FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA Agent Message Transport Envelope Representation in Bit-Efficient Encoding Specification

Document title	FIPA AMT Envelope Representation in Bit-Efficient Encoding Specification		
Document number	XC00088B	**Document source**	FIPA Agent Management
Document status	Experimental	**Date of this status**	2001/08/10
Supersedes	None		
Contact	fab@fipa.org		
Change history			
2001/08/10	Approved for Experimental; Line numbering added		

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.

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 Procedures for Technical Work. A complete overview of the FIPA specifications and their current status may be found in the FIPA List of Specifications. A list of terms and abbreviations used in the FIPA specifications may be found in the FIPA Glossary.

FIPA is a non-profit association registered in Geneva, Switzerland. As of January 2000, the 56 members of FIPA represented 17 countries worldwide. Further information about FIPA as an organization, membership information, FIPA specifications and upcoming meetings may be found at http://www.fipa.org/.

Contents

1 Scope 1
2 Bit-Efficient Envelope Representation 2
2.1 Component Name 2
2.2 ACC Processing of Bit-Efficient Envelope 2
2.3 Concrete Message Envelope Syntax 3
2.4 Notes on the Grammar Rules 5
3 Examples 7
4 References 11

1 Scope

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

Syntactic representation of a message envelope in bit-efficient form.

Informative examples of the bit-efficient envelope syntax are given in *Section 3, Examples.*

2 Bit-Efficient Envelope Representation

This section gives the concrete syntax for the message envelope specification that must be used to transport messages over a Message Transport Protocol (MTP - see [FIPA00067]). This concrete syntax is designed to complement [FIPA00069].

The message envelope transport syntax is expressed in standard EBNF format (see *Table 1*).

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`

Table 1: EBNF Rules

N.B. White space is not allowed between tokens.

2.1 Component Name

The name assigned to this component is:

```
fipa.mts.env.rep.bitefficient.std
```

2.2 ACC Processing of Bit-Efficient Envelope

According to [FIPA00067], a FIPA compliant ACC is not allowed to modify any element of the envelope that it receives. It is however allowed to update a value in any of the envelope's slots by adding a new `ExtEnvelope` element at the beginning of the `messageEnvelopes` sequence. This new element is required to have only those slot values that the ACC wishes to add or update plus a new `ReceivedObject` element[1].

The following pseudo code algorithm may be used to obtain the latest values for each of the envelope's slots.

```
EnvelopeWithAllSlots := new empty Envelope
while (not all envelopes processed) {
      tempEnvelope = getNextEnvelope;
      foreach slot in an envelope {
            if ((this slot has no value in EnvelopeWithAllSlots)
                  AND (this slot has a value in tempEnvelope))
            then copy the value of this slot to EnvelopeWithAllSlots
      }
}
```

`EnvelopeWithAllSlots` now contains the latest values for all the slots set in the envelope.

[1] The new `ReceivedObject` is forced, syntactically, to be in all envelopes of the `messageEnvelopes` sequence except the first one.

2.3 Concrete Message Envelope Syntax

```
MessageEnvelope         = (ExtEnvelope)* BaseEnvelope Payload.

BaseEnvelope            = BaseEnvelopeHeader (Slot)* EndOfEnvelope.

ExtEnvelope             = ExtEnvelopeHeader (Slot)* EndOfEnvelope.

BaseEnvelopeHeader      = BaseMsgId EnvLen ACLRepresentation Date.

ExtEnvelopeHeader       = ExtMsgId EnvLen ReceivedObject.

EnvLen                  = Len16
                        | JumboEnvelope.        /* See comment 1 (Section 2.4) */

JumboEnvelope           = EmptyLen16 Len32.

BaseMsgId               = 0xFE.

ExtMsgId                = 0xFD.

EndOfEnvelope           = EndOfCollection.

Payload                 = /* See comment 2 (Section 2.4) */

Slot                    = PredefinedSlot
                        | UserDefinedSlot.      /* See comment 5 (Section 2.4) */

PredefinedSlot          = 0x02 AgentIdentifierSequence  /* to                 */
                        | 0x03 AgentIdentifier          /* from               */
                        | 0x04 ACLRepresentation        /* acl-representation */
                        | 0x05 Comments                 /* comments           */
                        | 0x06 PayloadLength            /* payload-length     */
                        | 0x07 PayloadEncoding          /* payload-encoding   */
                        | 0x08 Encrypted                /* encrypted          */
                        | 0x09 IntendedReceiver         /* intended-receiver  */
                        | 0x0a ReceivedObject           /* received           */
                        | 0x0b TransportBehaviour.      /* transport-behaviour */

ACLRepresentation       = UserDefinedACLRepresentation
                        | 0x10          /* fipa.acl.rep.bitefficient.std [FIPA00069]*/
                        | 0x11          /* fipa.acl.rep.string.std [FIPA00070] */
                        | 0x12.         /* fipa.acl.rep.xml.std [FIPA00071] */

Date                    = BinDateTimeToken.

Comments                = NullTerminatedString.

PayloadLength           = BinNumber.

PayloadEncoding         = NullTerminatedString.

Encrypted               = StringSequence.

IntendedReceiver        = AgentIdentifierSequence.

TransportBehaviour      = Any.

UserDefinedACLRepresentation
                        = 0x00 NullTerminatedString.

ReceivedObject          = By
```

```
                            Date
                            [From]
                            [Id]
                            [Via]
                            EndOfCollection.

By                        = URL.

From                      = 0x02 URL.

Id                        = 0x03 NullTerminatedString.

Via                       = 0x04 NullTerminatedString.

BinNumber                 = Digits.               /* See comment 4 (Section 2.4) */

Digits                    = CodedNumber+.

NullTerminatedString      = String 0x00.

UserDefinedSlot           = 0x00 Keyword NullTerminatedString.

KeyWord                   = NullTerminatedString.

Any                       = 0x14 NullTerminatedString
                          | ByteLenEncoded.

ByteLenEncoded            = 0x16 Len8 ByteSequence
                          | 0x17 Len16 ByteSequence
                          | 0x19 Len32 ByteSequence.

ByteSequence              = Byte*.

AgentIdentifierSequence = (AgentIdentifier)* EndOfCollection.

AgentIdentifier           = 0x02 AgentName
                            [Addresses]
                            [Resolvers]
                            (UserDefinedParameter)*
                            EndOfCollection.

AgentName                 = NullTerminatedString.

Addresses                 = 0x02 UrlSequence.

Resolvers                 = 0x03 AgentIdentifierSequence.

UserDefinedParameter      = 0x04 NullTerminatedString Any.

UrlSequence               = (URL)* EndOfCollection.

URL                       = NullTerminatedString.

StringSequence            = (NullTerminatedString)* EndOfCollection.

BinDateTimeToken          = 0x20 BinDate
                          | 0x21 BinDate TypeDesignator.

BinDate                   = Year Month Day Hour Minute Second Millisecond.
                                                  /* See comment 3 (Section 2.4) */
EndOfCollection           = 0x01.

EmptyLen16                = 0x00 0x00.
```

```
Len8                    = Byte.          /* See comment 6 (Section 2.4) */

Len16                   = Short.         /* See comment 6 (Section 2.4) */

Len32                   = Long.          /* See comment 6 (Section 2.4) */

Year                    = Byte Byte.

Month                   = Byte.

Day                     = Byte.

Hour                    = Byte.

Minute                  = Byte.

Second                  = Byte.

Millisecond             = Byte Byte.

String                  = /* As in [FIPA00070] */

CodedNumber             = /* See comment 4 (Section 2.4) */

TypeDesignator          = /* As in [FIPA00070] */
```

2.4 Notes on the Grammar Rules

1. Normally, the length of an envelope does not exceed 65536 bytes (2^16). Therefore, only two bytes are reserved for envelope length (len16). However, the syntax also allows envelopes with greater lengths. In this case, the sender sets the reserved envelope length slot (two bytes) to length zero, and the following four bytes are used to represent the real length (maximum envelope length is therefore 2^32 bytes).

   The length of the envelope comprises all the parts of the envelope, including the message identifier and the length slot itself. The length of the envelope is expressed in the network byte order.

2. The payload (ACL message) starts at the first byte after the `BaseEnvelope`. White space is allowed between the envelope and the ACL message only if the syntax of ACL allows this. For instance, `fipa.acl.rep.string.std` allows white space, but `fipa.acl.rep.bitefficient.std` does not.

3. Dates are coded as numbers, that is, four bits are reserved for each ASCII number (see comment 4 below). Information as to whether the type designator is present or not is coded into an identifier byte. These slots always have static length (two bytes for year and milliseconds, one byte for other components).

4. 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 2* 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 an odd number of characters, the last four bits of the coded number are set to zero (the `Padding` token), otherwise an additional `0x00` byte is added to the end of the coded number. If the number to be coded is either an 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).

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 2: Binary Representation of Number Tokens

5. All envelope parameters defined in [FIPA00067] have a predefined code. If an envelope contains a user-defined parameter, an extension mechanism is used (byte `0x00`). The names of the user-defined envelope parameters should have the prefix "`X-CompanyName-`".

6. `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).

3 Examples

1. Here is a simple example of an envelope encoded using XML representation:

```
<?xml version="1.0"?>
<envelope>
  <params index="1">
    <to>
      <agent-identifier>
        <name>receiver@foo.com</name>
        <addresses>
          <url>http://foo.com/acc</url>
        </addresses>
      </agent-identifier>
    </to>
    <from>
      <agent-identifier>
        <name>sender@bar.com</name>
        <addresses>
          <url>http://bar.com/acc</url>
        </addresses>
      </agent-identifier>
    </from>

    <acl-representation>fipa.acl.rep.xml.std</acl-representation>

    <date>20000508T042651481</date>

    <encrypted>no encryption</encrypted>

    <received>
      <received-by value="http://foo.com/acc" />
      <received-date value="20000508T042651481" />
      <received-id value="123456789" />
    </received>
  </params>
</envelope>
```

Using the bit-efficient representation, the envelope becomes:

```
0xfe 0x00 0x97 0x12 0x20 0x31 0x11 0x06 0x19 0x15 0x37 0x62 0x59 0x20 0x02 0x03 0x02
‘r’  ‘e’  ‘c’  ‘e’  ‘i’  ‘v’  ‘e’  ‘r’  ‘@’  ‘f’  ‘o’  ‘o’  ‘.’  ‘c’  ‘o’  ‘m’  0x00
0x02 ‘h’  ‘t’  ‘t’  ‘p’  ‘:’  ‘/’  ‘/’  ‘f’  ‘o’  ‘o’  ‘.’  ‘c’  ‘o’  ‘m’  ‘/’  ‘a’
‘c’  ‘c’  0x00 0x01 0x01 0x02 ‘s’  ‘e’  ‘n’  ‘d’  ‘e’  ‘r’  ‘@’  ‘b’  ‘a’  ‘r’  ‘.’
‘c’  ‘o’  ‘m’  0x00 0x02 ‘h’  ‘t’  ‘t’  ‘p’  ‘:’  ‘/’  ‘/’  ‘b’  ‘a’  ‘r’  ‘.’  ‘c’
‘o’  ‘m’  ‘/’  ‘a’  ‘c’  ‘c’  0x00 0x01 0x01 0x08 ‘n’  ‘o’  ‘ ’  ‘e’  ‘n’  ‘c’  ‘r’
‘y’  ‘p’  ‘t’  ‘i’  ‘o’  ‘n’  0x00 0x0a ‘h’  ‘t’  ‘t’  ‘p’  ‘:’  ‘/’  ‘/’  ‘b’  ‘a’
‘r’  ‘.’  ‘c’  ‘o’  ‘m’  ‘/’  ‘a’  ‘c’  ‘c’  0x00 0x20 0x31 0x11 0x06 0x19 0x15 0x37
0x62 0x59 0x20 0x03 ‘1’  ‘2’  ‘3’  ‘4’  ‘5’  ‘6’  ‘7’  ‘8’  ‘9’  0x00 0x01
```

The length of the original message is about 620 bytes and the encoded result is 151 bytes giving a compression ratio of about 4:1.

2. Here is an example that covers all aspects of an envelope.

```
<?xml version="1.0"?>
<envelope>
  <params index="1">
  <to>
    <agent-identifier>
      <name>receiver@foo.com</name>
      <addresses>
        <url>http://foo.com/acc</url>
      </addresses>
      <resolvers>
        <agent-identifier>
          <name>resolver@bar.com</name>
          <addresses>
            <url>http://bar.com/acc1</url>
            <url>http://bar.com/acc2</url>
            <url>http://bar.com/acc3</url>
          </addresses>
        </agent-identifier>
      </resolvers>
    </agent-identifier>
  </to>

  <from>
    <agent-identifier>
      <name>sender@bar.com</name>
      <addresses>
        <url>http://bar.com/acc</url>
      </addresses>
      <resolvers>
        <agent-identifier>
          <name>resolver@foobar.com</name>
          <addresses>
            <url>http://foobar.com/acc1</url>
            <url>http://foobar.com/acc2</url>
            <url>http://foobar.com/acc3</url>
          </addresses>
        </agent-identifier>
      </resolvers>
    </agent-identifier>
  </from>

  <comments>No comments!</comments>

  <acl-representation>fipa.acl.rep.xml.std</acl-representation>

  <payload-encoding>US-ASCII</payload-encoding>

  <date>20000508T042651481</date>

  <encrypted>no encryption</encrypted>

  <intended-receiver>
    <agent-identifier>
      <name>intendedreceiver@foobar.com</name>
      <addresses>
        <url>http://foobar.com/acc1</url>
        <url>http://foobar.com/acc2</url>
        <url>http://foobar.com/acc3</url>
      </addresses>
      <resolvers>
        <agent-identifier>
          <name>resolver@foobar.com</name>
```

```
            <addresses>
              <url>http://foobar.com/acc1</url>
              <url>http://foobar.com/acc2</url>
              <url>http://foobar.com/acc3</url>
            </addresses>
            <resolvers>
              <agent-identifier>
                <name>resolver@foobar.com</name>
                <addresses>
                  <url>http://foobar.com/acc1</url>
                  <url>http://foobar.com/acc2</url>
                  <url>http://foobar.com/acc3</url>
                </addresses>
              </agent-identifier>
            </resolvers>
          </agent-identifier>
        </resolvers>
      </agent-identifier>
    </intended-receiver>


    <received>
      <received-by value="http://foo.com/acc" />
      <received-from value="http://foobar.com/acc" />
      <received-date value="20000508T042651481" />
      <received-id value="123456789" />
      <received-via value="http://bar.com/acc" />
    </received>

    </params>

</envelope>
```

Using the bit-efficient representation, the envelope becomes:

```
0xfe 0x01 0xea 0x12 0x20 0x31 0x11 0x06 0x19 0x15 0x37 0x62 0x59 0x20 0x02 0x02 'r'
'e'  'c'  'e'  'i'  'v'  'e'  'r'  '@'  'f'  'o'  'o'  '.'  'c'  'o'  'm'  0x00 0x02
'h'  't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'  '.'  'c'  'o'  'm'  '/'  'a'  'c'
'c'  0x00 0x01 0x03 0x02 's'  'e'  'n'  'd'  'e'  'r'  '@'  'b'  'a'  'r'  '.'  'c'
'o'  'm'  0x00 0x02 'h'  't'  't'  'p'  ':'  '/'  '/'  'b'  'a'  'r'  '.'  'c'  'o'
'm'  '/'  'a'  'c'  'c'  0x00 0x01 0x07 'U'  'S'  '-'  'A'  'S'  'C'  'I'  'I'  0x00
0x08 'n'  'o'  ' '  'e'  'n'  'c'  'r'  'y'  'p'  't'  'i'  'o'  'n'  0x00 0x01 0x09
0x02 'i'  'n'  't'  'e'  'n'  'd'  'e'  'd'  'r'  'e'  'c'  'e'  'i'  'v'  'e'  'r'
'@'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  0x00 0x02 'h'  't'  't'  'p'
':'  '/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'
'1'  0x00 'h'  't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'
'o'  'm'  '/'  'a'  'c'  'c'  '2'  0x00 'h'  't'  't'  'p'  ':'  '/'  '/'  'f'  'o'
'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'  '3'  0x00 0x01 0x03 0x02
'r'  'e'  's'  'o'  'l'  'v'  'e'  'r'  '@'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'
'o'  'm'  0x00 0x02 'h'  't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'
'.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'  '1'  0x00 'h'  't'  't'  'p'  ':'  '/'  '/'
'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'  '2'  0x00 'h'
't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'
'a'  'c'  'c'  '3'  0x00 0x01 0x03 0x02 'r'  'e'  's'  'o'  'l'  'v'  'e'  'r'  '@'
'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm' 0x00 0x02  'h'  't'  't'  'p'  ':'
'/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'  '1'
0x00 'h'  't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'
'm'  '/'  'a'  'c'  'c'  '2'  0x00 'h'  't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'
'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'  '3'  0x00 0x01 0x01 0x0a 'h'
't'  't'  'p'  ':'  '/'  '/'  'f'  'o'  'o'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'
0x00 0x20 0x31 0x11 0x06 0x19 0x15 0x37 0x62 0x59 0x20 0x02 'h'  't'  't'  'p'  ':'
'/'  '/'  'f'  'o'  'o'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c'  0x00
0x03 '1'  '2'  '3'  '4'  '5'  '6'  '7'  '8'  '9'  0x00 0x01 0x01 0x04 'h'  't'  't'
'p'  ':'  '/'  '/'  'b'  'a'  'r'  '.'  'c'  'o'  'm'  '/'  'a'  'c'  'c' 0x00  0x01
```

461
462 The length of the original message is about 2400 bytes and the encoded result is 490 bytes giving a compression ratio
463 of about 5:1.

464

4 References

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

[FIPA00069] FIPA ACL Message Representation in Bit-Efficient Encoding Specification. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00069/

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

[FIPA00071] FIPA ACL Message Representation in XML Specification. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00071/