

Encodings for OBIX: Common Encodings Version 1.0

Committee Specification Draft 02 / Public Review Draft 02

19 December 2013

Specification URIs

This version:

<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd02/obix-encodings-v1.0-csprd02.pdf>
(Authoritative)
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd02/obix-encodings-v1.0-csprd02.html>
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd02/obix-encodings-v1.0-csprd02.doc>

Previous version:

<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd01/obix-encodings-v1.0-csprd01.pdf>
(Authoritative)
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd01/obix-encodings-v1.0-csprd01.html>
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd01/obix-encodings-v1.0-csprd01.doc>

Latest version:

<http://docs.oasis-open.org/obix/obix-encodings/v1.0/obix-encodings-v1.0.pdf> (Authoritative)
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/obix-encodings-v1.0.html>
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/obix-encodings-v1.0.doc>

Technical Committee:

OASIS Open Building Information Exchange (oBIX) TC

Chair:

Toby Considine (toby.considine@unc.edu), University of North Carolina at Chapel Hill

Editor:

Markus Jung (mjung@auto.tuwien.ac.at), Institute of Computer Aided Automation, Vienna University of Technology

Related work:

This specification is related to:

- *OBIX Version 1.1*. Edited by Craig Gemmill. Latest version. <http://docs.oasis-open.org/obix/obix/v1.1/obix-v1.1.html>.
- *Bindings for OBIX: REST Bindings Version 1.0*. Edited by Craig Gemmill and Markus Jung. Latest version. <http://docs.oasis-open.org/obix/obix-rest/v1.0/obix-rest-v1.0.html>.
- *Bindings for OBIX: SOAP Bindings Version 1.0*. Edited by Markus Jung. Latest version. <http://docs.oasis-open.org/obix/obix-soap/v1.0/obix-soap-v1.0.html>.
- *Bindings for OBIX: Web Socket Bindings Version 1.0*. Edited by Matthias Hub. Latest version. <http://docs.oasis-open.org/obix/obix-websocket/v1.0/obix-websocket-v1.0.html>.

Abstract:

This document specifies different encodings for OBIX objects adhering to the OBIX object model.

Status:

This document was last revised or approved by the OASIS Open Building Information Exchange (oBIX) TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at <http://www.oasis-open.org/committees/obix/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/obix/ipr.php>).

Citation format:

When referencing this specification the following citation format should be used:

[OBIX-Encodings]

Encodings for OBIX: Common Encodings Version 1.0. Edited by Markus Jung. 19 December 2013. OASIS Committee Specification Draft 02 / Public Review Draft 02. <http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd02/obix-encodings-v1.0-csprd02.html>. Latest version: <http://docs.oasis-open.org/obix/obix-encodings/v1.0/obix-encodings-v1.0.html>.

Notices

Copyright © OASIS Open 2013. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/policies-guidelines/trademark> for above guidance.

Table of Contents

1	Introduction.....	7
1.1	Terminology.....	7
1.2	Normative References.....	7
1.3	Non-Normative References.....	7
2	XML Encoding.....	8
2.1	Design Philosophy.....	8
2.2	XML Syntax.....	8
2.3	XML Encoding.....	8
2.4	XML Decoding.....	8
2.5	XML Namespace.....	9
2.6	Namespace Prefixes in Contract Lists.....	9
3	OBIX Binary.....	10
3.1	Binary Overview.....	10
3.2	Binary Constants.....	10
3.3	Value Encodings.....	11
3.3.1	Bool Encodings.....	11
3.3.2	Int Encodings.....	12
3.3.3	Real Encodings.....	12
3.3.4	Str Encodings.....	12
3.3.5	Abstime Encodings.....	13
3.3.6	Reltime Encodings.....	13
3.3.7	Time Encodings.....	13
3.3.8	Date Encodings.....	14
3.3.9	Status Encodings.....	14
3.4	Facets.....	15
3.4.1	Custom Facets.....	15
3.5	Children.....	16
4	JSON encoding.....	17
4.1	Object and value encoding rules.....	18
4.1.1	Bool encoding.....	18
4.1.2	Int encoding.....	18
4.1.3	Real encoding.....	18
4.1.4	Other types and facets.....	18
4.2	XML Namespace.....	19
4.3	Examples.....	19
4.4	MIME Type.....	19
5	EXI encoding.....	20
5.1	EXI options.....	20
5.1.1	Alignment options.....	20
5.1.2	Preservation options.....	20
5.2	Non-schema-informed EXI.....	20
5.3	Schema-informed EXI.....	20
5.4	MIME types.....	21

6	Conformance	22
Appendix A.	Acknowledgments	23
Appendix B.	Revision History	24

Table of Tables

Table 3-1 Binary Constants.....	11
Table 3-2 Bool Encodings.....	11
Table 3-3 Int Encodings.....	12
Table 3-4 Real Encodings.....	12
Table 3-5 Str Encodings.....	12
Table 3-6 Abstime Encodings.....	13
Table 3-7 Reltime Encodings.....	13
Table 3-8 Time Encodings.....	14
Table 3-9 Date Encodings.....	14
Table 3-10 Status Encodings.....	14
Table 3-11 Custom Facets.....	15

1 Introduction

This document specifies the encodings for OBIX.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

1.2 Normative References

- 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>.
- OBIX** *OBIX Version 1.1*. Edited by Craig Gemmill. Latest version. <http://docs.oasis-open.org/obix/obix/v1.1/obix-v1.1.html>.
- EXI** J. Schneider, T. Kamiya, Efficient XML Interchange (EXI) Format 1.0, W3C Recommendation, 10 March 2011
- RFC4627** Crockford, D., "The application/json Media type for JavaScript Object Notation (JSON)", RFC 4627, July 2007
- RFC768** Postel, J., "User Datagram Protocol", RFC 768, August 1980

1.3 Non-Normative References

- REST** Fielding R.T., *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation, University of California at Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- EXI MR** Y. Doi, EXI Messaging Requirements, IETF Internet-Draft, 25 February 2013
- EXI BP** M. Cokus, D. Vogelheim, Efficient XML Interchange (EXI) Best Practices, W3C Working Draft, 19 December 2007

26 2 XML Encoding

27 This chapter specifies how the OBIX object model is encoded in XML.

28 2.1 Design Philosophy

29 Since there are many different approaches to developing an XML syntax, it is worthwhile to provide a bit
30 of background to how the OBIX XML syntax was designed. Historically in M2M systems, non-standard
31 extensions have been second class citizens at best, but usually opaque. One of the design principles of
32 OBIX is to embrace vertical domain and vendor specific extensions, so that all data and services have a
33 level playing field.

34

35 In order to achieve this goal, the XML syntax is designed to support a small, fixed schema for all OBIX
36 documents. If a client agent understands this very simple syntax, then the client is guaranteed access to
37 the server's object tree regardless of whether those objects implement standard or non-standard
38 contracts.

39

40 Higher level semantics are captured via contracts. Contracts "tag" an object with a type and can be
41 applied dynamically. This is very useful for modeling systems which are dynamically configured in the
42 field. What is important is that contracts are optionally understood by clients. Contracts do not affect the
43 XML syntax nor are clients required to use them for basic access to the object tree. Contracts are merely
44 an abstraction which is layered cleanly above the object tree and its fixed XML syntax.

45 2.2 XML Syntax

46 The OBIX XML syntax maps very closely to the abstract object model. The syntax is summarized:

- 47 • Every OBIX object maps to exactly one XML element;
- 48 • An object's children are mapped as children XML elements;
- 49 • The XML element name maps to the built-in object type;
- 50 • Everything else about an object is represented as XML attributes;

51 The object model figure in Chapter 4 of the OBIX core specification [OBIX] illustrates the valid XML
52 elements and their respective attributes. Note the `val` object is simply an abstract base type for the
53 objects which support the `val` attribute - there is no `val` element.

54 2.3 XML Encoding

55 The following rules apply to encoding OBIX documents:

- 56 • OBIX documents MUST be well formed XML;
- 57 • OBIX documents SHOULD begin with XML Declaration specifying their encoding ;
- 58 • It is RECOMMENDED to use UTF-8 encoding without a byte order mark;
- 59 • OBIX documents MUST NOT include a Document Type Declaration – OBIX documents cannot
60 contain an internal or external subset;
- 61 • OBIX documents SHOULD include an XML Namespace definition. Convention is to declare the
62 default namespace of the document to "http://docs.oasis-open.org/obix/ns/201312/schema".

63 2.4 XML Decoding

64 The following rules apply to decoding of OBIX documents:

- 65 • MUST conform to XML processing rules as defined by XML 1.1;
- 66 • Documents which are not well formed XML MUST be rejected;
- 67 • Parsers are not required to understand a Document Type Declaration;
- 68 • Any unknown element MUST be ignored regardless of its XML namespace
- 69 • Any unknown attribute MUST be ignored regardless of its XML namespace

70

71 The basic rule of thumb is: strict in what you generate, and liberal in what you accept. OBIX parsers are
72 required to ignore elements and attributes which they do not understand. However an OBIX parser MUST
73 never accept an XML document which isn't well formed (such as mismatched tags).

74 2.5 XML Namespace

75 XML namespaces for standards within the OBIX umbrella should conform to the following pattern:

```
76 http://docs.oasis-open.org/obix/ns/{short-identifier and version}
```

77

78 The XML namespace for OBIX version 1.1 is:

```
79 http://docs.oasis-open.org/obix/ns/201312/schema
```

80 All XML in this document is assumed to have this namespace unless otherwise explicitly stated.

81 2.6 Namespace Prefixes in Contract Lists

82 XML namespace prefixes defined within an OBIX document may be used to prefix the URIs of a contract
83 list. If a URI within a contract list starts with string matching a defined XML prefix followed by the ":" colon
84 character, then the URI is normalized by replacing the prefix with its namespace value. This rule also
85 applies to the `href` attribute as a convenience for defining contract themselves.

86

87 The XML namespace prefix of "obix" is predefined. This prefix is used for all the OBIX defined contracts.
88 The "obix" prefix is literally translated into "http://docs.oasis-open.org/obix/ns/201312/def". For example
89 the URI "obix:bool" is translated to " http://docs.oasis-open.org/obix/ns/201312/def/bool". Documents
90 SHOULD NOT define an XML namespace using the prefix "obix" which collides with the predefined "obix"
91 prefix – if it is defined, then it is superseded by the predefined value of " http://docs.oasis-
92 open.org/obix/ns/201312/def". All OBIX defined contracts are accessible via their HTTP URI using the
93 HTTP binding (at least they should be one day).

94

95 An example OBIX document with XML namespace prefixes normalized:

```
96 <obj xmlns:acme="http://acme.com/def/" href="acme:CustomPoint"  
97 is="acme:Point obix:Point"/>  
98  
99 <obj href="http://acme.com/def/CustomPoint"  
100 is="http://acme.com/def/Point http://docs.oasis-open.org/obix/ns/201312/def/Point"/>
```

101

3 OBIX Binary

102
103
104
105
106

In addition to the XML encoding, a binary encoding is defined for the OBIX data model. The binary encoding allows OBIX objects to be serialized with higher compression using less computing resources. The use case for binary encoding is targeted for severely constrained edge devices and sensor networks such as 6LoWPANs. When possible, an XML encoding SHOULD always be preferred over a binary encoding.

107
108
109
110

Full fidelity with OBIX object model is maintained with the binary encoding. All object types and facets are preserved. However XML extensions such as custom namespaces, elements, and attributes are not address by the binary encoding. The OBIX binary encoding is based strictly on the obix data model itself, not its XML InfoSet.

111

3.1 Binary Overview

112
113
114

The OBIX data model is comprised of 16 object types (elements in XML) and 19 facets (attributes in XML). The OBIX binary encoding is based on assigning a numeric code to each object type and to each facet type. We format these codes using a byte header with the bits structured as:

115
116

```
7654 3210
MCCC CCVV
```

117
118
119

The top most bit M is the more flag, it is used to indicate more facets follow. Bits 6 through 2 are used to store a 5-bit numeric code for object types and facet types. The bottom 2 bits are used to indicate a 2-bit numeric code for how the value of the object or facet is encoded.

120

The binary grammar is defined according to the following BNF productions:

121
122
123
124

```
<obj>      := <objHeader> [objVal] (facet)* [children]
<facet>    := <facetHeader> [facetVal] |
             <facetHeader> <string> <value>
<children> := (<obj>)*
```

125
126
127
128
129
130
131
132

All documents start with a one byte objHeader structured as a MCCCCCVV bitmask. The 5-bit C mask indicates an Obj Code specified in Binary Constants table. If the object type contains a value encoding (specified in the Obj Value column), then the 2-bit V mask indicates how the following bytes are used to encode the "val" attribute. If the objHeader has the more bit set, then one or more facet productions follow. Facets are encoded with a one byte header using the same MCCCCCVV bitmask, except the 5-bit C mask indicates a Facet Code (not an Obj Code). The facet value is encoded using the 2-bit V mask. If one of the facets includes the hasChildren code, then one or more child objects follow terminated by the endChildren object code.

133

3.2 Binary Constants

134
135
136

The following table enumerates the Obj Codes and Facet Codes which are encoded into 5-bits in the MCCCCCVV bitmask. The Obj Value and Facet Value columns specifies how to interpret the 2-bit V code for the value encoding.

Numeric Code	Constant	Obj Code	Obj Value	Facet Code	Facet Value
1 << 2	0x04	obj	none	hasChildren	none
2 << 2	0x08	bool	bool	name	str
3 << 2	0x0C	int	int	href	str
4 << 2	0x10	real	real	is	str
5 << 2	0x14	str	str	of	str

6 << 2	0x18	enum	str	in	str
7 << 2	0x1C	uri	str	out	str
8 << 2	0x20	abstime	abstime	null	bool
9 << 2	0x24	reltime	reltime	icon	str
10 << 2	0x28	date	date	displayName	str
11 << 2	0x2C	time	time	display	str
12 << 2	0x30	list	none	writable	bool
13 << 2	0x34	op	none	min	obj specific
14 << 2	0x38	feed	none	max	obj specific
15 << 2	0x3C	ref	none	unit	str
16 << 2	0x40	err	none	precision	int
17 << 2	0x44	childrenEnd	none	range	str
18 << 2	0x48			tz	str
19 << 2	0x4C			status-0	status-0
20 << 2	0x50			status-1	status-1
21 << 2	0x54			customFacet	facet specific

137 *Table 3-1 Binary Constants*

138 3.3 Value Encodings

139 Each obj type and facet type may have an associated value encoding. For example, to encode the
 140 precision facet we must specify the facet code 0x40 plus the value of that facet which happens to be an
 141 integer. The object types bool, int, enum, real, str, uri, abstime, reltime, date, and time are always implied
 142 to have their value encoded (equivalent to the val attribute in XML).

143 3.3.1 Bool Encodings

144 The following boolean encodings are supported:

Constant	Encoding	Description
0	false	Indicates false value
1	true	Indicates true value

145 *Table 3-2 Bool Encodings*

146 The boolean encodings are fully specified in the 2-bit V mask. No extra bytes are required. Examples:

147 `<bool val="false"/> => 08`
 148 `<bool val="true"/> => 09`

149 The obj code for bool is 0x08. In the case of false we bit-wise OR this with a value code of 0, so the
 150 complete encoding is the single byte 0x08. When val is true, we bitwise OR 0x08 with 0x01 with a result
 151 of 0x09.

152 3.3.2 Int Encodings

153 The following integer encodings are supported:

Constant	Encoding	Description
0	u1	Unsigned 8-bit integer value
1	u2	Unsigned 16-bit integer value
2	s4	Signed 32-bit integer value
3	s8	Signed 64-bit integer value

154 *Table 3-3 Int Encodings*

155 Integers between 0 and 255 can be encoded in one byte. Larger numbers require 2, 4, or 8 bytes.
156 Numbers outside of the 64-bit range are not supported. Examples:

```
157 <int val="34"/> => 0C 22  
158 <int val="2093 "/> => 0D 08 2D  
159 <int val="76000"/> => 0E 00 01 28 E0  
160 <int val="-300"/> => 0E FF FF FE D4  
161 <int val="12345678901"/> => 0F 00 00 00 02 DF DC 1C 35
```

162 The obj code for int is 0x0C. In first example, the value can be encoded as an unsigned 8-bit number, so
163 we mask 0x0C with the value code 0x00 and then encode 34 using one byte. The second example is a
164 u2 encoding, so we mask 0x0C with value code 0x01 to get 0x0D and then use two additional bytes to
165 encode 2093 as a 16-bit unsigned integer. The other examples illustrate how values would be encoded in
166 s4 and s8. Encoders SHOULD select the encoding type which results in the fewest number of bytes.

167 3.3.3 Real Encodings

168 The following real encodings are supported:

Constant	Encoding	Description
0	f4	32-bit IEEE floating point value
1	f8	64-bit IEEE floating point value

169 *Table 3-4 Real Encodings*

170 Examples:

```
171 <real val="75.3"/> => 10 42 96 99 9A  
172 <real val="15067.059"/> => 11 40 CD 6D 87 8D 4F DF 3B
```

173 3.3.4 Str Encodings

174 The following str encodings are supported:

Constant	Encoding	Description
0	utf8	null terminated UTF-8 string
1	prev	u2 index of previously encoded string

175 *Table 3-5 Str Encodings*

176 String encoding are used for many obj and facet values. Every time a string value is encoded within a
177 given document, it is assigned a zero based index number. The first string encoded as utf8 is assigned
178 zero, the second one, and so on. If subsequent string values have the exact same value, then the prev
179 value encoding is used to reference the previous string via its index number. This requires binary
180 decoders to keep track of all strings during decoding, since later occurrences in the document might
181 reference that string.

182 Simple example which illustrates a null terminated string:

```
183 <str val="obix"/> => 14 6F 62 69 78 00  
184
```

185 Complex example which illustrates two strings with the same value:

```
186 <obj>  
187 <str val="abc"/>  
188 <str val="abc"/>  
189 </obj> => 84 04 14 61 62 63 00 15 00 00 44
```

190 The first byte 0x84 is the obj code masked with the more bit. The next byte 0x04 is the hasChildren
191 marker which indicates that children objects follow (covered further in section 3.5). The next byte is the
192 0x14 str obj code masked with the 0x00 utf8 value code followed by the 61 62 63 00 encoding of "abc".
193 The next byte 0x15 is the str obj type 0x14 masked with the 0x01 prev value code, followed by the u2
194 encoding of index zero which references string value zero "abc". The last byte 0x44 is the end of children
195 marker.

196 3.3.5 Abstime Encodings

197 The following abstime encodings are supported:

Constant	Encoding	Description
0	sec	signed 32-bit number of seconds since epoch
1	ns	signed 64-bit number of nanoseconds since epoch

198 *Table 3-6 Abstime Encodings*

199 The epoch for OBIX timestamps is defined as midnight 1 January 2000 UTC. Times before the epoch are
200 represented as negative numbers. Encoding with seconds provides a range of +/-68 years. The
201 nanosecond encoding provides a range of +/-292 years. Timestamps outside of this range are not
202 supported. Examples:

```
203 <abstime val="2000-01-30T00:00:00Z"/> => 20 00 26 3B 80  
204 <abstime val="1999-12-01T00:00:00Z"/> => 20 FF D7 21 80  
205 <abstime val="2009-10-20T13:00:00-04:00"/> => 20 12 70 A9 10  
206 <abstime val="2009-10-20T13:00:00.123Z"/> => 21 04 4B 10 30 8D 78 F4 C0
```

207 The first example is encoded as 0x00263B80 which equates to 29x24x60x60 seconds since the OBIX
208 epoch. The second example illustrates a negative number seconds for a timestamp before the epoch.
209 The last example illustrates a 64-bit nanosecond encoding.

210 3.3.6 Reltime Encodings

211 The following reltime encodings are supported:

Constant	Encoding	Description
0	sec	signed 32-bit number of seconds
1	ns	signed 64-bit number of nanoseconds

212 *Table 3-7 Reltime Encodings*

213 Consistent with the abstime encoding, both a second and nanosecond encoding are provided. No support
214 is provided for ambiguous periods such as 1 month which don't map to a fixed number of seconds.
215 Examples:

```
216 <reltime val="PT5M"/> => 24 00 00 01 2C  
217 <reltime val="PT0.123S"/> => 25 00 00 00 00 07 54 D4 C0
```

218 3.3.7 Time Encodings

219 The following time encodings are supported:

Constant	Encoding	Description
0	sec	unsigned 32-bit number of seconds since midnight
1	ns	unsigned 64-bit number of nanoseconds since midnight

220 *Table 3-8 Time Encodings*

221 The time encoding works similar to reltime using a number of seconds or nanoseconds since midnight.
 222 Examples:

```
223 <time val="04:30:00"/> => 2C 00 00 3F 48
224 <time val="04:30:00.123"/> => 2D 00 00 0E BB E2 93 A4 C0
```

225 3.3.8 Date Encodings

226 The following date encodings are supported:

Constant	Encoding	Description
0	yynd	u2 year, u1 month 1-12, u1 day 1-31

227 *Table 3-9 Date Encodings*

228 Dates are encoded using four bytes. The year is encoded as a common era year via a 16-bit integer, the
 229 month as a 8-bit integer between 1 and 12, and the day as an 8-bit integer between 1 and 31. Examples:

```
230 <date val="2009-10-20"/> => 28 07 D9 0A 14
```

231 3.3.9 Status Encodings

232 The following status encodings are supported:

Constant	Encoding	Description
0	status-0-disabled	disabled status
1	status-0-fault	fault status
2	status-0-down	down status
3	status-0-unacked-alarm	unackedAlarm status
0	status-1-alarm	alarm status
1	status-1-unacked	unacked status
2	status-1-overridden	overridden status

233 *Table 3-10 Status Encodings*

234 The status facet is encoded inline to avoid consuming an extra byte. Since there are eight status values,
 235 but only 2-bits for the value encoding we use two different facet codes to give us the required range. The
 236 ok status is implied by omitting the status facet. Examples:

```
237 <obj status="ok"/> => 04
238 <obj status="disabled"/> => 84 4C // 0x4C | 0x00
239 <obj status="fault"/> => 84 4D // 0x4C | 0x01
240 <obj status="down"/> => 84 4E // 0x4C | 0x02
241 <obj status="unackedAlarm"/> => 84 4F // 0x4C | 0x03
242 <obj status="alarm"/> => 84 50 // 0x50 | 0x00
243 <obj status="unacked"/> => 84 51 // 0x50 | 0x01
244 <obj status="overridden"/> => 84 52 // 0x50 | 0x02
```

245 The first example illustrates the ok status, the entire document is encoded with the one byte obj type code
 246 of 0x40. The rest of the examples start with 0x84 which represents the obj type code masked with the
 247 more bit. Status values from disabled to unackedAlarm use facet code status-0 and from alarm to

248 overridden use facet code status-1. It is illegal for a single object to define both the status-0 and status-1
 249 facet codes.
 250

251 3.4 Facets

252 Facets are encoded according to the value type as specified in the Binary Constants section. The
 253 min/max facet value types are implied by their containing object which must match the object value with
 254 exception of str which uses integers for min/max. Some examples:

```
255 <list name="foo"/> => B0 08 66 6F 6F 00
256 <list name="foo" displayName="Foo"/> => B0 88 66 6F 6F 00 28 46 6F 6F 00
257 <int val="3" min="0" max="100"/> => 8C 03 B4 00 38 64
258 <obj href="p4.2"/> => 84 0C 70 34 2E 32 00
```

259 Note that a string of multiple facets is indicated by masking the 0x80 more bit into the object/facet
 260 headers.

261

262 3.4.1 Custom Facets

263 The following extension encodings are supported:

Constant	Encoding	Description
0	extension	Facet name encoded as string value object, followed by value object containing value associated with facet.

264 *Table 3-11 Custom Facets*

265 Custom facets are facets which are not specified by this standard but rather supplied by a particular
 266 implementation. Custom facets will include two objects immediately following the header byte: a string
 267 object, specifying the name of the facet, and a value object, specifying the value associated with the
 268 facet.

269 Both the string and value objects associated with the facet must provide a value, and neither object may
 270 supply additional facets or contain any child objects. Additionally, the value object associated with the
 271 facet must be one of the following object types:

- 272 • bool
- 273 • int
- 274 • real
- 275 • str
- 276 • enum
- 277 • uri
- 278 • abstime
- 279 • reltime
- 280 • date
- 281 • time

282 Other types for the value object are not supported.

283

284 Examples:

```
285 <int val="34" my:int="50"/> => 8C 22 54 14 6D 79 3A 69 6E 6F 00 0C 32
286 <bool val="false" my:bool="true"/> => 88 54 14 6D 79 3A 69 6E 74 00 09
287 <bool val="true" my:str="hi!"/> => 89 54 14 6D 79 3A 73 74 72 00 14 68 69 21 00
```

288

289 3.5 Children

290 The special facet code hasChildren and the special object code endChildren are used to encode nested
291 children objects. Let's look at a simple example:

```
292 <obj> <bool val="false"/> </obj> => 84 04 08 44
```

293 Let's examine each byte: the first byte 0x84 is the mask of obj type code 0x04 with the 0x80 more bit
294 indicating a facet follows. The 0x04 facet code indicates the obj has children. The next byte is interpreted
295 as the beginning of a new object, which is the bool object code 0x08. Since the more bit is not set on the
296 bool object, there are no more facets. The next byte is the endChildren object code indicating we've
297 reached the end of the children objects for obj. It serves a similar purpose as the end tag in XML.

298 Technically the hasChildren facet could have additional facets following it by setting the more bit.
299 However, this specification requires that the hasChildren facet is always declared last within a given
300 object's facet list. This makes it an encoding error to have the more bit set on the hasChildren facet code.

301 Let's look a more complicated example with multiple nested children:

```
302 <list href="xyz">  
303   <bool val="false"/>  
304   <obj><int val="255"/></obj>  
305 </list>                                => B0 8C 78 79 7A 00 04 08 84 04 0C FF 44 44  
306  
307 <list>                                => B0                                // 0x80 | 0x30  
308 href="xyz"                            => 8C 78 79 7A 00          // 0x80 | 0x0C | 0x00 + x + y + z  
309 hasChildren                            => 04  
310 <bool val="false"/>                    => 08  
311 <obj>                                  => 84                                // 0x80 | 0x04  
312 hasChildren                            => 04  
313 <int val="255">                          => 0C FF              // 0x0C | 0x00 + u1 of 255  
314 endChildren </obj>                      => 44  
315 endChildren </list>                     => 44
```


316

4 JSON encoding

317 The Java script object notation is a lightweight, text-based, language-independent data interchange
318 format. It is derived from the object literals of JavaScript, as defined in the ECMAScript Programming
319 Language Standard (ECMA) [RFC4627].

320 JSON uses two structures for representing information:

- 321 • A collection of name/value pairs
- 322 • An ordered list of values

323 In JSON an object is an unordered set of name/value pairs and the encoding of an object starts with a left
324 brace and ends with a right brace. A colon is used to separate the name and the value and a comma
325 separates multiple name/value pairs. The JSON encoding of OBIX is inspired by JSONML, which
326 provides a lossless two-way conversation between JSON and XML. A Java reference implementation can
327 be found here¹.

328 The following grammar is used to represent OBIX objects:

329

```
330 element
331 = '{' obix-identifier ',' attribute-list ', "children":[' element-list ']'
332 | '{' obix-identifier ',' attribute-list '}'
333 | '{' obix-identifier ', "children":[' element-list ']'
334 | '{' obix-identifier '}'
335 | string
336 ;
337
338 obix-identifier
339 = "obix":type-name
340 ;
341
342 type-name
343 = string
344 ;
345
346 attribute-list
347 = attribute ',' attribute-list
348 | attribute
349 ;
350
351 attribute
352 = attribute-name ':' attribute-value
353 ;
354
355 attribute-name
356 = string
357 ;
358
359 attribute-value
360 = string
361 | number
362 | 'true'
363 | 'false'
364 ;
365
366 element-list
367 = element ',' element-list
368 | element
369 ;
```

¹ <http://json.org/java/>

370 4.1 Object and value encoding rules

371 Objects MUST be encoded according to the grammar given above. The OBIX object is encoded as JSON
372 object which an unordered list of name/value pairs. The object type which is used as element name in
373 XML is encoded as a name/value pair using "tag" as name and the object type as string value.

374 The XML and JSON representation of a simple `obj`:

```
375 <obj/> → {"obix":"obj"}
```

376 The attributes of an object are mapped to name value/pairs:

```
377 <obj name="myName" href="/myHref"> → {"obix":"obj", "name":"myName", "href":"/myHref"/>
```

378 If objects have an extent, the children objects contained in this extend are mapped to a name/value pair
379 using "children" as name and an ordered array of objects as value.

380 The XML representation of an object with extend is mapped to the JSON representation as shown in the
381 examples below.

382 XML:

```
383 <obj href="/a/">  
384   <obj name="b" href="b">  
385     <obj name="c"/>  
386     <ref name="d" href="d"/>  
387   </obj>  
388 </obj>
```

389 JSON:

```
390 {  
391   "obix": "obj",  
392   "href": "/a",  
393   "children": [{  
394     "obix": "obj",  
395     "name": "b",  
396     "href": "b",  
397     "children": [{  
398       "obix": "obj",  
399       "name": "c",  
400     }, {  
401       "obix": "ref",  
402       "name": "d",  
403       "href": "d",  
404     }  
405   ]  
406 }
```

407 4.1.1 Bool encoding

408 The `xs:boolean` val attribute of the `bool` object is mapped to the `true` or `false` literals of JSON.

```
409 <bool val="true"/> → {"obix":"bool", "val":true}
```

410 4.1.2 Int encoding

411 The `xs:long` val attribute of the `int` object is mapped to the number representation of JSON.

```
412 <int val="5"/> → {"obix":"int", "val":5}
```

413 4.1.3 Real encoding

414 The `xs:double` val attribute of the `real` object is mapped to the number representation of JSON.

```
415 <real val="5.5"/> → {"obix":"real", "val":5.5}
```

416 4.1.4 Other types and facets

417 All other types and facets are mapped to name/value pairs using JSON string representation. Facets are
418 mapped to name/value pairs as described by the rules above.

419 4.2 XML Namespace

420 If namespace information should be preserved in the JSON encoding, namespace prefixes SHOULD be
421 normalized before the object is encoded to JSON as shown in the examples below:

422 Object with namespace prefixes in use:

```
423 <obj xmlns:acme="http://acme.com/def/" href="acme:CustomPoint"  
424 is="acme:Point obix:Point"/>
```

425 Object with normalized namespace information:

```
426 <obj href="http://acme.com/def/CustomPoint"  
427 is="http://acme.com/def/Point http://docs.oasis-open.org/obix/ns/201312/def/Point"/>
```

428 JSON encoded object with normalized namespace information:

```
429 {obix:"obj", href:"http://acme.com/def/CustomPoint", is:"http://acme.com/def/Point  
430 http://docs.oasis-open.org/obix/ns/201312/def/Point"}
```

431

432 4.3 Examples

433 The following examples illustrate the JSON encoding:

434

435 **Example – OBIX About:**

436 XML:

```
437 <obj name="about">  
438 <str name="obixVersion" val="1.1"/>  
439 <str name="serverName" val="obix"/>  
440 <abstime name="serverTime" val="2006-02-08T09:40:55.000+05:00:00Z"/>  
441 <abstime name="serverBootTime" val="2006-02-08T09:33:31.980+05:00:00Z"/>  
442 <str name="vendorName" val="Acme, Inc."/>  
443 <uri name="vendorUrl" val="http://www.acme.com"/>  
444 <str name="productName" val="Acme OBIX Server"/>  
445 <str name="productVersion" val="1.0.3"/>  
446 <uri name="productUrl" val="http://www.acme.com/obix"/>  
447 </obj>
```

448

449 JSON:

```
450 {"obix":"obj", "name":"about", "children": [  
451 {"obix":"str", "name":"obixVersion", "val":"1.1"},  
452 {"obix":"str", "name":"serverName", "val":"obix"},  
453 {"obix":"abstime", "name":"serverTime", "val":"2006-02-08T09:40:55.000+05:00:00Z"},  
454 {"obix":"abstime", "name":"serverBootTime", "val":"2006-02-  
455 08T09:33:31.980+05:00:00Z"},  
456 {"obix":"str", "name":"vendorName", "val":"Acme, Inc."},  
457 {"obix":"uri", "name":"vendorURL", "val":"http://www.acme.com"},  
458 {"obix":"str", "name":"productName", "val":"Acme OBIX Server"},  
459 {"obix":"str", "name":"productVersion", "val":"1.0.3"},  
460 {"obix":"uri", "name":"prodctUrl", "val":"http://www.acme.com/obix"}  
461 ]}]
```

462 4.4 MIME Type

463 If a client wants to use JSON encoding it MUST use the JSON MIME type `application/json`
464 according to [RFC4627].

465 5 EXI encoding

466 The Efficient XML Interchange [EXI] format is a very compact representation for XML which aims at
467 providing high performance and significantly reduced bandwidth requirements for XML based protocols. It
468 uses a grammar driven approach based on entropy encoding which can be used with schema information
469 but also without any schema information.

470 5.1 EXI options

471 EXI provides several encoding options that communicating parties need to agree upon in order to ensure
472 interoperability.

473 If EXI encoding is used for OBIX the following options MUST be used by a client and server
474 implementation.

475 5.1.1 Alignment options

476 In contrast to XML EXI is by default bit-packed, which means the information is stored in the most
477 compact representation as possible, regardless of possible byte boundaries. This allows for example to
478 store 8 Boolean values into one single Byte, versus 8 Bytes with a single character representing the
479 value, e.g. 'T' or 'F'. If a textual representation like 'true' or 'false' is used, 4 to 5 Bytes are used for
480 representing the Boolean value.

481 EXI defines 4 options for alignment: `compress`, `preCompress`, `byteAligned` and `bitPacked`.

482 In order to have the best possible compression for OBIX `bitPacked` alignment MUST be used.

483 5.1.2 Preservation options

484 EXI implementation may provide preservation options specifying which type of XML information should be
485 remained in the EXI representation, like comments, programming instructions, document type
486 declarations and namespace.

487 For OBIX only name space declarations MUST be preserved. Every other non-relevant information MAY
488 be omitted.

489 5.2 Non-schema-informed EXI

490 EXI can be used without any schema information about the XML infoset that shall be encoded. This has
491 the advantage that no schema information is required at the decoders' site, but comes with the
492 disadvantage of being less efficient and providing only a limited compression for small payloads.

493 5.3 Schema-informed EXI

494 Schema-informed EXI allows making the encoding most efficient even for small payload sizes. Within
495 constrained environments schema-informed EXI SHALL be used to in order to have the best compression
496 effect. With object encoders and decoders even the performance penalty of processing XML structures in
497 memory can be avoided.

498 For schema-informed the normative `obix.xsd` schema file representing the OBIX 1.1 object model MUST
499 be used in order to provide interoperability among different vendor implementations.

500 For content negotiation and to determine if schema-informed or non-schema-informed EXI encoding
501 should be used either an out-of-band agreement between a client and server need to be done or the EXI
502 best practices [EXI BP] or the guidelines in [EXI MR] need to be followed.

503 **5.4 MIME types**

504 If a client wants to use EXI encoding it **MUST** use the MIME type `application/exi` for EXI without
505 schema information and the MIME type `application/x-obix-exi` for schema-informed
506 representation.

507 **6 Conformance**

508 An implementation is compliant with this specification if it implements all MUST or REQUIRED level
509 requirements. An implementation MUST specify its supported encodings.

510 Appendix A. Acknowledgments

511 The following individuals have participated in the creation of this specification and are gratefully
512 acknowledged:

513 Participants:

514 Ron Ambrosio, IBM
515 Brad Benson, Trane
516 Ron Bernstein, LonMark International*
517 Ludo Bertsch, Continental Automated Buildings Association (CABA)
518 Chris Bogen, US Department of Defense
519 Rich Blomseth, Echelon Corporation
520 Anto Budiardjo, Clasma Events, Inc.
521 Jochen Burkhardt, IBM
522 JungIn Choi, Kyungwon University
523 David Clute, Cisco Systems, Inc.*
524 Toby Considine, University of North Carolina at Chapel Hill
525 William Cox, Individual
526 Robert Dolin, Echelon Corporation
527 Marek Dziedzic, Treasury Board of Canada, Secretariat
528 Brian Frank, SkyFoundry
529 Craig Gemmill, Tridium, Inc.
530 Matthew Giannini, Tridium, Inc.
531 Christopher Kelly, Cisco Systems
532 Wonsuk Ko, Kyungwon University
533 Perry Krol, TIBCO Software Inc.
534 Corey Leong, Individual
535 Ulf Magnusson, Schneider Electric
536 Brian Meyers, Trane
537 Jeremy Roberts, LonMark International
538 Thorsten Roggendorf, Echelon Corporation
539 Anno Scholten, Individual
540 John Sublett, Tridium, Inc.
541 Dave Uden, Trane
542 Ron Zimmer, Continental Automated Buildings Association (CABA)*
543 Rob Zivney, Hirsch Electronics Corporation
544 Markus Jung, Institute of Computer Aided Automation, Vienna University of Technology
545

546

Appendix B. Revision History

547

Revision	Date	Editor	Changes Made
wd01	26 Mar 13	Markus Jung	Initial creation with XML and Binary encoding taken from the OBIX 1.1 WD07 working draft.
wd02	24 Apr 13	Markus Jung	First draft JSON and EXI encoding.
wd03	22 May 13	Markus Jung	Added JSON section on handling XML namespaces, shorter JSON names.
wd04	13 Jun 13	Markus Jung	Refined the use of examples (normative/non normative), EXI content negotiation.
wd05	28 Jun 13	Markus Jung	Updated reference section
wd06	8 Jul 13	Toby Considine	Updated acknowledgements
wd07	2 Oct 13	Markus Jung	Jira: OBIX-7, OBIX-56, OBIX-5, OBIX-6, OBIX-48
wd08	7 Nov 13	Markus Jung	Namespace rules, using straight quotes within the document.
wd09	12 Dec 13	Markus Jung	Fixed minor error (JSON encoding for real). Update OBIX namespace to current policy.
wd10	16 Dec 13	Markus Jung	Updated namespace (including date), using uppercase for OBIX
wd11	16 Dec 13	Markus Jung	Minor fixes: OBIX-79

548

549