



# Encodings for ~~e~~oOBIX: Common Encodings Version 1.0

Committee Specification Draft ~~01~~02 /  
Public Review Draft ~~01~~02

~~11 July~~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>

### Previous version:

N/A

### 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](mailto:toby.considine@unc.edu)), University of North Carolina at Chapel Hill

### Editor:

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

## Related work:

This specification is related to:

- ~~e~~oOBIX Version 1.1. ~~11 July 2013. OASIS Committee Specification Draft 01 / Public Review Draft 01. Edited by Craig Gemmill. Latest version.~~ <http://docs.oasis-open.org/obix/obix/v1.1/obix-v1.1.html>.
- ~~Bindings for eOBIX: REST Bindings Version 1.0. 11 July 2013. OASIS Committee Specification Draft 01 / Public Review Draft 01. 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 eOBIX: 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>~~11 July 2013. OASIS Committee Specification Draft 01 / Public Review Draft 01.~~

- [Bindings for OBIX: Web Socket Bindings Version 1.0](http://docs.oasis-open.org/obix/obix-websocket/v1.0/obix-websocket-v1.0.html). 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 **eOBIX** objects adhering to the **eOBIX** object model. ~~oBIX provides the core information model and interaction pattern for communication with building control systems. Specific implementations of oBIX must choose how to encode oBIX information. The core specification describes an XML encoding, which is used in all examples in that document. This document specifies common alternate encodings, including CoAP, EXI, and JSON.~~

**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:

**[eOBIX-Encodings]**

*Encodings for eOBIX: Common Encodings Version 1.0*. ~~11 July~~ **19 December** 2013. OASIS Committee Specification Draft ~~0402~~ / Public Review Draft ~~0402~~. <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.....	9
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 **eOBIX**.

## 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>.
- ~~eBIX~~ **eOBIX** ~~OBIX~~ *OBIX Version 1.1. Edited by Craig Gemmill. Latest version.* <http://docs.oasis-open.org/obix/obix/v1.1/obix-v1.1.html>  
~~See link in "Related work" section on cover page.~~
- 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** ~~RT~~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

---

## 2 XML Encoding

28 | This chapter specifies how the eOBIX object model is encoded in XML.

### 2.1 Design Philosophy

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

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

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

### 2.2 XML Syntax

47 | The eOBIX XML syntax maps very closely to the abstract object model. The syntax is summarized:

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

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

### 2.3 XML Encoding

56 | The following rules apply to encoding eOBIX documents:

- 57 | • eOBIX documents MUST be well formed XML;
- 58 | • eOBIX documents SHOULD begin with XML Declaration specifying their encoding ;
- 59 | • It is RECOMMENDED to use UTF-8 encoding without a byte order mark;
- 60 | • eOBIX documents MUST NOT include a Document Type Declaration – eOBIX documents cannot  
61 | contain an internal or external subset;
- 62 | • eOBIX documents SHOULD include an XML Namespace definition. Convention is to declare the  
63 | default namespace of the document to `"http://obix.org/ns/schema/1.1"`. ~~If eOBIX is embedded  
64 | inside another type of XML document, then the convention is to use "o" as the namespace prefix.  
65 | Note that the prefix "obix" SHOULD NOT be used (see Section -). "http://docs.oasis-  
66 | open.org/obix/ns/201312/schema"~~.



## 67 2.4 XML Decoding

68 The following rules apply to decoding of eOBIX documents:

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

74

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

## 78 2.5 XML Namespace

79 XML namespaces for standards within the eOBIX umbrella should conform to the following pattern:

```
80 http://ebixdocs.oasis-open.org/obix/ns/{spec}/{short-identifier\_and\_version}
```

81

82 The XML namespace for eOBIX version 1.1 is:

```
83 http://ebixdocs.oasis-open.org/obix/ns/201312/schema/1.1
```

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

## 85 2.6 Namespace Prefixes in Contract Lists

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

90

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

98

99 An example eOBIX document with XML namespace prefixes normalized:

```
100 <obj xmlns:acme="http://acme.com/def/" href="acme:CustomPoint"  
101 is="acme:Point obix:Point"/>  
102  
103 <obj href="http://acme.com/def/CustomPoint"  
104 is="http://acme.com/def/Point http://ebix.orgebixdocs.oasis-  
105 open.org/obix/ns/201312/def/Point"/>
```

## 3 eOBIX Binary

In addition to the XML encoding, a binary encoding is defined for the eOBIX data model. The binary encoding allows eOBIX 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.

Full fidelity with eOBIX 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 eOBIX binary encoding is based strictly on the obix data model itself, not its XML InfoSet.

### 3.1 Binary Overview

The eOBIX data model is comprised of 16 object types (elements in XML) and 19 facets (attributes in XML). The eOBIX 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:

```
7654 3210
MCCC CCVV
```

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.

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

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

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.

### 3.2 Binary Constants

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

142 [Table 3-1 Binary Constants](#)

### 143 3.3 Value Encodings

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

#### 148 3.3.1 Bool Encodings

149 The following boolean encodings are supported:

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

150 [Table 3-2 Bool Encodings](#)

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

```
152 • <bool val="false"/> => 08
153 • <bool val="true"/> => 09
```

154 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  
 155 complete encoding is the single byte 0x08. When val is true, we bitwise OR 0x08 with 0x01 with a result  
 156 of 0x09.

### 157 3.3.2 Int Encodings

158 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

159 [Table 3-3 Int Encodings](#)

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

```
162 <int val="34"/>          => 0C 22  
163 <int val="2093 "/>       => 0D 08 2D  
164 <int val="76000"/>      => 0E 00 01 28 E0  
165 <int val="-300"/>       => 0E FF FF FE D4  
166 <int val="12345678901"/> => 0F 00 00 00 02 DF DC 1C 35
```

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

### 172 3.3.3 Real Encodings

173 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

174 [Table 3-4 Real Encodings](#)

175 Examples:

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

### 178 3.3.4 Str Encodings

179 The following str encodings are supported:

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

180 [Table 3-5 Str Encodings](#)

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

187 Simple example which illustrates a null terminated string:

```
188 <str val="obix"/> => 14 6F 62 69 78 00
189
```

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

```
191 <obj>
192 <str val="abc"/>
193 <str val="abc"/>
194 </obj> => 84 04 14 61 62 63 00 15 00 00 44
```

195 The first byte 0x84 is the obj code masked with the more bit. The next byte 0x04 is the hasChildren  
196 marker which indicates that children objects follow (covered further in section 3.5). The next byte is the  
197 0x14 str obj code masked with the 0x00 utf8 value code followed by the 61 62 63 00 encoding of "abc".  
198 The next byte 0x15 is the str obj type 0x14 masked with the 0x01 prev value code, followed by the u2  
199 encoding of index zero which references string value zero "abc". The last byte 0x44 is the end of  
200 children marker.

### 201 3.3.5 Abstime Encodings

202 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

203 [Table 3-6 Abstime Encodings](#)

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

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

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

### 215 3.3.6 Reltime Encodings

216 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

217 [Table 3-7 Reltime Encodings](#)

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

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

### 223 3.3.7 Time Encodings

224 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

225 [Table 3-8 Time Encodings](#)

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

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

### 230 3.3.8 Date Encodings

231 The following date encodings are supported:

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

232 [Table 3-9 Date Encodings](#)

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

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

### 236 3.3.9 Status Encodings

237 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

238 [Table 3-10 Status Encodings](#)

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

```
242 <obj status="ok"/> => 04
243 <obj status="disabled"/> => 84 4C // 0x4C | 0x00
244 <obj status="fault"/> => 84 4D // 0x4C | 0x01
245 <obj status="down"/> => 84 4E // 0x4C | 0x02
246 <obj status="unackedAlarm"/> => 84 4F // 0x4C | 0x03
247 <obj status="alarm"/> => 84 50 // 0x50 | 0x00
248 <obj status="unacked"/> => 84 51 // 0x50 | 0x01
249 <obj status="overridden"/> => 84 52 // 0x50 | 0x02
```

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

253 overridden use facet code status-1. It is illegal for a single object to define both the status-0 and status-1  
 254 facet codes.  
 255

### 256 3.4 Facets

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

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

264 Note that a string of multiple facets is indicated by masking the 0x80 more bit into the object/facet  
 265 headers.  
 266

#### 267 3.4.1 Custom Facets

268 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.

269 [Table 3-11 Custom Facets](#)

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

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

- 277 • bool
- 278 • int
- 279 • real
- 280 • str
- 281 • enum
- 282 • uri
- 283 • abstime
- 284 • reltime
- 285 • date
- 286 • time

287 Other types for the value object are not supported.  
 288

289 Examples:

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

293

## 294 3.5 Children

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

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

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

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

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

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



---

## 4 JSON encoding

321

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

325 JSON uses two structures for representing information:

- 326 • A collection of name/value pairs
- 327 • An ordered list of values

328 In JSON an object is an unordered set of name/value pairs and the encoding of an object starts with a left  
329 brace and ends with a right brace. A colon is used to separate the name and the value and a comma  
330 separates multiple name/value pairs. The JSON encoding of eOBIX is inspired by JSONML, which  
331 provides a lossless two-way conversation between JSON and XML. A Java reference implementation can  
332 be found here<sup>1</sup>.

333 The following grammar is used to represent eOBIX objects:

334

```
335 element  
336 = '{' tagobix-identifier ',' attribute-list ', "nodes":{"children":[' element-list ']]}'  
337 | '{' tagobix-identifier ',' attribute-list '}'  
338 | '{' tagobix-identifier ', "nodes":{"children":[' element-list ']]}'  
339 | '{' tagobix-identifier '}'  
340 | string  
341 ;  
342  
343 tagobix-identifier  
344 = "tag":tagobix:type-name  
345 ;  
346  
347 tagtype-name  
348 = string  
349 ;  
350  
351 attribute-list  
352 = attribute ',' attribute-list  
353 | attribute  
354 ;  
355  
356 attribute  
357 = attribute-name ':' attribute-value  
358 ;  
359  
360 attribute-name  
361 = string  
362 ;  
363  
364 attribute-value  
365 = string  
366 | number  
367 | 'true'  
368 | 'false'  
369 ;  
370  
371 element-list  
372 = element ',' element-list  
373 | element  
374 ;
```

---

<sup>1</sup> <http://json.org/java/>

## 375 4.1 Object and value encoding rules

376 | Objects MUST be encoded according to the grammar given above. The eOBIX object is encoded as  
377 | JSON object which an unordered list of name/value pairs. The object type which is used as element  
378 | name in XML is encoded as a name/value pair using `"tag"` as name and the object type as string value.

379 | The XML and JSON representation of a simple `obj`:

380 | `<obj/>` → `{ "tag": "obj" }`

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

382 | `<obj name="myName" href="/myHref">` → `{ "tag": "obj", "name": "myName", "href": "/myHref" }`  
383 | `</obj>`

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

386 | The XML representation of an object with extend is mapped to the JSON representation as shown in the  
387 | examples below.

388 | XML:

389 | `<obj href="/a/">`  
390 | `<obj name="b" href="b">`  
391 | `<obj name="c"/>`  
392 | `<ref name="d" href="d"/>`  
393 | `</obj>`  
394 | `</obj>`

395 | JSON:

396 | `{ "tag": "obj",`  
397 |  `"href": "/a/", "nodes": [`  
398 |  `{ "tag": "obj", "children": [`  
399 |  `{ "tag": "obj",`  
400 |  `"name": "b",`  
401 |  `"href": "b", "nodes": [`  
402 |  `{ "tag": "obj", "children": [`  
403 |  `{ "tag": "obj",`  
404 |  `"name": "c",`  
405 |  `"tag": "ref",`  
406 |  `"name": "d",`  
407 |  `"href": "d",`  
408 |  `} ]`  
409 |  `} ]`  
410 |  `} ]`  
411 |  `} ]`  
412 |  `} ]`  
413 |  `} ]`  
414 | `}`

### 413 4.1.1 Bool encoding

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

415 | `<bool val="true"/>` → `{ "tag": "bool", "val": true }`

### 416 4.1.2 Int encoding

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

418 | `<int val="5"/>` → `{ "tag": "int", "val": 5 }`

### 419 4.1.3 Real encoding

420 | The `xs:longdouble` val attribute of the `intreal` object is mapped to the number representation of JSON.

421 | `<real val="5.5"/>` → `{ "tag": "real", "val": 5.5 }`

### 422 4.1.4 Other types and facets

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

## 4.2 XML Namespace

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

Object with namespace prefixes in use:

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

Object with normalized namespace information:

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

JSON encoded object with normalized namespace information:

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

## 4.3 Examples

The following examples illustrate the JSON encoding:

### Example – eOBIX About:

XML:

```
<obj name="about">
  <str name="obixVersion" val="1.1"/>
  <str name="serverName" val="obix"/>
  <abstime name="serverTime" val="2006-02-08T09:40:55.000+05:00:00Z"/>
  <abstime name="serverBootTime" val="2006-02-08T09:33:31.980+05:00:00Z"/>
  <str name="vendorName" val="Acme, Inc."/>
  <uri name="vendorUrl" val="http://www.acme.com"/>
  <str name="productName" val="Acme eOBIX Server"/>
  <str name="productVersion" val="1.0.3"/>
  <uri name="productUrl" val="http://www.acme.com/obix"/>
</obj>
```

JSON:

```
{tag:"obix:"obj", name:"about", nodes:{}, children:[
  {tag:"obix:"str", name:"obixVersion", val:"1.1"},
  {tag:"obix:"str", name:"serverName", val:"obix"},
  {tag:"obix:"abstime", name:"serverTime", val:"2006-02-
08T09:40:55.000+05:00:00Z"},
  {tag:"obix:"abstime", name:"serverBootTime", val:"2006-02-
08T09:33:31.980+05:00:00Z"},
  {tag:"obix:"str", name:"vendorName", val:"Acme, Inc."},
  {tag:"obix:"uri", name:"vendorURL",
val:"http://www.acme.com"},
  {tag:"obix:"str", name:"productName", val:"Acme eOBIX
Server"},
  {tag:"obix:"str", name:"productVersion", val:"1.0.3"},
  {tag:"obix:"uri", name:"prodctUrl", val:"http://www.acme.com/obix"}
]}
```

## 4.4 MIME Type

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

---

## 477 5 EXI encoding

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

### 482 5.1 EXI options

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

485 | If EXI encoding is used for eOBIX the following options MUST be used by a client and server  
486 implementation.

#### 487 5.1.1 Alignment options

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

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

494 | In order to have the best possible compression for eOBIX `bitPacked` alignment MUST be used.

#### 495 5.1.2 Preservation options

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

499 | For eOBIX only name space declarations MUST be preserved. Every other non-relevant information MAY  
500 be omitted.

## 501 5.2 Non-schema-informed EXI

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

## 505 5.3 Schema-informed EXI

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

510 | For schema-informed the normative obix.xsd schema file representing the eOBIX 1.1 object model MUST  
511 be used in order to provide interoperability among different vendor implementations.

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

515 **5.4 MIME types**

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

---

519 **6 Conformance**

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

---

## 522 Appendix A. Acknowledgments

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

### 525 Participants:

526 Ron Ambrosio, IBM  
527 Brad Benson, Trane  
528 Ron Bernstein, LonMark International\*  
529 Ludo Bertsch, Continental Automated Buildings Association (CABA)  
530 Chris Bogen, US Department of Defense  
531 Rich Blomseth, Echelon Corporation  
532 Anto Budiardjo, Clasma Events, Inc.  
533 Jochen Burkhardt, IBM  
534 JungIn Choi, Kyungwon University  
535 David Clute, Cisco Systems, Inc.\*  
536 Toby Considine, University of North Carolina at Chapel Hill  
537 William Cox, Individual  
538 Robert Dolin, Echelon Corporation  
539 Marek Dzedzic, Treasury Board of Canada, Secretariat  
540 Brian Frank, SkyFoundry  
541 Craig Gemmill, Tridium, Inc.  
542 Matthew Giannini, Tridium, Inc.  
543 ~~Harald Hofstätter, Institute of Computer Aided Automation, Vienna University of Technology~~  
544 ~~Markus Jung, Institute of Computer Aided Automation, Vienna University of Technology~~  
545 Christopher Kelly, Cisco Systems  
546 Wonsuk Ko, Kyungwon University  
547 Perry Krol, TIBCO Software Inc.  
548 Corey Leong, Individual  
549 Ulf Magnusson, Schneider Electric  
550 Brian Meyers, Trane  
551 Jeremy Roberts, LonMark International  
552 Thorsten Roggendorf, Echelon Corporation  
553 Anno Scholten, Individual  
554 John Sublett, Tridium, Inc.  
555 Dave Uden, Trane  
556 Ron Zimmer, Continental Automated Buildings Association (CABA)\*  
557 ~~Robert Zach, Institute of Computer Aided Automation~~  
558 Rob Zivney, Hirsch Electronics Corporation  
559 ~~Markus Jung, Vienna University of Technology~~  
560 Markus Jung, Institute of Computer Aided Automation, Vienna University of Technology  
561

562

## Appendix B. Revision History

563

Revision	Date	Editor	Changes Made
wd01	26 Mar 13	Markus Jung	Initial creation with XML and Binary encoding taken from the <a href="#">eOBIX 1.1 WD07</a> 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
<del>wd</del> wd06	8 Jul 13	Toby Considine	Updated acknowledgements
<a href="#">wd07</a>	<a href="#">2 Oct 13</a>	<a href="#">Markus Jung</a>	<a href="#">Jira: OBIX-7, OBIX-56, OBIX-5, OBIX-6, OBIX-48</a>
<a href="#">wd08</a>	<a href="#">7 Nov 13</a>	<a href="#">Markus Jung</a>	<a href="#">Namespace rules, using straight quotes within the document.</a>
<a href="#">wd09</a>	<a href="#">12 Dec 13</a>	<a href="#">Markus Jung</a>	<a href="#">Fixed minor error (JSON encoding for real). Update OBIX namespace to current policy.</a>
<a href="#">wd10</a>	<a href="#">16 Dec 13</a>	<a href="#">Markus Jung</a>	<a href="#">Updated namespace (including date), using uppercase for OBIX</a>
<a href="#">wd11</a>	<a href="#">16 Dec 13</a>	<a href="#">Markus Jung</a>	<a href="#">Minor fixes: OBIX-79</a>

564

565