

Encodings for OBIX: Common Encodings Version 1.0

Committee Specification Draft 03 / Public Review Draft 03

06 November 2014

Specification URIs

This version:

<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd03/obix-encodings-v1.0-csprd03.pdf>
(Authoritative)
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd03/obix-encodings-v1.0-csprd03.html>
<http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd03/obix-encodings-v1.0-csprd03.doc>

Previous 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>

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:

Encodings for OBIX: Common Encodings v1.0 specifies different encodings for OBIX objects adhering to the OBIX object model. OBIX provides the core information model and interaction pattern for communication with building control systems.

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. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix#technical.

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC’s public comment list, after subscribing to it by following the instructions at the “[Send A Comment](#)” button on the TC’s web page at <https://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 (<https://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. 06 November 2014. OASIS Committee Specification Draft 03 / Public Review Draft 03. <http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd03/obix-encodings-v1.0-csprd03.html>. Latest version: <http://docs.oasis-open.org/obix/obix-encodings/v1.0/obix-encodings-v1.0.html>.

Notices

Copyright © OASIS Open 2014. 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 <https://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** Efficient XML Interchange (EXI) Format 1.0 (Second Edition) , J. Schneider, T. Kamiya, D. Peintner, R. , Editors, W3C Proposed Edited Recommendation (work in progress), 22 October 2013, <http://www.w3.org/TR/2013/PER-exi-20131022/> . Latest version available at <http://www.w3.org/TR/exi/>
- RFC4627** Crockford, D., "The application/json Media type for JavaScript Object Notation (JSON)", RFC 4627, July 2007
- RFC768** Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980. <http://www.ietf.org/rfc/rfc0768.txt>

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** Efficient XML Interchange (EXI) Best Practices , M. , D. , Editors, W3C Working Draft (work in progress), 19 December 2007, <http://www.w3.org/TR/2007/WD-exi-best-practices-20071219/> . Latest version available at <http://www.w3.org/TR/exi-best-practices/>

31 2 XML Encoding

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

33 2.1 Design Philosophy

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

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

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

48 2.2 XML Syntax

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

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

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

57 2.3 XML Encoding

58 The following rules apply to encoding OBIX documents:

- 59 • OBIX documents MUST be well formed XML;
- 60 • OBIX documents SHOULD begin with XML Declaration specifying their encoding ;
- 61 • It is RECOMMENDED to use UTF-8 encoding without a byte order mark;
- 62 • OBIX documents MUST NOT include a Document Type Declaration – OBIX documents cannot
63 contain an internal or external subset;
- 64 • OBIX documents SHOULD include an XML Namespace definition. Convention is to declare the
65 default namespace of the document to "<http://docs.oasis-open.org/obix/ns/201410/schema>".

66 2.4 XML Decoding

67 The following rules apply to decoding of OBIX documents:

- 68 • MUST conform to XML processing rules as defined by XML 1.1;
- 69 • Documents which are not well formed XML MUST be rejected;

- 70 • Parsers are not required to understand a Document Type Declaration;
- 71 • Any unknown element MUST be ignored regardless of its XML namespace
- 72 • Any unknown attribute MUST be ignored regardless of its XML namespace

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

76 2.5 XML Namespace

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

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

79 The XML namespace for OBIX version 1.1 is:

```
80 http://docs.oasis-open.org/obix/ns/201410/schema
```

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

82 2.6 Namespace Prefixes in Contract Lists

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

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/201410/def". For example
89 the URI "obix:bool" is translated to " http://docs.oasis-open.org/obix/ns/201410/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/201410/def". All OBIX defined contracts are accessible via their HTTP URI using the
93 HTTP binding (at least they should be one day).

94 An example OBIX document with XML namespace prefixes normalized:

```
95 <obj xmlns:acme="http://acme.com/def/" href="acme:CustomPoint"  
96 is="acme:Point obix:Point"/>  
97  
98 <obj href="http://acme.com/def/CustomPoint"  
99 is="http://acme.com/def/Point http://docs.oasis-open.org/obix/ns/201410/def/Point"/>
```

100

3 OBIX Binary

101
102
103
104
105

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.

106
107
108
109

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.

110

3.1 Binary Overview

111
112
113

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:

114
115

```
7654 3210
MCCC CCVV
```

116
117
118

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.

119

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

120
121
122
123

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

124
125
126
127
128
129
130
131

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.

132

3.2 Binary Constants

133
134
135

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

136 *Table 3-1 Binary Constants*

137 3.3 Value Encodings

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

142 3.3.1 Bool Encodings

143 The following boolean encodings are supported:

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

144 *Table 3-2 Bool Encodings*

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

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

148 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
 149 complete encoding is the single byte 0x08. When val is true, we bitwise OR 0x08 with 0x01 with a result
 150 of 0x09.

151 3.3.2 Int Encodings

152 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

153 *Table 3-3 Int Encodings*

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

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

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

166 3.3.3 Real Encodings

167 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

168 *Table 3-4 Real Encodings*

169 Examples:

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

172 3.3.4 Str Encodings

173 The following str encodings are supported:

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

174 *Table 3-5 Str Encodings*

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

181 Simple example which illustrates a null terminated string:

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

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

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

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

195 3.3.5 Abstime Encodings

196 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

197 *Table 3-6 Abstime Encodings*

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

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

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

209 3.3.6 Reltime Encodings

210 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

211 *Table 3-7 Reltime Encodings*

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

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

217 3.3.7 Time Encodings

218 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

219 *Table 3-8 Time Encodings*

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

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

224 3.3.8 Date Encodings

225 The following date encodings are supported:

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

226 *Table 3-9 Date Encodings*

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

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

230 3.3.9 Status Encodings

231 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

232 *Table 3-10 Status Encodings*

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

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

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

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

249 3.4 Facets

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

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

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

259 3.4.1 Custom Facets

260 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.

261 *Table 3-11 Custom Facets*

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

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

- 269 • bool
- 270 • int
- 271 • real
- 272 • str
- 273 • enum
- 274 • uri
- 275 • abstime
- 276 • reltime
- 277 • date
- 278 • time

279 Other types for the value object are not supported.

280

281 Examples:

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

285

286 **3.5 Children**

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

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

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

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

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

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


313

4 JSON encoding

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

317 JSON uses two structures for representing information:

- 318 • A collection of name/value pairs
- 319 • An ordered list of values

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

325 The following grammar is used to represent OBIX objects:

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

¹ <http://json.org/java/>

366 4.1 Object and value encoding rules

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

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

371 `<obj/> → {"obix":"obj"}`

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

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

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

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

378 XML:

```
379 <obj href="/a/">  
380   <obj name="b" href="b">  
381     <obj name="c"/>  
382     <ref name="d" href="d"/>  
383   </obj>  
384 </obj>
```

385 JSON:

```
386 {  
387   "obix": "obj",  
388   "href": "/a",  
389   "children": [{  
390     "obix": "obj",  
391     "name": "b",  
392     "href": "b",  
393     "children": [{  
394       "obix": "obj",  
395       "name": "c",  
396     }, {  
397       "obix": "ref",  
398       "name": "d",  
399       "href": "d",  
400     }  
401   ]  
402 }
```

403 4.1.1 Bool encoding

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

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

406 4.1.2 Int encoding

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

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

409 4.1.3 Real encoding

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

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

412 4.1.4 Other types and facets

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

415 4.2 XML Namespace

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

418 Object with namespace prefixes in use:

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

421 Object with normalized namespace information:

```
422 <obj href="http://acme.com/def/CustomPoint"  
423 is="http://acme.com/def/Point http://docs.oasis-open.org/obix/ns/201410/def/Point"/>
```

424 JSON encoded object with normalized namespace information:

```
425 {obix:"obj", href:"http://acme.com/def/CustomPoint", is:"http://acme.com/def/Point  
426 http://docs.oasis-open.org/obix/ns/201410/def/Point" }
```

427 4.3 Examples

428 The following examples illustrate the JSON encoding:

429 **Example – OBIX About:**

430 XML:

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

442 JSON:

```
443 {"obix":"obj", "name":"about", "children": [  
444 {"obix":"str", "name":"obixVersion", "val":"1.1"},  
445 {"obix":"str", "name":"serverName", "val":"obix"},  
446 {"obix":"abstime", "name":"serverTime", "val":"2006-02-08T09:40:55.000+05:00:00Z"},  
447 {"obix":"abstime", "name":"serverBootTime", "val":"2006-02-  
448 08T09:33:31.980+05:00:00Z"},  
449 {"obix":"str", "name":"vendorName", "val":"Acme, Inc."},  
450 {"obix":"uri", "name":"vendorURL", "val":"http://www.acme.com"},  
451 {"obix":"str", "name":"productName", "val":"Acme OBIX Server"},  
452 {"obix":"str", "name":"productVersion", "val":"1.0.3"},  
453 {"obix":"uri", "name":"productUrl", "val":"http://www.acme.com/obix"}  
454 ]}]
```

455 4.4 MIME Type

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

458 5 EXI encoding

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

463 5.1 EXI options

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

466 If EXI encoding is used for OBIX the following options **MUST** be used by a client and server
467 implementation.

468 5.1.1 Alignment options

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

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

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

476 5.1.2 Preservation options

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

480 For OBIX only name space declarations **MUST** be preserved. Every other non-relevant information **MAY**
481 be omitted.

482 5.2 Non-schema-informed EXI

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

486 5.3 Schema-informed EXI

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

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

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

496 **5.4 MIME types**

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

500 **6 Conformance**

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

503 Appendix A. Acknowledgments

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

506 **Participants:**

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

539

Appendix B. Revision History

540

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
wd12	17 Apr 14	Markus Jung	OBIX-209, OBIX-208
wd13	26 May 14	Markus Jung	OBIX-153, OBIX-151, OBIX-149, OBIX-145, preparation for public review
Wd14	5 Nov 14	Toby Considine	Cleaned up template used, namespace

541