FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA ACL Message Representation in String Specification

Document title	FIPA ACL Message Representation in String Specification		
Document number	XC00070H	**Document source**	FIPA Agent Management
Document status	Experimental	**Date of this status**	2002/10/18~~05/10~~
Supersedes	FIPA00024		
Contact	fab@fipa.org		
Change history	See *Informative Annex A — ChangeLog*		

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.

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]~~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~~on the FIPA Web site.

FIPA is a non-profit association registered in Geneva, Switzerland. As of June~~anuary~~ 2002~~0~~, the 56 members of FIPA represented ~~17~~ 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 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 ACL in string form.

2 String ACL Representation

This section defines the message transport syntax for strings which 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".`

Table 1: EBNF Rules

2.1 Component Name

The name assigned to this component is:

```
fipa.acl.rep.string.std
```

2.2 Syntax

```
ACLCommunicativeAct     = Message.

Message                 = "(" MessageType
                              MessageSlot* ")".

MessageType             = See [FIPA00037]

MessageSlot             = ":sender" AgentIdentifier
                        | ":receiver" AgentIdentifierSet
                        | ":content" String
                        | ":reply-with" Expression
                        | ":reply-by" DateTime
                        | ":in-reply-to" Expression
                        | ":reply-to" AgentIdentifierSet
                        | ":language" Expression
                        | ":encoding" Expression
                        | ":ontology" Expression
                        | ":protocol" Word
                        | ":conversation-id" Expression
                        | UserDefinedSlot Expression.

UserDefinedSlot         = Word[1].

Expression              = Word
                        | String
                        | Number
                        | DateTime
                        | "(" Expression* ")".

AgentIdentifier         = "(" "agent-identifier"
                              ":name" word
```

[1] User-defined parameters must start with ":X-".

```
                            [ ":addresses" URLSequence ]
                            [ ":resolvers" AgentIdentifierSequence ]
                            ( UserDefinedSlot Expression )* ")".


AgentIdentifierSequence = "(" "sequence" AgentIdentifier* ")".

AgentIdentifierSet      = "(" "set" AgentIdentifier* ")".

URLSequence             = "(" "sequence" URL* ")".

DateTime                = DateTimeToken.

URL                     = See [RFC2396]
```

2.3 Lexical Rules

Some slightly different rules apply for the generation of lexical tokens. Lexical tokens use the same notation as above, with the exceptions noted in Table 2.

Lexical rule component	**Example**
Square brackets enclose a character set	`[ "a", "b", "c" ]`
Dash in a character set denotes a range	`[ "a″ – "z" ]`
Tilde denotes the complement of a character set if it is the first character	`[ ~ "(", ")" ]`
Post-fix question-mark operator denotes that the preceding lexical expression is optional (may appear zero or one times)	`[ "0" – "9" ] ? [ "0" – "9" ]`

Table 2: Lexical Rules

All white space, tabs, carriage returns and line feeds between tokens should be skipped by the lexical analyser.

```
Word                    = [~ "\0x00" – "\0x20", "(", ")", "#", "0" – "9", "-", "@"]
                          [~ "\0x00" – "\0x20", "(", ")"]*.

String                  = StringLiteral | ByteLengthEncodedString.

StringLiteral           = "\"" ([ ~ "\"" ] | "\\\"")* "\"".

ByteLengthEncodedString = "#" Digit+ "\"" <byte sequence>.

Number                  = Integer | Float.

URL                     = See [RFC2396]

DateTimeToken           =  Sign~~"+"~~ ?
                           Year Month Day "T"
                           Hour Minute Second MilliSecond
                           ( TypeDesignator ? ).

Year                    = Digit Digit Digit Digit.

Month                   = Digit Digit.

Day                     = Digit Digit.

Hour                    = Digit Digit.

Minute                  = Digit Digit.

Second                  = Digit Digit.

MilliSecond             = Digit Digit Digit.
```

```
TypeDesignator         = AlphaCharacter.

AlphaCharacter         = [ "a" – "z" ] | [ "A" – "Z" ].

Digit                  = [ "0" – "9" ].

Sign                   = [ "+" , "-" ] .

Integer                = Sign? Digit+.

Dot                    = [ "." ].

Float                  = Sign? FloatMantissa FloatExponent?
                       | Sign? Digit+ FloatExponent

FloatMantissa          = Digit+ Dot Digit*
                       | Digit* Dot Digit+

FloatExponent          = Exponent Sign? Digit+

Exponent               = [ "e", "E" ]
```

2.4 Representation of Time

Time tokens are based on [ISO8601], with extension for relative time and millisecond durations. Time expressions may be absolute, or relative. Relative times are distinguished by the sign character "+" or "-" appearing as the first character in the token. ~~Time tokens are based on [ISO8601], with extension for millisecond durations.~~ If no type designator is given, the local time zone is then used. The type designator for UTC is the character Z; UTC is preferred to prevent time zone ambiguities. Note that years must be encoded in four digits. As an example, 8:30 am on 15th April, 1996 local time would be encoded as:

```
19960415T083000000
```

The same time in UTC would be:

```
19960415T083000000Z
```

while one hour, 15 minutes and 35 milliseconds from now would be:

+00000000T011500035

2.5 Notes on the Grammar Rules

1. The standard definitions for integers and floating point are assumed.

2. All keywords are case-insensitive.

3. A length encoded string is a context sensitive lexical token. Its meaning is as follows: the message envelope of the token is everything from the leading # to the separator " inclusive. Between the markers of the message envelope is a decimal number with at least one digit. This digit then determines that *exactly* that number of 8-bit bytes are to be consumed as part of the token, without restriction. It is a lexical error for less than that number of bytes to be available.

4. Note that not all implementations of the ACC (see [FIPA00067]) will support the transparent transmission of 8-bit characters. It is the responsibility of the agent to ensure, by reference to internal API of the ACC, that a given channel is able to faithfully transmit the chosen message encoding.

5. A well-formed message will obey the grammar, and in addition, will have at most one of each of the slots. It is an error to attempt to send a message which is not well formed. Further rules on well-formed messages may be stated or implied the operational definitions of the values of slots as these are further developed.

6. Strings encoded in accordance with [ISO2022] may contain characters which are otherwise not permitted in the definition of `Word`. These characters are ESC (`0x1B`), SO (`0x0E`) and SI (`0x0F`). This is due to the complexity that would result from including the full [ISO2022] grammar in the above EBNF description. Hence, despite the basic description above, a word may contain any well-formed [ISO2022] encoded character, other (representations of) parentheses, spaces, or the `#` character. Note that parentheses may legitimately occur as *part* of a well formed escape sequence; the preceding restriction on characters in a word refers only to the encoded characters, not the form of the encoding.

7. The format for time tokens is defined in section *2.4, Representation of Time*.

8. The format for an AID is defined in [FIPA00023].

3 References

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

[FIPA00037] FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00037/`

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

[FIPA00075] FIPA Agent Message Transport Protocol for IIOP Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00075/`

[ISO2022] Information Technology, Character Code Structure and Extension Techniques. International Standards Organisation, 1994. `http://www.iso.ch/cate/d22747.html`

[ISO8601] Date Elements and Interchange Formats, Information Interchange-Representation of Dates and Times. International Standards Organisation, 1998. `http://www.iso.ch/cate/d15903.html`

[RFC2396] Uniform Resource Identifiers: Generic Syntax. Request for Comments, 1998. `http://www.ietf.org/rfc/rfc2396.txt`

4 Informative Annex A — ChangeLog

4.1 2002/05/10 - version H by FIPA Architecture Board

Page 3~~x~~, line 138~~y~~: **Fixed the definition of relative time**~~<blah>~~

Page 4, line 180-194 : Added description of definition of relative time.