1

2 **FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS**

3

4

5 # FIPA ACL Message Representation in
6 # Bit-Efficient Encoding Specification

7

Document title	FIPA ACL Message Representation in Bit-Efficient Encoding Specification		
Document number	SC00069G	Document source	FIPA TC Agent Management
Document status	Standard	Date of this status	2002/12/03
Supersedes	FIPA00024		
Contact	fab@fipa.org		
Change history	See *Informative Annex A — ChangeLog*		

8

9

10

11

12

13

14

15

16

17

18 © 1996-2002 Foundation for Intelligent Physical Agents
19 http://www.fipa.org/
20 *Geneva, Switzerland*

Foreword

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

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

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

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

Contents

1 Scope

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

- Syntactic representation of ACL in a bit-efficient form.

56 **2   Bit-Efficient ACL Representation**

57 This section defines the message transport syntax for a bit-efficient encoding which is expressed in standard EBNF
58 format[1] (see *Table 1*).
59

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`

60
61   **Table 1:** EBNF Rules
62

63 **2.1   Component Name**

64 The name assigned to this component is:
65
66 `fipa.acl.rep.bitefficient.std`
67

68 **2.2   Syntax**

```
69 ACLCommunicativeAct      = Message.
70
71 Message                  = Header MessageType MessageParameter* EndofMsg.
72
73 Header                   = MessageId Version.
74
75 MessageId                = 0xFA
76                          | 0xFB
77                          | 0xFC.                     /* See comment 1 below */
78
79 Version                  = Byte.                      /* See comment 2 below */
80
81 EndofMsg                 = EndOfCollection.
82
83 EndOfCollection          = 0x01.
84
85 MessageType              = PredefinedMsgType
86                          | UserDefinedMsgType.        /* See comment 3 below */
87
88 UserDefinedMsgType       = 0x00 MsgTypeName.
89
90 MsgTypeName              = BinWord.
91
92 MessageParameter         = PredefinedParam
93                          | UserDefinedMsgParam.       /* See comment 4 below */
94
95 UserDefinedMsgParam      = 0x00 ParameterName ParameterValue.
96
97 ParameterName            = BinWord.
```

---

[1] White space is not allowed between tokens.

```
 98
 99    ParamterValue          = BinExpression.
100
101    PredefinedMsgType      = 0x01                   /* accept-proposal  */
102                           | 0x02                   /* agree            */
103                           | 0x03                   /* cancel           */
104                           | 0x04                   /* cfp              */
105                           | 0x05                   /* confirm          */
106                           | 0x06                   /* disconfirm       */
107                           | 0x07                   /* failure          */
108                           | 0x08                   /* inform           */
109                           | 0x09                   /* inform-if        */
110                           | 0x0a                   /* inform-ref       */
111                           | 0x0b                   /* not-understood   */
112                           | 0x0c                   /* propagate        */
113                           | 0x0d                   /* propose          */
114                           | 0x0e                   /* proxy            */
115                           | 0x0f                   /* query-if         */
116                           | 0x10                   /* query-ref        */
117                           | 0x11                   /* refuse           */
118                           | 0x12                   /* reject-proposal  */
119                           | 0x13                   /* request          */
120                           | 0x14                   /* request-when     */
121                           | 0x15                   /* request-whenever */
122                           | 0x16.                  /* subscribe        */
123
124    PredefinedMsgParam     = 0x02 AgentIdentifier  /* sender           */
125                           | 0x03 RecipientExpr    /* receiver         */
126                           | 0x04 MsgContent       /* content          */
127                           | 0x05 ReplyWithParam   /* reply-with       */
128                           | 0x06 ReplyByParam     /* reply-by         */
129                           | 0x07 InReplyToParam   /* in-reply-to      */
130                           | 0x08 ReplyToParam     /* reply-to         */
131                           | 0x09 Language         /* language         */
132                           | 0x0a Encoding         /* encoding         */
133                           | 0x0b Ontology         /* ontology         */
134                           | 0x0c Protocol         /* protocol         */
135                           | 0x0d ConversationID.  /* conversation-id  */
136
137    AgentIdentifier        = 0x02 AgentName
138                             [Addresses]
139                             [Resolvers]
140                             (UserDefinedParameter)*
141                             EndOfCollection.
142
143    AgentName              = BinWord.
144
145    Addresses              = 0x02 UrlCollection.
146
147    Resolvers              = 0x03 AgentIdentifierCollection.
148
149    UserDefinedParameter   = 0x04 BinWord BinExpression.
150
151    UrlCollection          = (Url)* EndofCollection.
152
153    Url                    = BinWord.
154
155    AgentIdentifierCollection
156                           = (AgentIdentifier)* EndOfCollection.
157
158    RecipientExpr          = AgentIdentifierCollection.
159
160    MsgContent             = BinString.
161
```

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

```
226                                   | 0x50 Word 0x00    /* Level up, new word follows             */
227                                   | 0x51 Index        /* Level up, word code follows            */
228                                   | 0x52 Digits       /* Level up, number follows               */
229                                   | 0x53 Digits       /* Level up, hexadecimal number follows */
230                                   | 0x54 String 0x00  /* Level up, new string follows           */
231                                   | 0x55 Index        /* Level up, string code follows          */
232                                   | 0x56 Len8 String  /* Level up, new byte string (1 byte)    */
233                                   | 0x57 Len16 String /* Level up, new byte string (2 byte)    */
234                                   | 0x58 Len32 String /* Level up, new byte string (4 byte)    */
235                                   | 0x59 Index.       /* Level up, byte string code follows    */
236
237    ByteSeq              = Byte*.
238
239    Index                = Byte
240                         | Short.              /* See comment 6 below */
241
242    Len8                 = Byte.               /* See comment 7 below */
243
244    Len16                = Short.              /* See comment 7 below */
245
246    Len32                = Long.               /* See comment 7 below */
247
248    Year                 = Byte Byte.
249
250    Month                = Byte.
251
252    Day                  = Byte.
253
254    Hour                 = Byte.
255
256    Minute               = Byte.
257
258    Second               = Byte.
259
260    Millisecond          = Byte Byte.
261
262    Word                 = /* as in [FIPA00070]   */
263
264    String               = /* as in [FIPA00070]   */
265
266    CodedNumber          = /* See comment 5 below */
267
268    TypeDesignator       = /* as in [FIPA00070]   */
269
```

270   2.3   Using Dynamic Code Tables

271   The transport syntax can be used with or without dynamic code table. Using dynamic code tables is an optional feature,
272   which gives more compact output but might not be appropriate if communicating peers does not have sufficient memory
273   (for example, in case of low-end PDAs or smart phones).
274
275   To use dynamic code tables the encoder inserts new entries (for example, Word, String, etc.) into a code table while
276   constructing bit-efficient representation for ACL message. The code table is initially empty and whenever a new entry is
277   added to the code table, the smallest available code number is allocated to it. There is no need to transfer these index
278   codes explicitly over the communication channel. Once the code table becomes full and a new code needs to be added,
279   the sender first removes $size>>3$[2] entries from the code table using a Least Recently Used (LRU) algorithm and then
280   adds a new entry to code table. For example, should the code table size be 512 entries, 64 entries are removed.
281   Correspondingly the decoder removes entries from the code table when it receives a new entry from the encoder.
282

---

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

283 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
284 one or two bytes (one byte only when the code table size is 2^8). Using two-byte output code wastes some bits, but
285 allows for much faster parsing of messages. The code table is unidirectional, that is, if sender A adds something to the
286 code table when sending a message to B, then B cannot use this code table entry when sending a message back to A.
287
288 Both peers must agree the code table size before its usage; this process is not part of this specification. Furthermore,
289 having more compact output, one code table should be applied to more than one message; the method of mapping
290 messages to appropriate code table is not part of this specification.
291

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).

332   8.   Dates are coded as numbers, that is, four bits are reserved for each ASCII number (see comment 5 above).
333   Information whether the type designator is present or not, is coded into identifier byte. These fields always have
334   static length (two bytes for year and milliseconds, one byte for other components).
335

336   9.   None of the actual content of the message (the information contained in the `content` parameter of the ACL
337   message) is coded nor are any of its components are added to a code table.
338

339 **3   References**

340   [FIPA00023]   FIPA Agent Management Specification. Foundation for Intelligent Physical Agents, 2000.
341   `http://www.fipa.org/specs/fipa00023/`
342   [FIPA00067]   FIPA Agent Message Transport Service Specification. Foundation for Intelligent Physical Agents, 2000.
343   `http://www.fipa.org/specs/fipa00067/`
344   [FIPA00070]   FIPA ACL Message Representation in String Specification. Foundation for Intelligent Physical Agents,
345   2000.
346   `http://www.fipa.org/specs/fipa00070/`
347   [FIPA00075]   FIPA Agent Message Transport Protocol for IIOP Specification. Foundation for Intelligent Physical
348   Agents, 2000.
349   `http://www.fipa.org/specs/fipa00075/`
350

351 # 4   Informative Annex A — ChangeLog

352 ## 4.1   2002/11/01 - version F by TC X2S

353 Page 2, line 56:   Removed sentence on compatibility issue with FIPA00075
354 **Page 4, line 158:   `MsgContent` value changed from `BinExpression` to `BinString`**
355 **Page 4, line 193:   Added signs to `BinDateTimeToken`**
356

357 ## 4.2   2002/12/03 - version G by FIPA Architecture Board

358 Entire document: Promoted to Standard status
359