.1.1.1: The UBEXF is a simple format for representing and exchanging key-value pair data in simple JSON-style binary format .

1.1.1.2: UBEXF is not designed to be completely compatible with JSON. However, provided that the string length of every key in JSON document is less than 256 characters, then such document can seamlessly be converted to UBEXF and back to JSON without any alteration or structural changes.

1.1.1.3: Reference is made to the JSON standard as defined in http://json.org, henceforth known as JSON

1.1.1.4: This standard refers to UBJSON standard as defined in http://ubjson.org, henceforth known as UBJSON

1.1.1.5:

1.1.2. General Information

***

1.2. REPRESENTATION

1.2.1: A valid UBEXF must be an Object, there shall be no other type that forms a valid UBEXF on its own.

1.2.2: UBEXF is organized into structural tokens, where each token is represented by a byte. And its value maps exactly to an ASCII standard character value.

1.2.3: UBEXF is not intended to be completely human readable or human modifiable. It is intended to be Machine friendly and Parsing friendly.

1.2.4: The Use of JSON's Key-Value semantics inherits all the advantages of JSON except for it's text-wise readability and it's limitation on the string length of keys

1.2.5: The structural tokens are called markers, and each marker defines either, a the start of a type, a type, or the end of a type

ionogu@acm.org

1.3. DEFINITION OF TERMS

1.3.1. Byte:

A byte is a contiguous sequence of 8 bits. There is no notion of signess associated with it.

1.3.2. Character:

A Character is a contiguous sequence of 8 bits, otherwise known as a byte. A character is a signed byte.

1.3.3. Marker:

A marker is byte delegated by the rest of this standard, that has a structural description for the bytes proceeding it.

1.3.4. Size Marker:

A Size-marker is a Marker delegated by the rest of this standard, that binds a proceeding sequence of unsigned bytes to be read.

1.3.5. Type:

A Type is a structural requirement on the organization and translation of a sequence of bytes.

1.3.6. Payload:

A sequence of bytes proceeding a Marker that is defined to have a payload. The number of bytes is defined by the same marker

ionogu@acm.org

2. STRUCTURE

2.1. SYNTAX

2.1.1. Object:

object:

[{][}]

[{][*size...*] [[*key...*][*value...*]...][}]

[{][*width...*] [*size...*] [[*key...*][*value...*]...][}]

size:

[**I**][x]

[**J**][xx]

[**K**][xxxx]

width:

size

key:

[u][x...]

value:

lone_type
direct_type
sequence_type
container_type

lone_type:

[**n**]

[**t**]

[**f**]

ionogu@acm.org

direct_type:

[**B**][x]

[**c**][x]

[**I**][x]

[**j**][xx]

[**k**][xxxx]

[**l**][xxxx xxxx]

[**J**][xx]

[**K**][xxxx]

[**L**][xxxx xxxx]

[**d**][xxxx]

[**D**][xxxx xxxx]

sequence_type:

[**s**][[***size...***]][xxx...]

[**b**][[***size...***]][xxx...]

container_type:

homo_array

hetro_array

object

hetro_array:

[**[**][**]**]

[**[**][[***size...***]] [[[***value...***]]...] [**]**]

homo_array:

[**(**][**)**]

[**(**][*dtype*][[***size...***]] [[[***direct_type...***]]...] [**)**]

[**(**][*stype*][[***size...***]] [[[***sequence_type...***]]...] [**)**]

ionogu@acm.org

dtype => any *marker* of direct_type

stype => any *marker* of sequence_type

---

ionogu@acm.org

Types

Direct Types (requires Payload)

Type	Marker	Bytes	Usable for Size marker	Range
bool	A	1	No	*0* or *1* → *true* or *false*
char	c	1	No	ASCII codepoints (0 to 127)
uint8	I	1	Yes	0 to 255
int16	j	2	No	-32,767 to 32,767
int32	k	4	No	-2,147,483,647 to 2,147,483,647
int64	l	8	No	-4,294,967,295 to 4,294,967,295
uint16	J	2	Yes	0 to 65,535
uint32	K	4	Yes	0 to 4,294,967,295
uint64	L	8	No	0 to $1.844674407 \times 10^{19}$
float32	d	4	No	3.4E-38 to 3.4E+38 (7 digits)
float64	D	8	No	1.7E-308 to 1.7E+308 (15 digits)

Lone Types (No payload)

Type	Marker	Bytes	
null	n	1	
true	t	1	
false	f	1	

Sequence Types (requires, size-marker, size and Payload)

Type	Marker	Payload Marker		Payload
string	s	I, J or K		
binary	b	I, J or K		

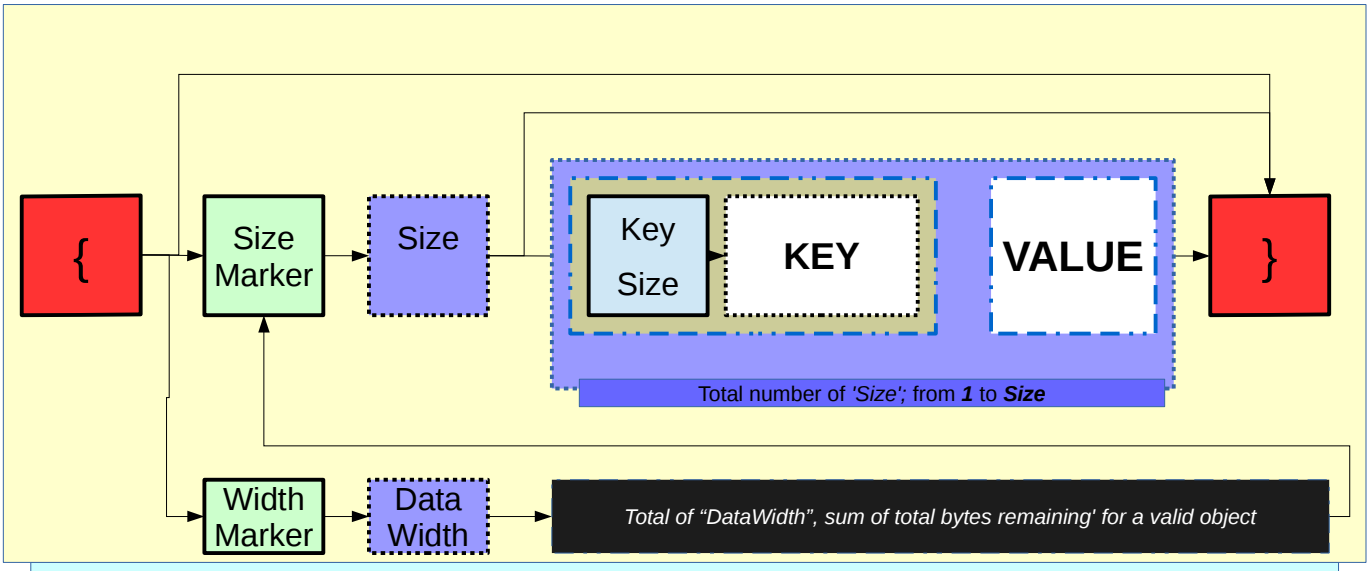
Container Types (Requires size-marker, size and payload)

Type	Marker	Payload Marker		Payload
HomoArray (homogeneous_array)	(and)	I, J or K		
HetroArray (heterogeneous_array)	[and]	I, J or K		
Object	{ and }	I, J or K		

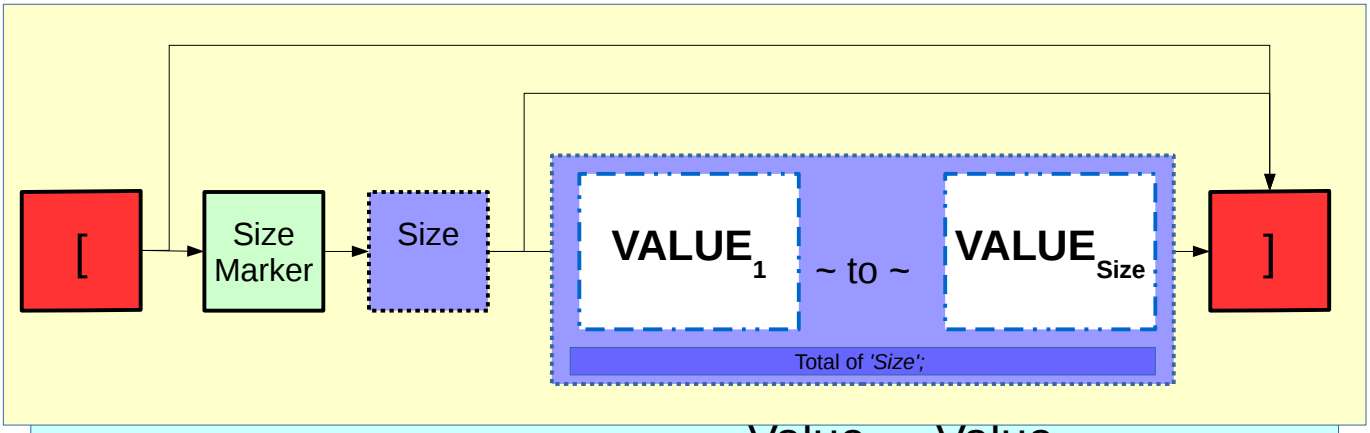
Width Type (requires, size-marker, size and Payload)

Type	Marker	Payload	
width	W	size then object	

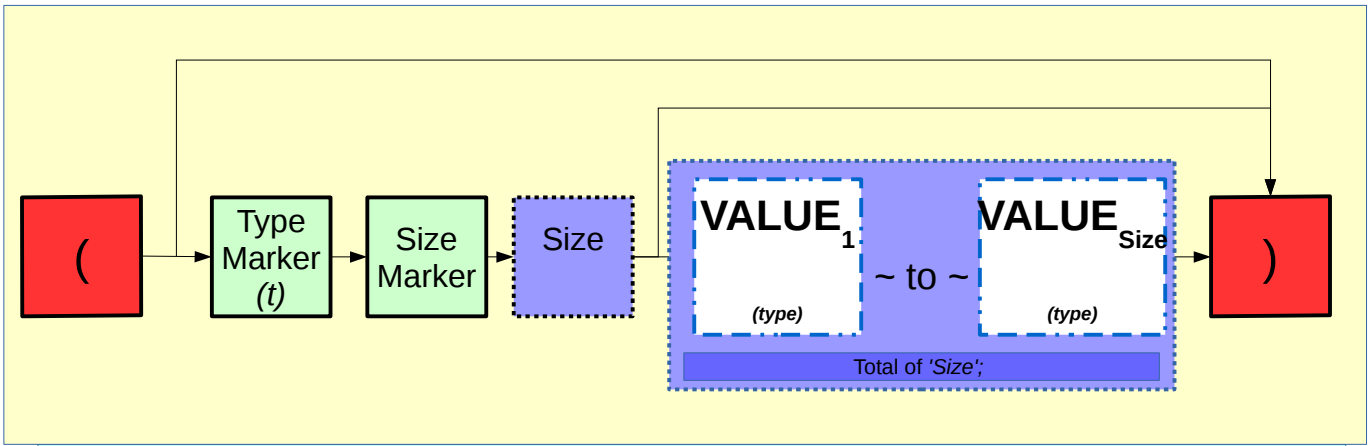
ionogu@acm.org



Object (Key : Value)



HetroArray - Hetrogeneous Array [$Value_1$ to $Value_{size}$]



HomoArray - Homogeneous Array ($Value_1$ to $Value_{size}$)

ionogu@acm.org

KNOWN LIMITATIONS OF UBEX
1. The size of an Object cannot exceed 4GB
2. An object cannot be easily modified by hand (With a TextEditor).
3. Debugging an ill-formed UBEX document is more difficult than JSON

THEORETICAL USES OF UBEX
1. Binary data exchange protocol
2. extreme tight bandwidth requirements for dynamic data forms
3. Persistent data storage format

OVERHEAD OVER JSON
Every UBEX document is always less than it's JSON equivalent
(the only known exception is a witty case such as {“”:0} and {“a”:0}, here, its UBEX equivalent will be one byte greater)
UBEX is very easy and fast to parse.
It's parsing algorithm has been described in detail

ANOTHER PROTOCOL?
Yes. UBEX was necessitated out of the need to have a very efficient and flexible format that is cross platform, with very minimum overhead with regards to many custom protocols.

UBEX is not a foolproof solution to serialization problems, it does have its limitations, but it's designed to have

RATIONALE FOR LIMITING KEY-LENGTH:
From my experience and ongoing survey, almost all JSON documents I have come across have key lengths less than 256 characters. And it's usually reasonable to have key lengths less than that value.
If this ever becomes an issue to users, they may overcome this by breaking down their keys into sets of 255 character and do multiple key-value-key-value mapping.

Example, a 255 character key length is:

twitter_api_version_32.34_name_using_the_federal_delimeters_for_a_non_currupt_twitter_api_version_32.34_name_using_the_federal_delimeters_for_a_non_currupt_twitter_api_version_32.34_name_using_the_federal_delimeters_for_a_non_currupt_twitter_api_version0

This really makes little sense for a key, except if the JSON data is in several hundreds of Gigabytes, th

ionogu@acm.org