XBin Format

The XBin (XINA Binary) format provides a XINA standard binary format for time based data files. It uses the file extension `xbin`.

The xbin format organizes **key-value** data by **time**. The data content is a series of **rows** in ascending time order, with each row having a single microsecond precision Unix time, unique within the file.

Segment Format

XBin data is often encoded in **segments**, which are defined by an initial 1, 2, or 4 byte unsigned integer length, then that number of bytes. These are referred to in this document as:

- **seg1** (up to 255 bytes)
- **seg2** (up to 65,535 bytes)
- **seg4** (up to 2,147,483,647 bytes)

If the length value of a segment is zero there is no following data and the value is considered **empty**.

Examples

The string `"foo"` has a 3 byte UTF-8 encoding: `0x66`, `0x6f`, `0x6f`.

As a seg1, this is encoded with a total of 4 bytes (the initial byte containing the length, 3):

`0x03` `0x66` `0x6f` `0x6f`

As a seg2, 5 bytes:

`0x00` `0x03` `0x66` `0x6f` `0x6f`

And as a seg4, 7 bytes:

`0x00` `0x00` `0x00` `0x03` `0x66` `0x6f` `0x6f`

Value Format

Each value starts with a 1 byte unsigned integer indicating the value type, followed by additional byte(s) containing the value itself, as applicable.

Value Type Definition

Code	Value	Length (bytes)	Description
`0`	`null`	0	literal `null` / empty string
`1`	ref dict index	1	index 0 to 255 (see below)

Code	Value	Length (bytes)	Description
2	ref dict index	2	index 256 to 65,535
3	ref dict index	4	index 65,536 to 2,147,483,647
4	true	0	boolean literal
5	false	0	boolean literal
6	int1	1	1 byte signed integer
7	int2	2	2 byte signed integer
8	int4	4	4 byte signed integer
9	int8	8	8 byte signed integer
10	float4	4	4 byte floating point
11	float8	8	8 byte floating point
12	string1	variable	seg1 UTF-8 encoded string
13	string2	variable	seg2 UTF-8 encoded string
14	string4	variable	seg4 UTF-8 encoded string
15	json1	variable	seg1 UTF-8 encoded JSON
16	json2	variable	seg2 UTF-8 encoded JSON
17	json4	variable	seg4 UTF-8 encoded JSON
18	jsonarray1	variable	seg1 UTF-8 encoded JSON array
19	jsonarray2	variable	seg2 UTF-8 encoded JSON array
20	jsonarray4	variable	seg4 UTF-8 encoded JSON array
21	jsonobject1	variable	seg1 UTF-8 encoded JSON object
22	jsonobject2	variable	seg2 UTF-8 encoded JSON object
23	jsonobject4	variable	seg4 UTF-8 encoded JSON object
24	bytes1	variable	seg1 raw byte array
25	bytes2	variable	seg2 raw byte array
26	bytes4	variable	seg4 raw byte array
27	xstring1	variable	seg1 xstring
28	xstring2	variable	seg2 xstring
29	xstring4	variable	seg4 xstring
30	xjsonarray1	variable	seg1 xjson array
31	xjsonarray2	variable	seg2 xjson array
32	xjsonarray4	variable	seg4 xjson array
33	xjsonobject1	variable	seg1 xjson object

Code	Value	Length (bytes)	Description
`34`	xjsonobject2	variable	seg2 xjson object
`35`	xjsonobject4	variable	seg4 xjson object
`36` - `255`	unusued, reserved		

XString Format

The **xstring** value type allows chaining mutliple encoded values to be interpretted as a string. The xstring segment length must be the total number of bytes of all encoded values in the string.

Note that although any data type may be included in an xstring, the exact string representation of certain values may vary depending on the decoding environment (specifically, the formatting of floating point values) and thus it is not recommended to include them in xstring values. JSON values will be converted to their minimal string representation. Byte arrays will be converted to a hex string. Null values will be treated as an empty string.

XJSON Array Format

The **xjsonarray** value type allows chaining mutliple encoded values to be interpretted as a JSON array. The xjsonarray segment length must be the total number of bytes of all encoded values in the array.

XJSON Object Format

The **xjsonobject** value type allows chaining mutliple encoded values to be interpretted as a JSON object. Each pair of values in the list is interpretted as a key-value pair. The xjsonobject segment length must be the total number of bytes of all encoded key-value pairs in the object. Note that key values must resolve to a string, xstring, number, boolean, or null (which will be interpretted as an empty string key).

Examples

Null Value:

Code	Content (0 bytes)
`0x00`	

300 (as 2 byte integer):

Code	Content (2 bytes)
`0x07`	`0x01` `0x2c`

0.24 (as 8 byte float):

Code	Content (8 bytes)
`0x0b`	`0x3f` `0xce` `0xb8` `0x51` `0xEB` `0x85` `0x1E` `0xb8`

"foo" (as string1):

Code	Content (4 bytes)
`0x0c`	`0x03` `0x66` `0x6f` `0x6f`

{"foo":"bar"} (as json1):

Code	Content (14 bytes)
`0x0f`	`0x0d` `0x7b` `0x22` `0x66` `0x6f` `0x6f` `0x22` `0x3a` `0x22` `0x62` `0x61` `0x72` `0x22` `0x7d`

"foo123" (as xstring1, split as string1 "foo" and int1 123):

Code	Content (7 bytes)
`0x1b`	[`0x06`](total length) [`0x03` `0x66` `0x6f` `0x6f`]("foo") [`0x04` `0x7b`](123)

Reference Dictionary

The xbin format provides user-managed compression through the reference dictionary. It can contain up to the 4 byte signed integer index space (2,147,483,647). The order of values affects the compression ratio; index 0-255 can be represented with a single byte, 256-65,535 with 2 bytes, and above requires 4 bytes.

Binary File Format

UUID

The file starts with a 16 byte binary encoded UUID. This is intended to uniquely identify the file, but the exact implementation and usage beyond this is not explicitly defined as part of the format definition. For XINA purposes two xbin files with the same UUID would be expected to be identical.

Header

A value which must either be `null` or a `jsonobject1`, `jsonobject2`, or `jsonobject4`. This is currently a placeholder with no defined parameters.

Reference Dict

A seg4 containing 0 to 2,147,483,647 encoded values, which may be referenced by zero based index with the reference dict index value types.

Rows

Each row contains:

- 8 byte signed integer containing Unix time with microsecond precision
- seg4 of row data, containing
  - header, single value which must either be `null` or a `jsonobject1`, `jsonobject2`, or `jsonobject4`
  - one or more key,value pairs

The row header is currently a placeholder with no defined parameters.

Example File

Given a data set with UUID 9462ef87-f232-4694-922c-12b93c95e27c:

t	voltage	current	label
0	5	10	"foo"
1			"bar"
2	5	null	

A corresponding xbin file containing the same data would be:

UUID (16 bytes)

`0x94` `0x62` `0xef` `0x87` `0xf2` `0x32` `0x46` `0x94` `0x92` `0x2c` `0x12` `0xb9` `0x3c` `0x95` `0xe2` `0x7c`

Header (1 byte)

`0x00` (null, 1 byte)

Reference Dict, three values, "voltage", "current", "label" (29 bytes)

`0x00` `0x00` `0x00` `0x19` (seg4 length, 25)

`0x0a` `0x07` `0x76` `0x6f` `0x6c` `0x74` `0x61` `0x67` `0x65` ("voltage", 9 bytes)

`0x0a` `0x07` `0x63` `0x75` `0x72` `0x72` `0x65` `0x6e` `0x74` ("current", 9 bytes)

`0x0a` `0x05` `0x6c` `0x61` `0x62` `0x65` `0x6c` ("label", 7 bytes)

Row t0 (22 bytes)

`0x00` `0x00` `0x00` `0x00` `0x00` `0x00` `0x00` `0x00` (time, 0, 8 bytes)

`0x00` `0x00` `0x00` `0x0e` (row length, 15, 4 bytes)

`0x00` (header, null, 1 byte)

`0x01` `0x00` (reference to index 0, "voltage", 2 bytes)

`0xff` (type code reference to index 0, 5, 1 byte)

`0x01` `0x01` (reference to index 1, "current", 2 bytes)

`0x04` `0x0a` (integer value 10, 2 bytes)

`0x01` `0x02` (reference to index 2, "label", 2 bytes)

`0x0a` `0x03` `0x66` `0x6f` `0x6f` (string "foo", 5 bytes)

Row t1 (20 bytes)

`0x00` `0x00` `0x00` `0x00` `0x00` `0x00` `0x00` `0x01` (time, 1, 8 bytes)

`0x00` `0x00` `0x00` `0x08` (row length, 8, 4 bytes)

`0x00` (header, null, 1 byte)

`0x01` `0x02` (reference to index 2, "label", 2 bytes)

`0x0a` `0x03` `0x62` `0x61` `0x72` (string "bar", 5 bytes)

Row t2 (19 bytes)

`0x00` `0x00` `0x00` `0x00` `0x00` `0x00` `0x00` `0x02` (time, 2, 8 bytes)

`0x00` `0x00` `0x00` `0x0e` (row length, 15, 4 bytes)

`0x00` (header, null, 1 byte)

`0x01` `0x00` (reference to index 0, "voltage", 2 bytes)

`0x00` (type code reference to index 0, 5, 1 byte)

`0x01` `0x01` (reference to index 1, "current", 2 bytes)

`0x00` (null, 1 byte)

Revision #34
Created 2 August 2022 17:27:24 by Nick Dobson
Updated 5 February 2025 15:28:14 by Nick Dobson