

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA ACL Message Representation in Bit-Efficient Encoding Specification

Document title	FIPA ACL Message Representation in Bit-Efficient Encoding Specification		
Document number	XC00069F	Document source	FIPA TC Agent Management
Document status	Experimental	Date of this status	2002/11/01
Supersedes	FIPA00024		
Contact	fab@fipa.org		
Change history	See <i>Informative Annex A — ChangeLog</i>		

© 1996-2002 Foundation for Intelligent Physical Agents
<http://www.fipa.org/>
Geneva, Switzerland

Notice

Use of the technologies described in this specification may infringe patents, copyrights or other intellectual property rights of FIPA Members and non-members. Nothing in this specification should be construed as granting permission to use any of the technologies described. Anyone planning to make use of technology covered by the intellectual property rights of others should first obtain permission from the holder(s) of the rights. FIPA strongly encourages anyone implementing any part of this specification to determine first whether part(s) sought to be implemented are covered by the intellectual property of others, and, if so, to obtain appropriate licenses or other permission from the holder(s) of such intellectual property prior to implementation. This specification is subject to change without notice. Neither FIPA nor any of its Members accept any responsibility whatsoever for damages or liability, direct or consequential, which may result from the use of this specification.

21 **Foreword**

22 The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the
23 industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-
24 based applications. This occurs through open collaboration among its member organizations, which are companies and
25 universities that are active in the field of agents. FIPA makes the results of its activities available to all interested parties
26 and intends to contribute its results to the appropriate formal standards bodies where appropriate.

27 The members of FIPA are individually and collectively committed to open competition in the development of agent-
28 based applications, services and equipment. Membership in FIPA is open to any corporation and individual firm,
29 partnership, governmental body or international organization without restriction. In particular, members are not bound to
30 implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their
31 participation in FIPA.

32 The FIPA specifications are developed through direct involvement of the FIPA membership. The status of a
33 specification can be either Preliminary, Experimental, Standard, Deprecated or Obsolete. More detail about the process
34 of specification may be found in the FIPA Document Policy [f-out-00000] and the FIPA Specifications Policy [f-out-
35 00003]. A complete overview of the FIPA specifications and their current status may be found on the FIPA Web site.

36 FIPA is a non-profit association registered in Geneva, Switzerland. As of June 2002, the 56 members of FIPA
37 represented many countries worldwide. Further information about FIPA as an organization, membership information,
38 FIPA specifications and upcoming meetings may be found on the FIPA Web site at <http://www.fipa.org/>.

39 **Contents**

40	1	Scope.....	1
41	2	Bit-Efficient ACL Representation	2
42	2.1	Component Name	2
43	2.2	Syntax.....	2
44	2.3	Using Dynamic Code Tables.....	5
45	2.4	Notes on the Grammar Rules.....	7
46	3	References	9
47	4	Informative Annex A — ChangeLog.....	10
48	4.1	2002/11/01 – version F by TC X2S	10

49 **1 Scope**

50 This document deals with message transportation between inter-operating agents and also forms part of the FIPA
51 Agent Management Specification [FIPA00023]. It contains specifications for:

- 52
- 53 • Syntactic representation of ACL in a bit-efficient form.
- 54

55 2 Bit-Efficient ACL Representation

56 This section defines the message transport syntax for a bit-efficient encoding which is expressed in standard EBNF
57 format¹ (see *Table 1*).

58

Grammar rule component	Example
Terminal tokens are enclosed in double quotes	" ("
Non-terminals are written as capitalised identifiers	Expression
Square brackets denote an optional construct	[", " OptionalArg]
Vertical bars denote an alternative between choices	Integer Float
Asterisk denotes zero or more repetitions of the preceding expression	Digit*
Plus denotes one or more repetitions of the preceding expression	Alpha+
Parentheses are used to group expansions	(A B)*
Productions are written with the non-terminal name on the left-hand side, expansion on the right-hand side and terminated by a full stop	ANonTerminal = "terminal".
0x?? is a hexadecimal byte	0x00

59

60

61

Table 1: EBNF Rules

62 2.1 Component Name

63 The name assigned to this component is:

64

65 fipa.acl.rep.bitefficient.std

66

67 2.2 Syntax

68 ACLCommunicativeAct

= Message.

69

70 Message

= Header MessageType MessageParameter* EndofMsg.

71

72 Header

= MessageId Version.

73

74 MessageId

= 0xFA

| 0xFB

| 0xFC.

/* See comment 1 below */

75

76 Version

= Byte.

/* See comment 2 below */

77

78 EndofMsg

= EndOfCollection.

79

80 EndOfCollection

= 0x01.

81

82 MessageType

= PredefinedMsgType

| UserDefinedMsgType.

/* See comment 3 below */

83

84 UserDefinedMsgType

= 0x00 MsgTypeName.

85

86 MsgTypeName

= BinWord.

87

88 MessageParameter

= PredefinedParam

| UserDefinedMsgParam.

/* See comment 4 below */

89

90 UserDefinedMsgParam

= 0x00 ParameterName ParameterValue.

91

92 ParameterName

= BinWord.

93

¹ White space is not allowed between tokens.

```

97
98 ParamterValue          = BinExpression.
99
100 PredefinedMsgType      = 0x01          /* accept-proposal */
101                        | 0x02          /* agree           */
102                        | 0x03          /* cancel          */
103                        | 0x04          /* cfp             */
104                        | 0x05          /* confirm         */
105                        | 0x06          /* disconfirm      */
106                        | 0x07          /* failure         */
107                        | 0x08          /* inform          */
108                        | 0x09          /* inform-if       */
109                        | 0x0a          /* inform-ref      */
110                        | 0x0b          /* not-understood  */
111                        | 0x0c          /* propagate       */
112                        | 0x0d          /* propose         */
113                        | 0x0e          /* proxy           */
114                        | 0x0f          /* query-if        */
115                        | 0x10          /* query-ref       */
116                        | 0x11          /* refuse          */
117                        | 0x12          /* reject-proposal */
118                        | 0x13          /* request         */
119                        | 0x14          /* request-when    */
120                        | 0x15          /* request-whenever */
121                        | 0x16          /* subscribe       */
122
123 PredefinedMsgParam     = 0x02 AgentIdentifier /* sender          */
124                        | 0x03 RecipientExpr  /* receiver        */
125                        | 0x04 MsgContent     /* content         */
126                        | 0x05 ReplyWithParam /* reply-with      */
127                        | 0x06 ReplyByParam   /* reply-by        */
128                        | 0x07 InReplyToParam /* in-reply-to     */
129                        | 0x08 ReplyToParam   /* reply-to        */
130                        | 0x09 Language       /* language        */
131                        | 0x0a Encoding       /* encoding        */
132                        | 0x0b Ontology       /* ontology        */
133                        | 0x0c Protocol       /* protocol        */
134                        | 0x0d ConversationID /* conversation-id */
135
136 AgentIdentifier        = 0x02 AgentName
137                        [Addresses]
138                        [Resolvers]
139                        (UserDefinedParameter)*
140                        EndOfCollection.
141
142 AgentName              = BinWord.
143
144 Addresses              = 0x02 UrlCollection.
145
146 Resolvers              = 0x03 AgentIdentifierCollection.
147
148 UserDefinedParameter  = 0x04 BinWord BinExpression.
149
150 UrlCollection          = (Url)* EndofCollection.
151
152 Url                   = BinWord.
153
154 AgentIdentifierCollection
155                       = (AgentIdentifier)* EndOfCollection.
156
157 RecipientExpr         = AgentIdentifierCollection.
158
159 MsgContent            = BinString.
160

```

```

161 ReplyWithParam      = BinExpression.
162
163 ReplyByParam        = BinDateTimeToken.
164
165 InReplyToParam      = BinExpression.
166
167 ReplyToParam        = RecipientExpr.
168
169 Language            = BinExpression.
170
171 Encoding            = BinExpression.
172
173 Ontology            = BinExpression.
174
175 Protocol            = BinWord.
176
177 ConversationID      = BinExpression.
178
179 BinWord             = 0x10 Word 0x00
180                   | 0x11 Index.
181
182 BinNumber           = 0x12 Digits          /* Decimal number      */
183                   | 0x13 Digits.          /* Hexadecimal number */
184
185 Digits              = CodedNumber+.
186
187 BinString           = 0x14 String 0x00     /* New string literal  */ /*
188                   | 0x15 Index           /* String literal from code table */
189                   | 0x16 Len8 ByteSeq    /* New ByteLengthEncoded string */
190                   | 0x17 Len16 ByteSeq   /* New ByteLengthEncoded string */
191                   | 0x18 Index          /* ByteLengthEncoded from code table */
192                   | 0x19 Len32 ByteSeq.  /* New ByteLengthEncoded string */ /*
193
194 BinDateTimeToken    = 0x20 BinDate         /* Absolute time      */ /*
195                   | 0x21 BinDate         /* Relative time (+) */ /*
196                   | 0x22 BinDate         /* Relative time (-) */ /*
197                   | 0x24 BinDate TypeDesignator /* Absolute time      */ /*
198                   | 0x25 BinDate TypeDesignator /* Relative time (+) */ /*
199                   | 0x26 BinDate TypeDesignator. /* Relative time (-) */ /*
200
201 BinDate             = Year Month Day Hour Minute Second Millisecond.
202                   /* See comment 8 below */
203
204 BinExpression       = BinExpr
205                   | 0xFF BinString.      /* See comment 9 below */
206
207 BinExpr             = BinWord
208                   | BinString
209                   | BinNumber
210                   | ExprStart BinExpr* ExprEnd.
211
212 ExprStart           =i 0x60              /* Level down (i.e. '(' -character) */ /*
213                   | 0x70 Word 0x00      /* Level down, new word follows      */ /*
214                   | 0x71 Index          /* Level down, word code follows     */ /*
215                   | 0x72 Digits         /* Level down, number follows        */ /*
216                   | 0x73 Digits         /* Level down, hex number follows    */ /*
217                   | 0x74 String 0x00    /* Level down, new string follows    */ /*
218                   | 0x75 Index          /* Level down, string code follows   */ /*
219                   | 0x76 Len8 String    /* Level down, new byte string (1 byte) */ /*
220                   | 0x77 Len16 String   /* Level down, new byte string (2 byte) */ /*
221                   | 0x78 Len32 String   /* Level down, new byte string (4 byte) */ /*
222                   | 0x79 Index.         /* Level down, byte string code follows */ /*
223
224 ExprEnd             = 0x40              /* Level up (i.e. ')' -character)   */ /*

```

```

225 | 0x50 Word 0x00 /* Level up, new word follows */
226 | 0x51 Index /* Level up, word code follows */
227 | 0x52 Digits /* Level up, number follows */
228 | 0x53 Digits /* Level up, hexadecimal number follows */
229 | 0x54 String 0x00 /* Level up, new string follows */
230 | 0x55 Index /* Level up, string code follows */
231 | 0x56 Len8 String /* Level up, new byte string (1 byte) */
232 | 0x57 Len16 String /* Level up, new byte string (2 byte) */
233 | 0x58 Len32 String /* Level up, new byte string (4 byte) */
234 | 0x59 Index. /* Level up, byte string code follows */
235
236 ByteSeq = Byte*.
237
238 Index = Byte
239 | Short. /* See comment 6 below */
240
241 Len8 = Byte. /* See comment 7 below */
242
243 Len16 = Short. /* See comment 7 below */
244
245 Len32 = Long. /* See comment 7 below */
246
247 Year = Byte Byte.
248
249 Month = Byte.
250
251 Day = Byte.
252
253 Hour = Byte.
254
255 Minute = Byte.
256
257 Second = Byte.
258
259 Millisecond = Byte Byte.
260
261 Word = /* as in [FIPA00070] */
262
263 String = /* as in [FIPA00070] */
264
265 CodedNumber = /* See comment 5 below */
266
267 TypeDesignator = /* as in [FIPA00070] */
268

```

269 2.3 Using Dynamic Code Tables

270 The transport syntax can be used with or without dynamic code table. Using dynamic code tables is an optional feature,
 271 which gives more compact output but might not be appropriate if communicating peers does not have sufficient memory
 272 (for example, in case of low-end PDAs or smart phones).

273
 274 To use dynamic code tables the encoder inserts new entries (for example, `Word`, `String`, etc.) into a code table while
 275 constructing bit-efficient representation for ACL message. The code table is initially empty and whenever a new entry is
 276 added to the code table, the smallest available code number is allocated to it. There is no need to transfer these index
 277 codes explicitly over the communication channel. Once the code table becomes full and a new code needs to be added,
 278 the sender first removes $size \gg 3^2$ entries from the code table using a Least Recently Used (LRU) algorithm and then
 279 adds a new entry to code table. For example, should the code table size be 512 entries, 64 entries are removed.
 280 Correspondingly the decoder removes entries from the code table when it receives a new entry from the encoder.

281

² Right shifted by 3 bit positions – approximately 10%.

282 The size of the code table, if used, is between 256 (2^8) and 65536 (2^{16}) entries. The output of this code table is always
283 one or two bytes (one byte only when the code table size is 2^8). Using two-byte output code wastes some bits, but
284 allows for much faster parsing of messages. The code table is unidirectional, that is, if sender A adds something to the
285 code table when sending a message to B, then B cannot use this code table entry when sending a message back to A.
286

287 Both peers must agree the code table size before its usage; this process is not part of this specification. Furthermore,
288 having more compact output, one code table should be applied to more than one message; the method of mapping
289 messages to appropriate code table is not part of this specification.
290

2.4 Notes on the Grammar Rules

1. The first byte defines the message identifier. The identifier byte can be used to separate bit-efficient ACL messages from (for example) string-based messages and separate different coding schemes. The value 0xFA defines a bit-efficient coding scheme without dynamic code tables and the value 0xFB defines a bit-efficient coding scheme with dynamic code tables. The message identifier 0xFC is used when dynamic code tables are being used, but the sender does not want to update code tables (even if message contains strings that should be added to code table).
2. The second byte defines the version number. The version number byte contains the major version number in the upper four bits and minor version number in the lower four bits. This specification defines version 1.0 (coded as 0x10).
3. All message types defined in this specification have a predefined code. If an encoder sends an ACL message with a message type which has no predefined code, it must use the extension mechanism which adds a new message type into code table (if code tables are being used).
4. All message parameters defined in this specification have a predefined code. If a message contains a user defined message parameter, an extension mechanism is used (byte 0x00) and new entry is added to code table (if code table is used).
5. Numbers are coded by reserving four bits for each digit in the number's ASCII representation, that is, two ASCII numbers are coded into one byte. Table 1 shows a 4-bit code for each number and special codes that may appear in ASCII coded numbers.

If the ASCII presentation of a number contains odd number characters, the last four bits of the coded number are set to zero (the `Padding` token), otherwise an additional 0x00 byte is added to end of coded number. If the number to be coded is integer, decimal number, or octal number, the identifier byte 0x12 is used. For hexadecimal numbers, the identifier byte 0x13 is used. Hexadecimal numbers are converted to integers before coding (the coding scheme does not allow characters from a through f to appear in number form).

Numbers are never added to a dynamic code table.

Token	Code	Token	Code
Padding	0000	7	1000
0	0001	8	1001
1	0010	9	1010
2	0011	+	1100
3	0100	E	1101
4	0101	-	1110
5	0110	.	1111
6	0111		

Table 1: Binary Representation of Number Tokens

6. `Index` is a pointer to code table entry and its size (in bits) depends on the code table size. If the code table size is 256 entries, the size of the index is one byte; otherwise its size is two bytes (represented in network byte order).
7. `Byte` is a one-byte code word, `Short` is a short integer (two bytes, network byte order) and `Long` is a long integer (four bytes, network byte order).

- 331 8. Dates are coded as numbers, that is, four bits are reserved for each ASCII number (see comment 5 above).
332 Information whether the type designator is present or not, is coded into identifier byte. These fields always have
333 static length (two bytes for year and milliseconds, one byte for other components).
334
- 335 9. None of the actual content of the message (the information contained in the `content` parameter of the ACL
336 message) is coded nor are any of its components are added to a code table.
337

338 **3 References**

339 [FIPA00023] FIPA Agent Management Specification. Foundation for Intelligent Physical Agents, 2000.
340 <http://www.fipa.org/specs/fipa00023/>

341 [FIPA00067] FIPA Agent Message Transport Service Specification. Foundation for Intelligent Physical Agents, 2000.
342 <http://www.fipa.org/specs/fipa00067/>

343 [FIPA00070] FIPA ACL Message Representation in String Specification. Foundation for Intelligent Physical Agents,
344 2000.
345 <http://www.fipa.org/specs/fipa00070/>

346 [FIPA00075] FIPA Agent Message Transport Protocol for IOP Specification. Foundation for Intelligent Physical
347 Agents, 2000.
348 <http://www.fipa.org/specs/fipa00075/>
349

350 **4 Informative Annex A — ChangeLog**

351 **4.1 2002/11/01 – version F by TC X2S**

352 Page 2, line 56: Removed sentence on compatibility issue with FIPA00075

353 **Page 4, line 158: `MsgContent` value changed from `BinExpression` to `BinString`**

354 **Page 4, line 193: Added signs to `BinDateTimeToken`**

355