# *:96 Internet application layer protocols and standards

## Compendium 1: Allowed during the exam

**Last revision: 28 Aug 2001**

## ASN.1



*There is no page 11*

## ABNF



## DNS



*There is no page 34*

## E-mail



## LDAP



*The documents are not ordered in a suitable order for reading them, see compendium 0 page 14-17*

*Pages 2-12 are offprints from "Abstract Syntax Notation One (ASN.1): The Turorial and Reference" by Douglas Steedman 1990*

# Appendix A:
# ASN.1 syntax (basic items)

## A.1 ASN.1 character set

The following characters can be found in ASN.1 specifications:

**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**
**a b c d e f g h i j k l m n o p q r s t u v w x y z**
**0 1 2 3 4 5 6 7 8 9 ( ) { } [ ] < > : ; = , . - ' "**

Within two of the basic syntactic items which are described below, comment and cstring, additional characters can appear.

## A.2 Basic syntactic items

The table below describes the basic syntactic items of ASN.1. Each item is shown as it appears in the syntax diagrams of Appendices B and C. A "string" is a sequence of zero or more characters, while a "name" is a string of at least one character, drawn from **A-Z**, **a-z**, **0-9** and hyphen ("-"). In a name a hyphen cannot appear as the last character or adjacent to another hyphen.

| Syntactic item | Contains |
|---|---|
| bstring | string of 0s and 1s, bracketed by "'" and "'**B**" |
| comment | a pair of hyphens ("--") followed by a string of arbitrary characters and terminated by the end of a line or a pair of hyphens, whichever is first |
| cstring | string of characters from some ASN.1 character set, with "" represented by """" |
| hstring | string of hexadecimal digits (**0-9**, **A-F**) bracketed by "'" and "'**H**" |
| identifier | name starting with lower-case letter |
| modulereference | name starting with upper-case letter |
| number | one or more decimal digits; no leading zeros |
| typereference | name starting with upper-case letter |
| valuereference | name starting with lower-case letter |





A number of additional basic items are added for macros:

| | |
|---|---|
| astring | string of characters from the ASN.1 character set |
| localtypereference | name starting with upper-case letter |
| localvaluereference | name starting with *upper*-case† letter |
| macroreference | name in which all letters are upper-case |
| productionreference | name starting with upper-case letter |

† This is presently a discrepancy between the ISO and CCITT versions, the latter aligning this with value reference. The discrepancy seems to have originated in a misinterpretation of X.409, ASN.1's predecessor.

## A.3 Non-alphabetic items

The brackets, separators, terminators and other special symbols of ASN.1 are as follows:

| | |
|---|---|
| { | starts a list |
| } | ends a list |
| [ | starts a tag |
| ] | ends a tag |
| ( | starts a subtype expression<br>starts a named number, bit or object identifier component |
| ) | ends a subtype expression<br>ends a named number, bit or object identifier component |
| < | "alternative of" in selection type<br>starts embedded definitions (macros) |
| > | ends embedded definitions (macros) |
| {..., | starts a list in partial specification of inner subtyping |
| " | starts or ends character string value |
| , | separates list items |
| . | separates module reference from type or value reference |
| ; | terminates import and export statements |
| .. | range separator |

| | |
|---|---|
| \| | alternative subtype value set<br>alternative syntax (macros) |
| - | (hyphen) minus sign for negative numbers |
| ::= | "defined as" in assignments (also for introducing module body and macro alternative list) |
| 0 | zero (exact real value) |
| 2 | indicates binary base for real value |
| 10 | indicates decimal base for real value |

## A.4 Keyword items

The following character sequences constitute the keywords of ASN.1. Unfortunately, as shown by the annotations, it is somewhat unclear which of these are reserved, that is, not available for for use in forming identifiers and references. Accordingly, it is recommended that all of the keywords should be avoided when forming such names.

| | | | |
|---|---|---|---|
| ABSENT† | ANY | APPLICATION† | BEGIN |
| BIT | BOOLEAN | BY† | CHOICE |
| COMPONENT† | COMPONENTS | DEFAULT | DEFINED† |
| DEFINITIONS† | empty** | END | ENUMERATED† |
| EXPLICIT† | EXPORTS† | EXTERNAL | FALSE |
| FROM† | IA5String° | identifier** | IDENTIFIER |
| GeneralizedTime° | GeneralString° | GraphicString° | IMPLICIT |
| IMPORTS† | INCLUDES† | INTEGER | ISO646String° |
| MACRO* | MAX† | MIN† | MINUS-INFINITY† |
| NOTATION* | NULL | NumericString° | number** |
| OBJECT | ObjectDescriptor° | OCTET | OF |
| OPTIONAL | PLUS-INFINITY† | PRESENT† | PrintableString° |
| PRIVATE† | REAL† | SEQUENCE | SET |
| SIZE† | string** | STRING | TAGS† |
| TeletexString° | TRUE | type* | TYPE* |
| T61String° | value* | VALUE* | VideotexString° |
| VisibleString° | UNIVERSAL† | UTCTime° | WITH† |

† not reserved by ISO 8824, but reserved by CCITT X.208. This is not a deliberate difference but arose from misalignment of publication schedules.

* reserved inside macro definitions.

** can appear inside macro definitions, but not reserved.

° character set type or useful type; it is not clear whether or not these are reserved (anomalously, **EXTERNAL**, a useful type, is reserved).



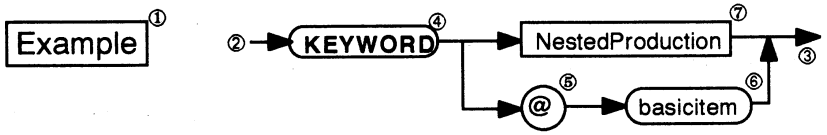
# Appendix B: ASN.1 syntax

The complete syntax of ASN.1, excluding the macro capability, is shown in the syntax diagrams which follow. The various diagrams, each named for the part of the syntax which they illustrate, are arranged in alphabetical order for easy reference.

An ASN.1 module is only valid if it corresponds to some path through the diagrams, starting with Module and following other diagrams whenever their names appear on the path. The path through a diagram must proceed from the start to the end and travel only in the direction of the arrows (never following a line "upstream").

Comments and white-space can freely appear between, but not within, syntactic items.

Each diagram is of the following form:




**KEY**

① the name of the part of the syntax being illustrated.
② the start of the path to be followed (always near the upper left corner).
③ the end of the path (almost always near the upper right corner).
④ a keyword of the syntax(A.4), shown in boldface within a rounded box.
⑤ a non-alphabetic item (A.3), shown within a circle or rounded box.
⑥ a basic item of the syntax (A.2), shown in a rounded box.
⑦ the name of a diagram which is logically nested within this one, shown in a rectangle.

In defining the syntax, these diagrams are not sufficient, because there are context-specific rules, such as that forbidding the multiple use of the same identifier within a set type, or insisting that the members of that set type have distinct tags. Such rules are described following the diagram of the affected construct.

Some rules constrain a Type to be "of integer type" (for example). In determining whether some Type is indeed appropriate in such a situation, it must be reduced to its associated BuiltInType, as described under "Type" below. It can then be said to be "of integer type" if and only if the resulting BuiltInType is IntegerType.

There is no page 130

In this description, tagging is subsumed by the concept of Type. Thus the "tagged types" of the tutorial do not appear as such.

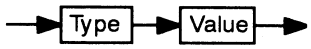
AnyType


*Further rules*:

a) the **DEFINED BY** option can only be present in BuiltinType in Type in NamedType in ElementType in SetType (or SequenceType).

b) the identifier must be that of another component of the SetType (or SequenceType) and that component must be of integer or object identifier type.

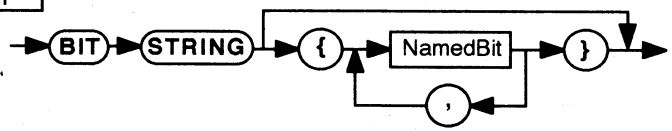
AnyValue




*Further rules*:

a) Value must be a value of Type.

BitStringType

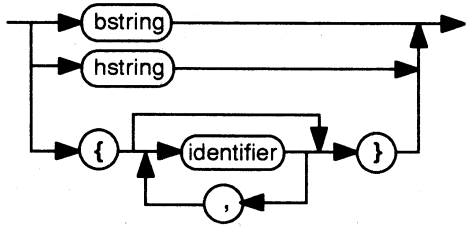



*Further rules*:

a) the significance of trailing zero bits is decided by the user and may be indicated by comment.

b) the identifiers in the various NamedBits must be distinct, as must the bit positions.

BitStringValue




*Further rules*:

a) if trailing zero bits are significant then the notation starting "{" cannot be used, and the hstring notation can only be used if the value contains a multiple of four bits.

b) in the bstring and hstring notations, earlier digits correspond to lower-numbered bits.

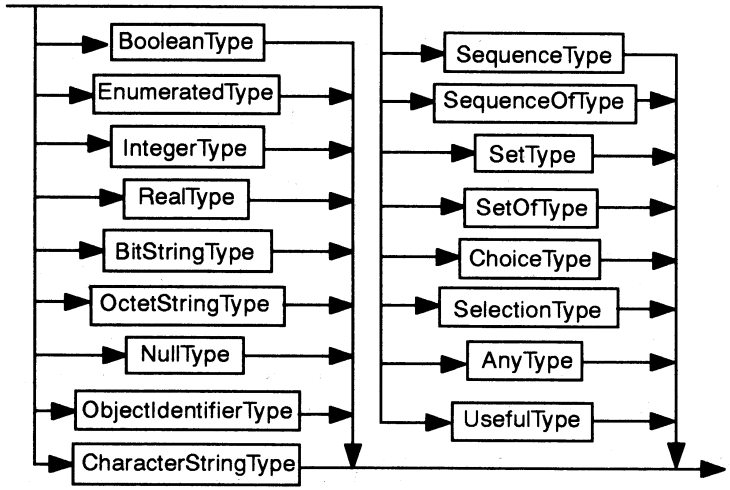
c) in the hstring notation, each hexadecimal digit describes four bits, with the lower-numbered being more significant.



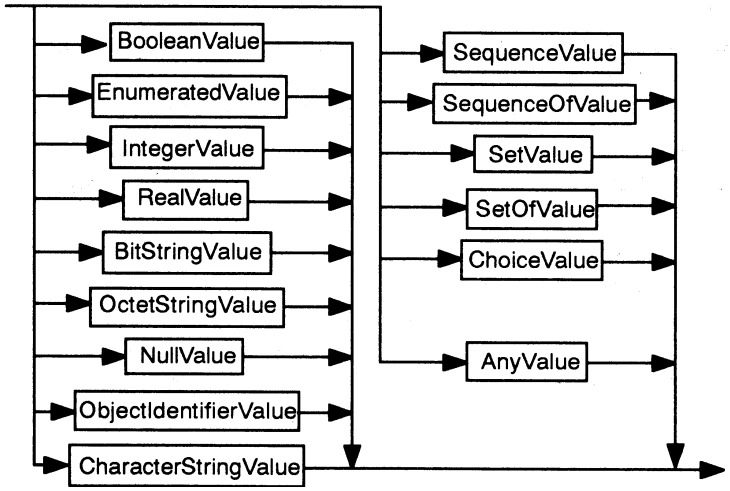
BooleanType


BooleanValue

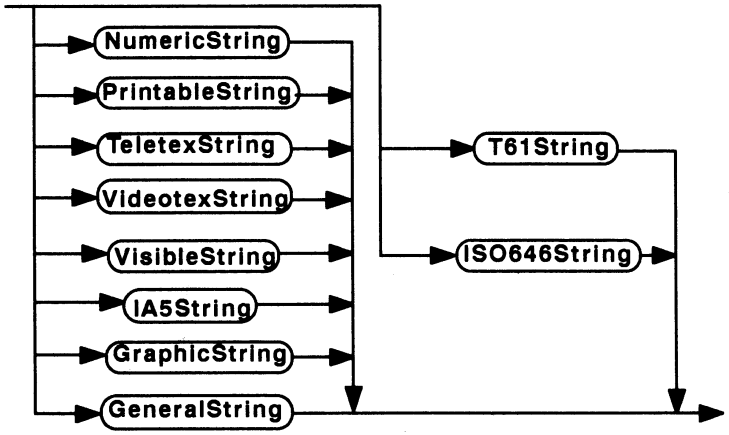

BuiltinType



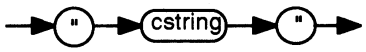

BuiltinValue



CharacterStringType



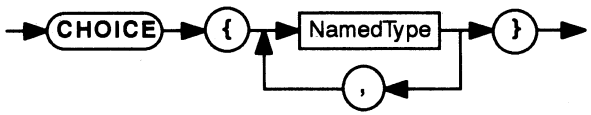

CharacterStringValue




*Further rules*:

a) the characters in the cstring must all belong to the character set implied by the corresponding type.

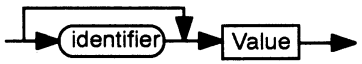
ChoiceType




*Further rules*:

a) the identifiers in the various NamedTypes must be distinct, as must the tags of the types.
b) this type has the tags of all of the NamedTypes.

ChoiceValue

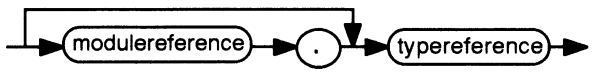



*Further rules*:

a) if an identifier is present, Value must be a value of the identified alternative of the corresponding ChoiceType.
b) if no identifier is present, Value must be a value of an alternative (of the corresponding ChoiceType) with no identifier.



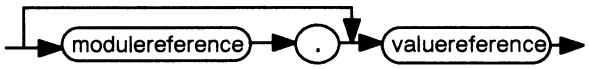
DefinedType




*Further rules*:

a) if no modulereference is present, the typereference must appear in a TypeAssignment, or (if not) as (precisely one) Symbol in Imports, in (the current) Module.
b) if a modulereference is present, then the typereference must appear in a TypeAssignment in (the identified) Module.
c) if Imports is present in (the current) Module, then the modulereference must appear in a ModuleIdentifier therein, and the typereference must be one of the Symbols imported hence.

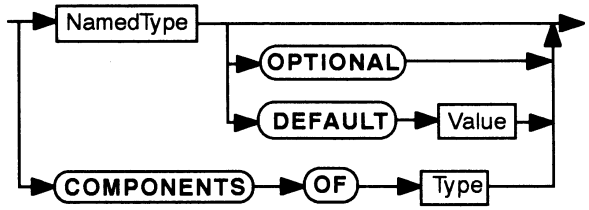
DefinedValue




*Further rules*:

(As for DefinedType, but replacing typereference by valuereference and TypeAssignment by ValueAssignment).

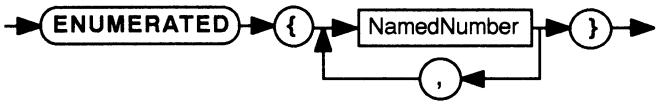
ElementType




*Further rules*:

a) the Value must be a value of the type of the NamedType.
b) the Type must be a set (sequence) type if this appears in a SetType (SequenceType).

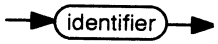
EnumeratedType




*Further rules*:

a) the identifiers in the various NamedNumbers must be distinct, as must the integer values.

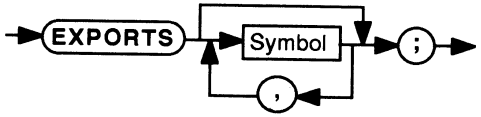
EnumeratedValue




*Further rules*:

a) the identifier must be one of those appearing in the corresponding EnumeratedType.

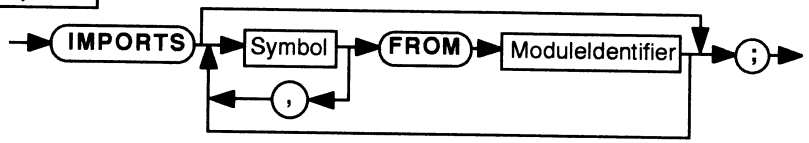
Exports




*Further rules*:

a) each Symbol must appear as the reference in an assignment (type, value, or macro, as appropriate) in the current Module.
b) a particular Symbol cannot appear more than once.

Imports



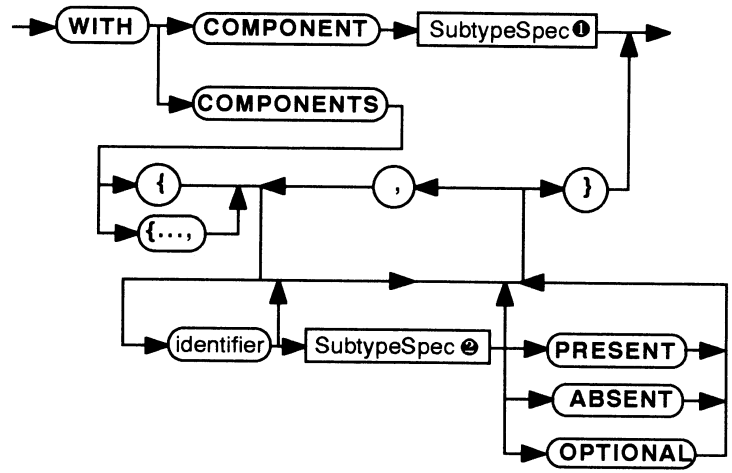

*Further rules*:

a) each Symbol must appear as the reference in an assignment (type, value, or macro, as appropriate) in the Module identified by the closest following ModuleIdentifier.
b) all Symbols imported from the same Module must appear in a contiguous list (without an intervening ModuleIdentifier), and a particular Symbol cannot appear more than once in such a list.
c) each ModuleIdentifier must appear as it does in the identified Module, except that if an ObjectIdentifierValue is present then the modulereference may be changed if it is necessary to allow unambiguous reference (when the same Symbol is imported from two Modules with the same modulereference).





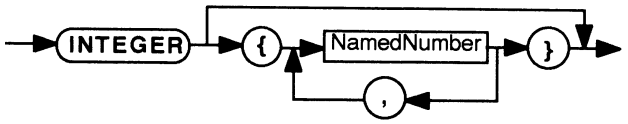
InnerSubtyping




*Further rules*:

a) **WITH COMPONENT** is permitted only if the parent type is a set-of type or sequence-of type, and SubtypeSpec❶ must be a valid subtype specification for the component type.
b) the identifier, if present, must be one of those appearing in the parent type, and identifies the component to which any following constraint applies. If the identifier is omitted, then the constrained component is identified by position.
c) where the parent type is a set type or a sequence type, a constraint (though perhaps empty) must be present for every mandatory component.
d) SubtypeSpec❷ must be a valid subtype specification for the constrained component.
e) where the parent type is a set type or a sequence type, **PRESENT**, **ABSENT**, and **OPTIONAL** are only permitted if the constrained component is optional.
f) where the parent type is a choice type, **OPTIONAL** is not permitted.

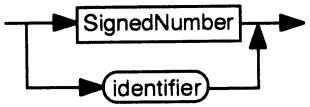
IntegerType




*Further rules*:

a) the identifiers in the various NamedNumbers (if any) must be distinct, as must the integer values.

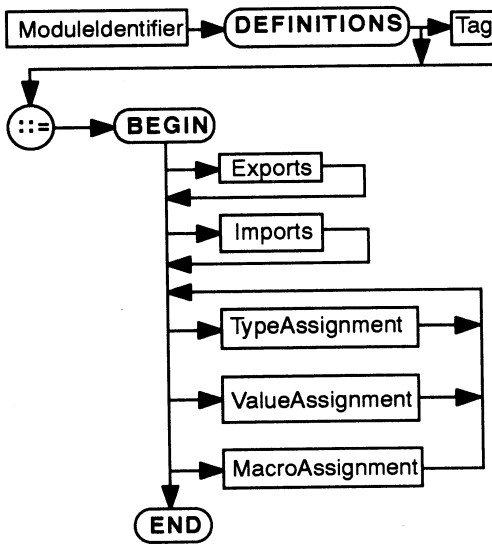
IntegerValue




*Further rules*:

a) the identifier must be one of those appearing in the corresponding IntegerType.

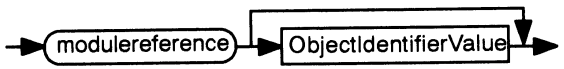
Module



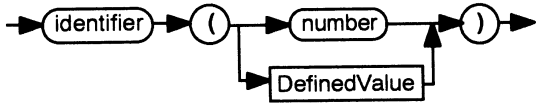

*Further rules:*

a) each Module should be assigned an ObjectIdentifierValue to appear in its ModuleIdentifier.
b) each assignment must have a distinct reference.
c) if no TagStyleDefault is present, explicit tagging is the default.

ModuleIdentifier




NamedBit

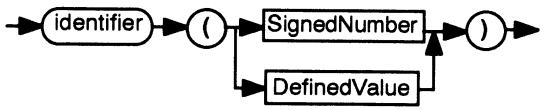



*Further rules*:

a) the DefinedValue must be of integer type, and must be non-negative.
b) the number or DefinedValue identifies a bit position (from zero).



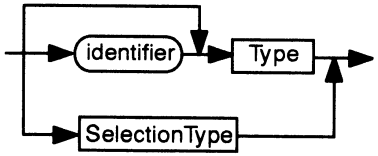
NamedNumber




*Further rules*:

a) the DefinedValue must be of integer type.
b) the SignedNumber or DefinedValue provides an integer value.

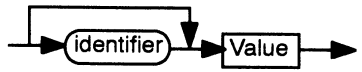
NamedType




*Further rules*:

a) where a SelectionType is present, the identifier is considered to be that which appears therein, and the type is considered to be the appropriate alternative of the choice Type which appears therein.

NamedValue

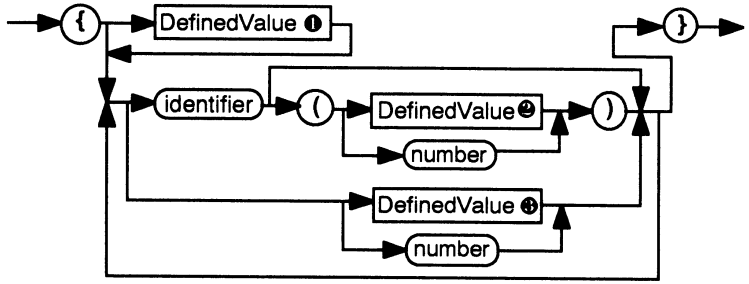



NullType


NullValue


ObjectIdentifierType


ObjectIdentifierValue

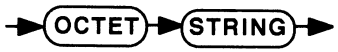



*Further rules*:

a) the DefinedValue❶ (if present) must be of object identifier type.

b) each DefinedValue❷ or DefinedValue❸ must be of integer type, and must be non-negative.
c) an identifer must be followed by "(", unless it is one of the predefined object identifier components shown in Appendix D.

OctetStringType



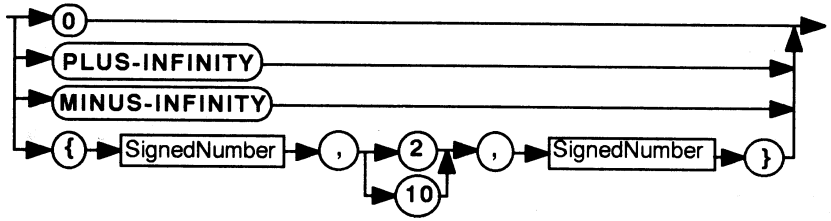

OctetStringValue


*Further rules*:

a) if necessary, trailing zeros are deemed to have been added so that the string denotes an integral number of octets.
b) in the bstring notation, each successive group of eight bits denotes a successive octet of the value.
c) in the hstring notation, each successive pair of hexadecimal digits denotes a successive octet of the value. The first of the pair is more significant.

RealType


RealValue




*Further rules*:

a) the notation containing the single item "0" is used if and only if the value is zero exactly.

SelectionType

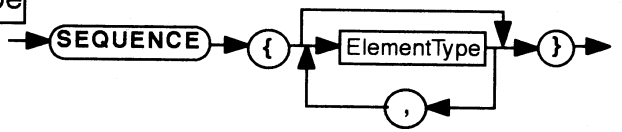

*Further rules*:

a) the Type must be a choice type.





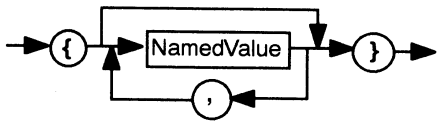
SequenceType




*Further rules*:

a) below, and in the further rules for SequenceValue, each ElementType which is a **"COMPONENTS OF"** is considered to have been replaced, *in situ*, by the elements of the designated sequence type. In the event of nested **"COMPONENTS OF"**, this is applied repeatedly. This must be possible in a finite number of steps.
b) the identifiers in the various ElementTypes must be distinct.
c) each series of consecutive ElementTypes marked **OPTIONAL** or **DEFAULT** must have distinct tags from each other, and from the following ElementType, if there is one.

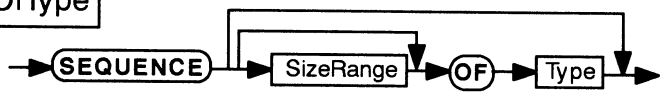
SequenceValue



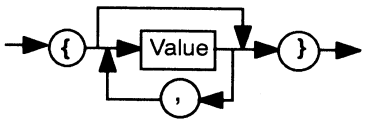

*Further rules*:

a) there must be exactly one NamedValue for each ElementType in the corresponding SequenceType which is not marked **OPTIONAL** or **DEFAULT**, and zero or one for each which *is* so marked.
b) the NamedValues must appear in the order in which the ElementTypes appear in the corresponding SequenceType.
c) the element to which a NamedValue corresponds may be indicated by an identifier, which must appear in the corresponding SequenceType.

SequenceOfType



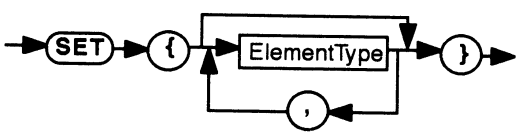

SequenceOfValue




*Further rules*:

a) each Value must be of the Type which appears in the corresponding SequenceOfType.

SetType

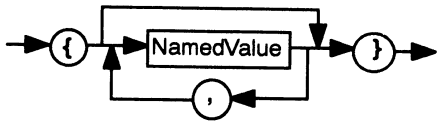



*Further rules*:

a) below, and in the further rules for SetValue, each ElementType which is a "**COMPONENTS OF**" is considered to have been replaced, *in situ*, by the elements of the designated set type. In the event of nested "**COMPONENTS OF**", this is applied repeatedly. This must be possible in a finite number of steps.

b) the identifiers in the various ElementTypes must be distinct, as must the tags of the types.

SetValue

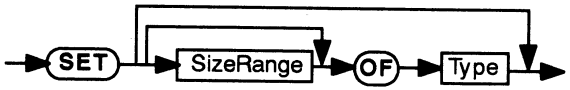



*Further rules*:

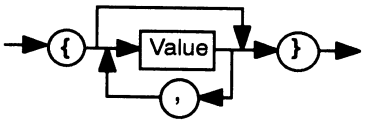
a) there must be exactly one NamedValue for each ElementType in the corresponding SetType which is not marked **OPTIONAL** or **DEFAULT**, and zero or one for each which *is* so marked.

b) the element to which a NamedValue corresponds may be indicated by an identifier, which must appear in the corresponding SetType.

SetOfType



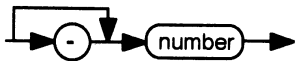

SetOfValue




*Further rules*:

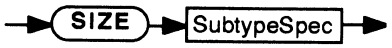
a) each Value must be of the Type which appears in the corresponding SetOfType.

SignedNumber





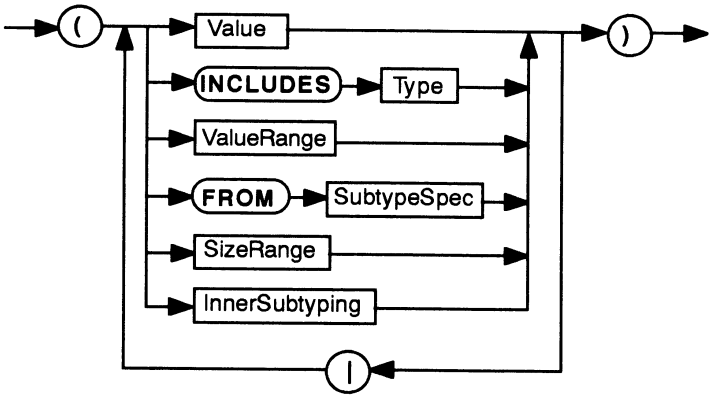

SizeRange




*Further rules*:

a) the SubtypeSpec must be a valid subtype specification for **INTEGER (0..MAX)**.

SubtypeSpec




*Further rules*:

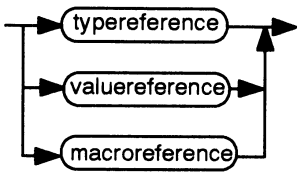
a) the valid subtype specifications for various parent types are shown in Appendix E below. Therein, the possibilities shown above by Value alone, **INCLUDES**, and **FROM** are referred to respectively as single value, contained subtype, and permitted alphabet.

b) Value must be a value of the parent type, as must Value❶ and Value❷ in SizeRange.

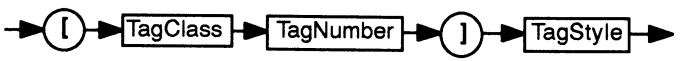
c) Type must be a subtype of the parent type.

d) SubtypeSpec must be a valid subtype specification for the same parent type, but of **SIZE(1)**.

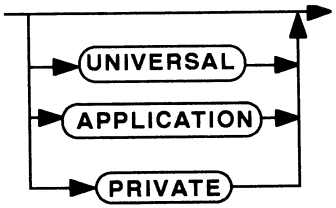
Symbol




Tag



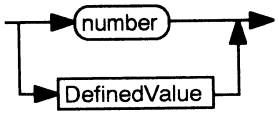
TagClass




*Further rules*:

a) the **UNIVERSAL** option is only available to the designers of ASN.1 itself.

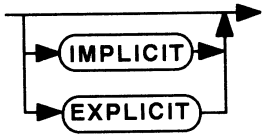
TagNumber




*Further rules*:

a) the DefinedValue must be of integer type, and must be non-negative.

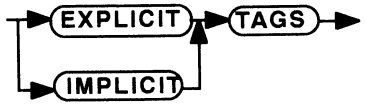
TagStyle



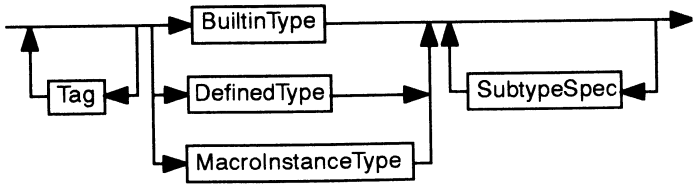

*Further rules*:

a) when neither **EXPLICIT** nor **IMPLICIT** is present, the TagStyle-Default for the current Module is taken.

TagStyleDefault




Type




*Further rules*:

a) the last Tag must not include **IMPLICIT** if the BuiltinType or DefinedType is an (untagged) choice or any type.

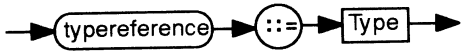


b) the SubtypeSpecs must be valid for the BuiltinType or DefinedType. The valid subtype specifications for various parent types are shown in Appendix E below.

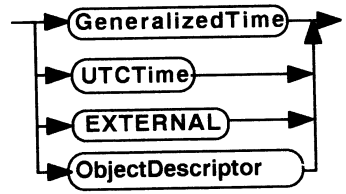
c) if the macro being instantiated by the MacroInstanceType can deliver different types for different MacroInstanceValues, then the rule (a) also applies here, and there must be no SubtypeSpecs.

d) where it is necessary to reduce a Type to its associated BuiltInType, this is done by iteratively discarding Tags and SubtypeSpecs, replacing any DefinedType by the Type by which it was defined, and reducing any MacroInstanceType to its delivered type.

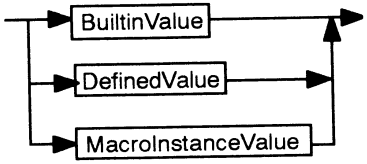
TypeAssignment



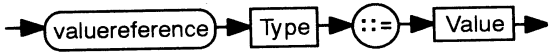

UsefulType




Value



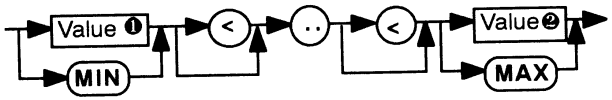

ValueAssignment




*Further rules*:

a) the Value must be a value of Type.

ValueRange




*Further rules*:

a) if both Values are present, then Value❶ must be not greater than Value❷.

Network Working Group
Request for Comments: 2234
Category: Standards Track

D. Crocker, Ed.
Internet Mail Consortium
P. Overell
Demon Internet Ltd.
November 1997

# Augmented BNF for Syntax Specifications: ABNF

## Status of this Memo



## Copyright Notice



## TABLE OF CONTENTS









## 1. INTRODUCTION

Internet technical specifications often need to define a format syntax and are free to employ whatever notation their authors deem useful. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. It balances compactness and simplicity, with reasonable representational power. In the early days of the Arpanet, each specification contained its own definition of ABNF. This included the email specifications, RFC733 and then RFC822 which have come to be the common citations for defining ABNF. The current document separates out that definition, to permit selective reference. Predictably, it also provides some modifications and enhancements.

The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. Appendix A (Core) supplies rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications. It is provided as a convenience and is otherwise separate from the meta language defined in the body of this document, and separate from its formal status.

## 2. RULE DEFINITION

### 2.1 Rule Naming

The name of a rule is simply the name itself; that is, a sequence of characters, beginning with an alphabetic character, and followed by a combination of alphabetics, digits and hyphens (dashes).

NOTE: Rule names are case-insensitive

The names <rulename>, <Rulename>, <RULENAME> and <rUlENamE> all refer to the same rule.

Unlike original BNF, angle brackets ("<", ">") are not required. However, angle brackets may be used around a rule name whenever their presence will facilitate discerning the use of a rule name. This is typically restricted to rule name references in free-form prose, or to distinguish partial rules that combine into a string not separated by white space, such as shown in the discussion about repetition, below.

## 2.2 Rule Form

A rule is defined by the following sequence:

```
name =  elements crlf
```

where <name> is the name of the rule, <elements> is one or more rule names or terminal specifications and <crlf> is the end-of- line indicator, carriage return followed by line feed. The equal sign separates the name from the definition of the rule. The elements form a sequence of one or more rule names and/or value definitions, combined according to the various operators, defined in this document, such as alternative and repetition.

For visual ease, rule definitions are left aligned. When a rule requires multiple lines, the continuation lines are indented. The left alignment and indentation are relative to the first lines of the ABNF rules and need not match the left margin of the document.

## 2.3 Terminal Values

Rules resolve into a string of terminal values, sometimes called characters. In ABNF a character is merely a non-negative integer. In certain contexts a specific mapping (encoding) of values into a character set (such as ASCII) will be specified.

Terminals are specified by one or more numeric characters with the base interpretation of those characters indicated explicitly. The following bases are currently defined:

```
b           =  binary

d           =  decimal

x           =  hexadecimal
```





Hence:

```
CR          =  %d13

CR          =  %x0D
```

respectively specify the decimal and hexadecimal representation of [US-ASCII] for carriage return.

A concatenated string of such values is specified compactly, using a period (".") to indicate separation of characters within that value. Hence:

```
CRLF        =  %d13.10
```

ABNF permits specifying literal text string directly, enclosed in quotation-marks. Hence:

```
command     =  "command string"
```

Literal text strings are interpreted as a concatenated set of printable characters.

```
NOTE:     ABNF strings are case-insensitive and
          the character set for these strings is us-ascii.
```

Hence:

```
rulename = "abc"
```

and:

```
rulename = "aBc"
```

will match "abc", "Abc", "aBc", "abC", "ABc", "aBC", "AbC" and "ABC".

```
          To specify a rule which IS case SENSITIVE,
             specify the characters individually.
```

For example:

```
rulename    =  %d97 %d98 %d99
```

or

```
rulename    =  %d97.98.99
```

will match only the string which comprises only lowercased characters, abc.

## 2.4 External Encodings

External representations of terminal value characters will vary according to constraints in the storage or transmission environment. Hence, the same ABNF-based grammar may have multiple external encodings, such as one for a 7-bit US-ASCII environment, another for a binary octet environment and still a different one when 16-bit Unicode is used. Encoding details are beyond the scope of ABNF, although Appendix A (Core) provides definitions for a 7-bit US-ASCII environment as has been common to much of the Internet.

By separating external encoding from the syntax, it is intended that alternate encoding environments can be used for the same syntax.

# 3. OPERATORS

## 3.1 Concatenation Rule1 Rule2

A rule can define a simple, ordered string of values -- i.e., a concatenation of contiguous characters -- by listing a sequence of rule names. For example:

```
foo         =  %x61          ; a

bar         =  %x62          ; b

mumble      =  foo bar foo
```

So that the rule <mumble> matches the lowercase string "aba".

LINEAR WHITE SPACE: Concatenation is at the core of the ABNF parsing model. A string of contiguous characters (values) is parsed according to the rules defined in ABNF. For Internet specifications, there is some history of permitting linear white space (space and horizontal tab) to be freelyPand implicitlyPinterspersed around major constructs, such as delimiting special characters or atomic strings.

NOTE: This specification for ABNF does not provide for implicit specification of linear white space.

Any grammar which wishes to permit linear white space around delimiters or string segments must specify it explicitly. It is often useful to provide for such white space in "core" rules that are





then used variously among higher-level rules. The "core" rules might be formed into a lexical analyzer or simply be part of the main ruleset.

## 3.2 Alternatives Rule1 / Rule2

Elements separated by forward slash ("/") are alternatives. Therefore,

```
foo / bar
```

will accept <foo> or <bar>.

NOTE: A quoted string containing alphabetic characters is special form for specifying alternative characters and is interpreted as a non-terminal representing the set of combinatorial strings with the contained characters, in the specified order but with any mixture of upper and lower case..

## 3.3 Incremental Alternatives Rule1 =/ Rule2

It is sometimes convenient to specify a list of alternatives in fragments. That is, an initial rule may match one or more alternatives, with later rule definitions adding to the set of alternatives. This is particularly useful for otherwise- independent specifications which derive from the same parent rule set, such as often occurs with parameter lists. ABNF permits this incremental definition through the construct:

```
oldrule     =/ additional-alternatives
```

So that the rule set

```
ruleset     =  alt1 / alt2

ruleset     =/ alt3

ruleset     =/ alt4 / alt5
```

is the same as specifying

```
ruleset     =  alt1 / alt2 / alt3 / alt4 / alt5
```

### 3.4 Value Range Alternatives %c##-##

A range of alternative numeric values can be specified compactly, using dash ("-") to indicate the range of alternative values. Hence:

```
DIGIT       =  %x30-39
```

is equivalent to:

```
DIGIT       =  "0" / "1" / "2" / "3" / "4" / "5" / "6" /

                    "7" / "8" / "9"
```

Concatenated numeric values and numeric value ranges can not be specified in the same string. A numeric value may use the dotted notation for concatenation or it may use the dash notation to specify one value range. Hence, to specify one printable character, between end of line sequences, the specification could be:

```
char-line = %x0D.0A %x20-7E %x0D.0A
```

### 3.5 Sequence Group (Rule1 Rule2)

Elements enclosed in parentheses are treated as a single element, whose contents are STRICTLY ORDERED. Thus,

```
elem (foo / bar) blat
```

which matches (elem foo blat) or (elem bar blat).

```
elem foo / bar blat
```

matches (elem foo) or (bar blat).

NOTE: It is strongly advised to use grouping notation, rather than to rely on proper reading of "bare" alternations, when alternatives consist of multiple rule names or literals.

Hence it is recommended that instead of the above form, the form:

```
(elem foo) / (bar blat)
```

be used. It will avoid misinterpretation by casual readers.

The sequence group notation is also used within free text to set off an element sequence from the prose.





### 3.6 Variable Repetition *Rule

The operator "*" preceding an element indicates repetition. The full form is:

```
<a>*<b>element
```

where <a> and <b> are optional decimal values, indicating at least <a> and at most <b> occurrences of element.

Default values are 0 and infinity so that *<element> allows any number, including zero; 1*<element> requires at least one; 3*3<element> allows exactly 3 and 1*2<element> allows one or two.

### 3.7 Specific Repetition nRule

A rule of the form:

```
<n>element
```

is equivalent to

```
<n>*<n>element
```

That is, exactly <N> occurrences of <element>. Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

### 3.8 Optional Sequence [RULE]

Square brackets enclose an optional element sequence:

```
[foo bar]
```

is equivalent to

```
*1(foo bar).
```

### 3.9 ; Comment

A semi-colon starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

### 3.10 Operator Precedence

The various mechanisms described above have the following precedence, from highest (binding tightest) at the top, to lowest and loosest at the bottom:

```
Strings, Names formation
Comment
Value range
Repetition
Grouping, Optional
Concatenation
Alternative
```

Use of the alternative operator, freely mixed with concatenations can be confusing.

Again, it is recommended that the grouping operator be used to make explicit concatenation groups.

## 4. ABNF DEFINITION OF ABNF

This syntax uses the rules provided in Appendix A (Core).

```
rulelist       =  1*( rule / (*c-wsp c-nl) )

rule           =  rulename defined-as elements c-nl
                       ; continues if next line starts
                       ;  with white space

rulename       =  ALPHA *(ALPHA / DIGIT / "-")

defined-as     =  *c-wsp ("=" / "=/") *c-wsp
                       ; basic rules definition and
                       ;  incremental alternatives

elements       =  alternation *c-wsp

c-wsp          =  WSP / (c-nl WSP)

c-nl           =  comment / CRLF
                       ; comment or newline

comment        =  ";" *(WSP / VCHAR) CRLF

alternation    =  concatenation
                  *(*c-wsp "/" *c-wsp concatenation)
```





```
concatenation  =  repetition *(1*c-wsp repetition)

repetition     =  [repeat] element

repeat         =  1*DIGIT / (*DIGIT "*" *DIGIT)

element        =  rulename / group / option /
                  char-val / num-val / prose-val

group          =  "(" *c-wsp alternation *c-wsp ")"

option         =  "[" *c-wsp alternation *c-wsp "]"

char-val       =  DQUOTE *(%x20-21 / %x23-7E) DQUOTE
                       ; quoted string of SP and VCHAR
                          without DQUOTE

num-val        =  "%" (bin-val / dec-val / hex-val)

bin-val        =  "b" 1*BIT
                  [ 1*("." 1*BIT) / ("-" 1*BIT) ]
                       ; series of concatenated bit values
                       ; or single ONEOF range

dec-val        =  "d" 1*DIGIT
                  [ 1*("." 1*DIGIT) / ("-" 1*DIGIT) ]

hex-val        =  "x" 1*HEXDIG
                  [ 1*("." 1*HEXDIG) / ("-" 1*HEXDIG) ]

prose-val      =  "<" *(%x20-3D / %x3F-7E) ">"
                       ; bracketed string of SP and VCHAR
                          without angles
                       ; prose description, to be used as
                          last resort
```

## 5. SECURITY CONSIDERATIONS

Security is truly believed to be irrelevant to this document.

## 6. APPENDIX A - CORE

This Appendix is provided as a convenient core for specific grammars. The definitions may be used as a core set of rules.

### 6.1 Core Rules

Certain basic rules are in uppercase, such as SP, HTAB, CRLF, DIGIT, ALPHA, etc.

```
ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z

BIT            =  "0" / "1"

CHAR           =  %x01-7F
                       ; any 7-bit US-ASCII character,
                          excluding NUL

CR             =  %x0D
                       ; carriage return

CRLF           =  CR LF
                       ; Internet standard newline

CTL            =  %x00-1F / %x7F
                       ; controls

DIGIT          =  %x30-39
                       ; 0-9

DQUOTE         =  %x22
                       ; " (Double Quote)

HEXDIG         =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

HTAB           =  %x09
                       ; horizontal tab

LF             =  %x0A
                       ; linefeed

LWSP           =  *(WSP / CRLF WSP)
                       ; linear white space (past newline)

OCTET          =  %x00-FF
                       ; 8 bits of data

SP             =  %x20
                       ; space

VCHAR          =  %x21-7E
                       ; visible (printing) characters

WSP            =  SP / HTAB
                       ; white space
```

### 6.2 Common Encoding

Externally, data are represented as "network virtual ASCII", namely 7-bit US-ASCII in an 8-bit field, with the high (8th) bit set to zero. A string of values is in "network byte order" with the higher-valued bytes represented on the left-hand side and being sent over the network first.

## 7. ACKNOWLEDGMENTS

The syntax for ABNF was originally specified in RFC 733. Ken L. Harrenstien, of SRI International, was responsible for re-coding the BNF into an augmented BNF that makes the representation smaller and easier to understand.

This recent project began as a simple effort to cull out the portion of RFC 822 which has been repeatedly cited by non-email specification writers, namely the description of augmented BNF. Rather than simply and blindly converting the existing text into a separate document, the working group chose to give careful consideration to the deficiencies, as well as benefits, of the existing specification and related specifications available over the last 15 years and therefore to pursue enhancement. This turned the project into something rather more ambitious than first intended. Interestingly the result is not massively different from that original, although decisions such as removing the list notation came as a surprise.

The current round of specification was part of the DRUMS working group, with significant contributions from Jerome Abela , Harald Alvestrand, Robert Elz, Roger Fajman, Aviva Garrett, Tom Harsch, Dan Kohn, Bill McQuillan, Keith Moore, Chris Newman , Pete Resnick and Henning Schulzrinne.

## 8. REFERENCES

[US-ASCII] Coded Character Set--7-Bit American Standard Code for Information Interchange, ANSI X3.4-1986.

[RFC733] Crocker, D., Vittal, J., Pogran, K., and D. Henderson, "Standard for the Format of ARPA Network Text Message," RFC 733, November 1977.

[RFC822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

## 9. CONTACT

David H. Crocker
Internet Mail Consortium
675 Spruce Dr.
Sunnyvale, CA 94086 USA

Phone: +1 408 246 8253
Fax: +1 408 249 6205
EMail: dcrocker@imc.org

Paul Overell
Demon Internet Ltd
Dorking Business Park
Dorking
Surrey, RH4 1HN
UK

paulo@turnpike.com





## 10. Full Copyright Statement

Network Working Group
Request for Comments: 1034
Obsoletes: RFCs 882, 883, 973

P. Mockapetris
ISI

November 1987

# DOMAIN NAMES - CONCEPTS AND FACILITIES

## 1. STATUS OF THIS MEMO

This RFC is an introduction to the Domain Name System (DNS), and omits many details which can be found in a companion RFC, "Domain Names - Implementation and Specification" [RFC-1035]. That RFC assumes that the reader is familiar with the concepts discussed in this memo.

A subset of DNS functions and data types constitute an official protocol. The official protocol includes standard queries and their responses and most of the Internet class data formats (e.g., host addresses).

However, the domain system is intentionally extensible. Researchers are continuously proposing, implementing and experimenting with new data types, query types, classes, functions, etc. Thus while the components of the official protocol are expected to stay essentially unchanged and operate as a production service, experimental behavior should always be expected in extensions beyond the official protocol. Experimental or obsolete features are clearly marked in these RFCs, and such information should be used with caution.

The reader is especially cautioned not to depend on the values which appear in examples to be current or complete, since their purpose is primarily pedagogical. Distribution of this memo is unlimited.

## 2. INTRODUCTION

This RFC introduces domain style names, their use for Internet mail and host address support, and the protocols and servers used to implement domain name facilities.

### 2.1. The history of domain names

The impetus for the development of the domain system was growth in the Internet:

- Host name to address mappings were maintained by the Network Information Center (NIC) in a single file (HOSTS.TXT) which was FTPed by all hosts [RFC-952, RFC-953]. The total network





bandwidth consumed in distributing a new version by this scheme is proportional to the square of the number of hosts in the network, and even when multiple levels of FTP are used, the outgoing FTP load on the NIC host is considerable. Explosive growth in the number of hosts didn't bode well for the future.

- The network population was also changing in character. The timeshared hosts that made up the original ARPANET were being replaced with local networks of workstations. Local organizations were administering their own names and addresses, but had to wait for the NIC to change HOSTS.TXT to make changes visible to the Internet at large. Organizations also wanted some local structure on the name space.

- The applications on the Internet were getting more sophisticated and creating a need for general purpose name service.

The result was several ideas about name spaces and their management [IEN-116, RFC-799, RFC-819, RFC-830]. The proposals varied, but a common thread was the idea of a hierarchical name space, with the hierarchy roughly corresponding to organizational structure, and names using "." as the character to mark the boundary between hierarchy levels. A design using a distributed database and generalized resources was described in [RFC-882, RFC-883]. Based on experience with several implementations, the system evolved into the scheme described in this memo.

The terms "domain" or "domain name" are used in many contexts beyond the DNS described here. Very often, the term domain name is used to refer to a name with structure indicated by dots, but no relation to the DNS. This is particularly true in mail addressing [Quarterman 86].

### 2.2. DNS design goals

The design goals of the DNS influence its structure. They are:

- The primary goal is a consistent name space which will be used for referring to resources. In order to avoid the problems caused by ad hoc encodings, names should not be required to contain network identifiers, addresses, routes, or similar information as part of the name.

- The sheer size of the database and frequency of updates suggest that it must be maintained in a distributed manner, with local caching to improve performance. Approaches that

attempt to collect a consistent copy of the entire database will become more and more expensive and difficult, and hence should be avoided. The same principle holds for the structure of the name space, and in particular mechanisms for creating and deleting names; these should also be distributed.

- Where there tradeoffs between the cost of acquiring data, the speed of updates, and the accuracy of caches, the source of the data should control the tradeoff.

- The costs of implementing such a facility dictate that it be generally useful, and not restricted to a single application. We should be able to use names to retrieve host addresses, mailbox data, and other as yet undetermined information. All data associated with a name is tagged with a type, and queries can be limited to a single type.

- Because we want the name space to be useful in dissimilar networks and applications, we provide the ability to use the same name space with different protocol families or management. For example, host address formats differ between protocols, though all protocols have the notion of address. The DNS tags all data with a class as well as the type, so that we can allow parallel use of different formats for data of type address.

- We want name server transactions to be independent of the communications system that carries them. Some systems may wish to use datagrams for queries and responses, and only establish virtual circuits for transactions that need the reliability (e.g., database updates, long transactions); other systems will use virtual circuits exclusively.

- The system should be useful across a wide spectrum of host capabilities. Both personal computers and large timeshared hosts should be able to use the system, though perhaps in different ways.

## 2.3. Assumptions about usage

The organization of the domain system derives from some assumptions about the needs and usage patterns of its user community and is designed to avoid many of the the complicated problems found in general purpose database systems.

The assumptions are:

- The size of the total database will initially be proportional







to the number of hosts using the system, but will eventually grow to be proportional to the number of users on those hosts as mailboxes and other information are added to the domain system.

- Most of the data in the system will change very slowly (e.g., mailbox bindings, host addresses), but that the system should be able to deal with subsets that change more rapidly (on the order of seconds or minutes).

- The administrative boundaries used to distribute responsibility for the database will usually correspond to organizations that have one or more hosts. Each organization that has responsibility for a particular set of domains will provide redundant name servers, either on the organization's own hosts or other hosts that the organization arranges to use.

- Clients of the domain system should be able to identify trusted name servers they prefer to use before accepting referrals to name servers outside of this "trusted" set.

- Access to information is more critical than instantaneous updates or guarantees of consistency. Hence the update process allows updates to percolate out through the users of the domain system rather than guaranteeing that all copies are simultaneously updated. When updates are unavailable due to network or host failure, the usual course is to believe old information while continuing efforts to update it. The general model is that copies are distributed with timeouts for refreshing. The distributor sets the timeout value and the recipient of the distribution is responsible for performing the refresh. In special situations, very short intervals can be specified, or the owner can prohibit copies.

- In any system that has a distributed database, a particular name server may be presented with a query that can only be answered by some other server. The two general approaches to dealing with this problem are "recursive", in which the first server pursues the query for the client at another server, and "iterative", in which the server refers the client to another server and lets the client pursue the query. Both approaches have advantages and disadvantages, but the iterative approach is preferred for the datagram style of access. The domain system requires implementation of the iterative approach, but allows the recursive approach as an option.

The domain system assumes that all data originates in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files are text files that are read by a local name server, and hence become available through the name servers to users of the domain system. The user programs access name servers through standard programs called resolvers.

The standard format of master files allows them to be exchanged between hosts (via FTP, mail, or some other mechanism); this facility is useful when an organization wants a domain, but doesn't want to support a name server. The organization can maintain the master files locally using a text editor, transfer them to a foreign host which runs a name server, and then arrange with the system administrator of the name server to get the files loaded.

Each host's name servers and resolvers are configured by a local system administrator [RFC-1033]. For a name server, this configuration data includes the identity of local master files and instructions on which non-local master files are to be loaded from foreign servers. The name server uses the master files or copies to load its zones. For resolvers, the configuration data identifies the name servers which should be the primary sources of information.

The domain system defines procedures for accessing the data and for referrals to other name servers. The domain system also defines procedures for caching retrieved data and for periodic refreshing of data defined by the system administrator.

The system administrators provide:

- The definition of zone boundaries.
- Master files of data.
- Updates to master files.
- Statements of the refresh policies desired.

The domain system provides:

- Standard formats for resource data.
- Standard methods for querying the database.
- Standard methods for name servers to refresh local data from foreign name servers.





## 2.4. Elements of the DNS

The DNS has three major components:

- The DOMAIN NAME SPACE and RESOURCE RECORDS, which are specifications for a tree structured name space and data associated with the names. Conceptually, each node and leaf of the domain name space tree names a set of information, and query operations are attempts to extract specific types of information from a particular set. A query names the domain name of interest and describes the type of resource information that is desired. For example, the Internet uses some of its domain names to identify hosts; queries for address resources return Internet host addresses.

- NAME SERVERS are server programs which hold information about the domain tree's structure and set information. A name server may cache structure or set information about any part of the domain tree, but in general a particular name server has complete information about a subset of the domain space, and pointers to other name servers that can be used to lead to information from any part of the domain tree. Name servers know the parts of the domain tree for which they have complete information; a name server is said to be an AUTHORITY for these parts of the name space. Authoritative information is organized into units called ZONEs, and these zones can be automatically distributed to the name servers which provide redundant service for the data in a zone.

- RESOLVERS are programs that extract information from name servers in response to client requests. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers. A resolver will typically be a system routine that is directly accessible to user programs; hence no protocol is necessary between the resolver and the user program.

These three components roughly correspond to the three layers or views of the domain system:

- From the user's point of view, the domain system is accessed through a simple procedure or OS call to a local resolver. The domain space consists of a single tree and the user can request information from any section of the tree.

- From the resolver's point of view, the domain system is composed of an unknown number of name servers. Each name

server has one or more pieces of the whole domain tree's data, but the resolver views each of these databases as essentially static.

- From a name server's point of view, the domain system consists of separate sets of local information called zones. The name server has local copies of some of the zones. The name server must periodically refresh its zones from master copies in local files or foreign name servers. The name server must concurrently process queries that arrive from resolvers.

In the interests of performance, implementations may couple these functions. For example, a resolver on the same machine as a name server might share a database consisting of the the zones managed by the name server and the cache managed by the resolver.

## 3. DOMAIN NAME SPACE and RESOURCE RECORDS

### 3.1. Name space specifications and terminology

The domain name space is a tree structure. Each node and leaf on the tree corresponds to a resource set (which may be empty). The domain system makes no distinctions between the uses of the interior nodes and leaves, and this memo uses the term "node" to refer to both.

Each node has a label, which is zero to 63 octets in length. Brother nodes may not have the same label, although the same label can be used for nodes which are not brothers. One label is reserved, and that is the null (i.e., zero length) label used for the root.

The domain name of a node is the list of the labels on the path from the node to the root of the tree. By convention, the labels that compose a domain name are printed or read left to right, from the most specific (lowest, farthest from the root) to the least specific (highest, closest to the root).

Internally, programs that manipulate domain names should represent them as sequences of labels, where each label is a length octet followed by an octet string. Because all domain names end at the root, which has a null string for a label, these internal representations can use a length byte of zero to terminate a domain name.

By convention, domain names can be stored with arbitrary case, but domain name comparisons for all present domain functions are done in a case-insensitive manner, assuming an ASCII character set, and a high order zero bit. This means that you are free to create a node with label "A" or a node with label "a", but not both as brothers; you could refer to either using "a" or "A". When you receive a domain name or





label, you should preserve its case. The rationale for this choice is that we may someday need to add full binary domain names for new services; existing services would not be changed.

When a user needs to type a domain name, the length of each label is omitted and the labels are separated by dots ("."). Since a complete domain name ends with the root label, this leads to a printed form which ends in a dot. We use this property to distinguish between:

- a character string which represents a complete domain name (often called "absolute"). For example, "poneria.ISI.EDU."

- a character string that represents the starting labels of a domain name which is incomplete, and should be completed by local software using knowledge of the local domain (often called "relative"). For example, "poneria" used in the ISI.EDU domain.

Relative names are either taken relative to a well known origin, or to a list of domains used as a search list. Relative names appear mostly at the user interface, where their interpretation varies from implementation to implementation, and in master files, where they are relative to a single origin domain name. The most common interpretation uses the root "." as either the single origin or as one of the members of the search list, so a multi-label relative name is often one where the trailing dot has been omitted to save typing.

To simplify implementations, the total number of octets that represent a domain name (i.e., the sum of all label octets and label lengths) is limited to 255.

A domain is identified by a domain name, and consists of that part of the domain name space that is at or below the domain name which specifies the domain. A domain is a subdomain of another domain if it is contained within that domain. This relationship can be tested by seeing if the subdomain's name ends with the containing domain's name. For example, A.B.C.D is a subdomain of B.C.D, C.D, D, and " ".

### 3.2. Administrative guidelines on use

As a matter of policy, the DNS technical specifications do not mandate a particular tree structure or rules for selecting labels; its goal is to be as general as possible, so that it can be used to build arbitrary applications. In particular, the system was designed so that the name space did not have to be organized along the lines of network boundaries, name servers, etc. The rationale for this is not that the name space should have no implied semantics, but rather that the choice of implied semantics should be left open to be used for the problem at

hand, and that different parts of the tree can have different implied semantics. For example, the IN-ADDR.ARPA domain is organized and distributed by network and host address because its role is to translate from network or host numbers to names; NetBIOS domains [RFC-1001, RFC-1002] are flat because that is appropriate for that application.

However, there are some guidelines that apply to the "normal" parts of the name space used for hosts, mailboxes, etc., that will make the name space more uniform, provide for growth, and minimize problems as software is converted from the older host table. The political decisions about the top levels of the tree originated in RFC-920. Current policy for the top levels is discussed in [RFC-1032]. MILNET conversion issues are covered in [RFC-1031].

Lower domains which will eventually be broken into multiple zones should provide branching at the top of the domain so that the eventual decomposition can be done without renaming. Node labels which use special characters, leading digits, etc., are likely to break older software which depends on more restrictive choices.

### 3.3. Technical guidelines on use

Before the DNS can be used to hold naming information for some kind of object, two needs must be met:

- A convention for mapping between object names and domain names. This describes how information about an object is accessed.

- RR types and data formats for describing the object.

These rules can be quite simple or fairly complex. Very often, the designer must take into account existing formats and plan for upward compatibility for existing usage. Multiple mappings or levels of mapping may be required.

For hosts, the mapping depends on the existing syntax for host names which is a subset of the usual text representation for domain names, together with RR formats for describing host addresses, etc. Because we need a reliable inverse mapping from address to host name, a special mapping for addresses into the IN-ADDR.ARPA domain is also defined.

For mailboxes, the mapping is slightly more complex. The usual mail address <local-part>@<mail-domain> is mapped into a domain name by converting <local-part> into a single label (regardles of dots it contains), converting <mail-domain> into a domain name using the usual text format for domain names (dots denote label breaks), and concatenating the two to form a single domain name. Thus the mailbox



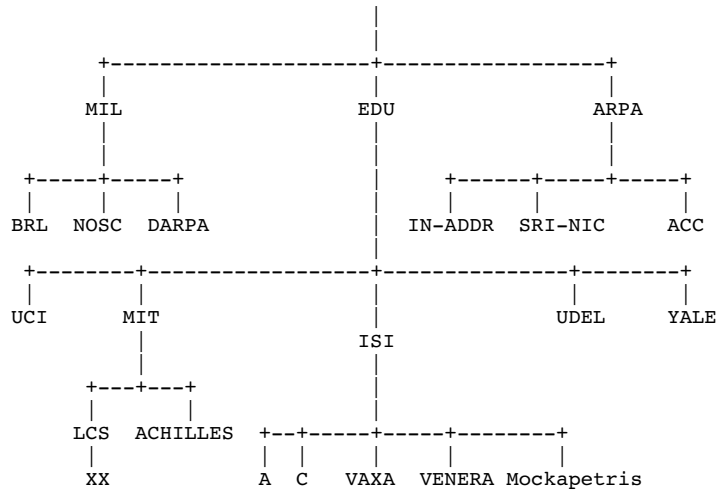


HOSTMASTER@SRI-NIC.ARPA is represented as a domain name by HOSTMASTER.SRI-NIC.ARPA. An appreciation for the reasons behind this design also must take into account the scheme for mail exchanges [RFC-974].

The typical user is not concerned with defining these rules, but should understand that they usually are the result of numerous compromises between desires for upward compatibility with old usage, interactions between different object definitions, and the inevitable urge to add new features when defining the rules. The way the DNS is used to support some object is often more crucial than the restrictions inherent in the DNS.

### 3.4. Example name space

The following figure shows a part of the current domain name space, and is used in many examples in this RFC. Note that the tree is a very small subset of the actual name space.




In this example, the root domain has three immediate subdomains: MIL, EDU, and ARPA. The LCS.MIT.EDU domain has one immediate subdomain named XX.LCS.MIT.EDU. All of the leaves are also domains.

### 3.5. Preferred name syntax

The DNS specifications attempt to be as general as possible in the rules

for constructing domain names. The idea is that the name of any existing object can be expressed as a domain name with minimal changes. However, when assigning a domain name for an object, the prudent user will select a name which satisfies both the rules of the domain system and any existing rules for the object, whether these rules are published or implied by existing programs.

For example, when naming a mail domain, the user should satisfy both the rules of this memo and those in RFC-822. When creating a new host name, the old rules for HOSTS.TXT should be followed. This avoids problems when old software is converted to use domain names.

The following syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET).

<domain> ::= <subdomain> | " "

<subdomain> ::= <label> | <subdomain> "." <label>

<label> ::= <letter> [ [ <ldh-str> ] <let-dig> ]

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig-hyp> ::= <let-dig> | "-"

<let-dig> ::= <letter> | <digit>

<letter> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case

<digit> ::= any one of the ten digits 0 through 9

Note that while upper and lower case letters are allowed in domain names, no significance is attached to the case. That is, two names with the same spelling but different case are to be treated as if identical.

The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less.

For example, the following strings identify hosts in the Internet:

A.ISI.EDU XX.LCS.MIT.EDU SRI-NIC.ARPA

### 3.6. Resource Records

A domain name identifies a node. Each node has a set of resource





information, which may be empty. The set of resource information associated with a particular name is composed of separate resource records (RRs). The order of RRs in a set is not significant, and need not be preserved by name servers, resolvers, or other parts of the DNS.

When we talk about a specific RR, we assume it has the following:

owner — which is the domain name where the RR is found.

type — which is an encoded 16 bit value that specifies the type of the resource in this resource record. Types refer to abstract resources.

This memo uses the following types:

| | |
|---|---|
| A | a host address |
| CNAME | identifies the canonical name of an alias |
| HINFO | identifies the CPU and OS used by a host |
| MX | identifies a mail exchange for the domain. See [RFC-974 for details. |
| NS | the authoritative name server for the domain |
| PTR | a pointer to another part of the domain name space |
| SOA | identifies the start of a zone of authority] |

class — which is an encoded 16 bit value which identifies a protocol family or instance of a protocol.

This memo uses the following classes:

| | |
|---|---|
| IN | the Internet system |
| CH | the Chaos system |

TTL — which is the time to live of the RR. This field is a 32 bit integer in units of seconds, an is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.

RDATA which is the type and sometimes class dependent data which describes the resource:

| | |
|---|---|
| A | For the IN class, a 32 bit IP address |
| | For the CH class, a domain name followed by a 16 bit octal Chaos address. |
| CNAME | a domain name. |
| MX | a 16 bit preference value (lower is better) followed by a host name willing to act as a mail exchange for the owner domain. |
| NS | a host name. |
| PTR | a domain name. |
| SOA | several fields. |

The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL) which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

The meaning of the TTL field is a time limit on how long an RR can be kept in a cache. This limit does not apply to authoritative data in zones; it is also timed out, but by the refreshing policies for the zone. The TTL is assigned by the administrator for the zone where the data originates. While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of Internet performance suggest that these times should be on the order of days for the typical host. If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change.

The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

### 3.6.1. Textual expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In this memo, we adopt a style similar to that used





in master files in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parentheses.

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

Following the owner, we list the TTL, type, and class of the RR. Class and type use the mnemonics defined above, and TTL is an integer before the type field. In order to avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interests of clarity.

The resource data or RDATA section of the RR are given using knowledge of the typical representation for the data.

For example, we might show the RRs carried in a message as:

```
    ISI.EDU.        MX      10 VENERA.ISI.EDU.
                    MX      10 VAXA.ISI.EDU.
    VENERA.ISI.EDU. A       128.9.0.32
                    A       10.1.0.52
    VAXA.ISI.EDU.   A       10.2.0.27
                    A       128.9.0.33
```

The MX RRs have an RDATA section which consists of a 16 bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32 bit internet address.

This example shows six RRs, with two RRs at each of three domain names.

Similarly we might see:

```
    XX.LCS.MIT.EDU. IN      A       10.0.0.44
                    CH      A       MIT.EDU. 2420
```

This example shows two addresses for XX.LCS.MIT.EDU, each of a different class.

### 3.6.2. Aliases and canonical names

In existing systems, hosts and other resources often have several names that identify the same resource. For example, the names C.ISI.EDU and USC-ISIC.ARPA both identify the same host. Similarly, in the case of mailboxes, many organizations provide many names that actually go to the same mailbox; for example Mockapetris@C.ISI.EDU, Mockapetris@B.ISI.EDU,

and PVM@ISI.EDU all go to the same mailbox (although the mechanism behind this is somewhat complicated).

Most of these systems have a notion that one of the equivalent set of names is the canonical or primary name and all others are aliases.

The domain system provides such a feature using the canonical name (CNAME) RR. A CNAME RR identifies its owner name as an alias, and specifies the corresponding canonical name in the RDATA section of the RR. If a CNAME RR is present at a node, no other data should be present; this ensures that the data for a canonical name and its aliases cannot be different. This rule also insures that a cached CNAME can be used without checking with an authoritative server for other RR types.

CNAME RRs cause special action in DNS software. When a name server fails to find a desired RR in the resource set associated with the domain name, it checks to see if the resource set consists of a CNAME record with a matching class. If so, the name server includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record. The one exception to this rule is that queries which match the CNAME type are not restarted.

For example, suppose a name server was processing a query with for USC-ISIC.ARPA, asking for type A information, and had the following resource records:

```
    USC-ISIC.ARPA   IN      CNAME   C.ISI.EDU

    C.ISI.EDU       IN      A       10.0.0.52
```

Both of these RRs would be returned in the response to the type A query, while a type CNAME or * query should return just the CNAME.

Domain names in RRs which point at another name should always point at the primary name and not the alias. This avoids extra indirections in accessing information. For example, the address to name RR for the above host should be:

```
    52.0.0.10.IN-ADDR.ARPA  IN      PTR     C.ISI.EDU
```

rather than pointing at USC-ISIC.ARPA. Of course, by the robustness principle, domain software should not fail when presented with CNAME chains or loops; CNAME chains should be followed and CNAME loops signalled as an error.

### 3.7. Queries

Queries are messages which may be sent to a name server to provoke a





response. In the Internet, queries are carried in UDP datagrams or over TCP connections. The response by the name server either answers the question posed in the query, refers the requester to another set of name servers, or signals some error condition.

In general, the user does not generate queries directly, but instead makes a request to a resolver which in turn sends one or more queries to name servers and deals with the error conditions and referrals that may result. Of course, the possible questions which can be asked in a query does shape the kind of service a resolver can provide.

DNS queries and responses are carried in a standard message format. The message format has a header containing a number of fixed fields which are always present, and four sections which carry query parameters and RRs.

The most important field in the header is a four bit field called an opcode which separates different queries. Of the possible 16 values, one (standard query) is part of the official protocol, two (inverse query and status query) are options, one (completion) is obsolete, and the rest are unassigned.

The four sections are:

| | |
|---|---|
| Question | Carries the query name and other query parameters. |
| Answer | Carries RRs which directly answer the query. |
| Authority | Carries RRs which describe other authoritative servers. May optionally carry the SOA RR for the authoritative data in the answer section. |
| Additional | Carries RRs which may be helpful in using the RRs in the other sections. |

Note that the content, but not the format, of these sections varies with header opcode.

#### 3.7.1. Standard queries

A standard query specifies a target domain name (QNAME), query type (QTYPE), and query class (QCLASS) and asks for RRs which match. This type of query makes up such a vast majority of DNS queries that we use the term "query" to mean standard query unless otherwise specified. The QTYPE and QCLASS fields are each 16 bits long, and are a superset of defined types and classes.

The QTYPE field may contain:

<any type> matches just that type. (e.g., A, PTR).

AXFR special zone transfer QTYPE.

MAILB matches all mail box related RRs (e.g. MB and MG).

* matches all RR types.

The QCLASS field may contain:

<any class> matches just that class (e.g., IN, CH).

* matches aLL RR classes.

Using the query domain name, QTYPE, and QCLASS, the name server looks for matching RRs. In addition to relevant records, the name server may return RRs that point toward a name server that has the desired information or RRs that are expected to be useful in interpreting the relevant RRs. For example, a name server that doesn't have the requested information may know a name server that does; a name server that returns a domain name in a relevant RR may also return the RR that binds that domain name to an address.

For example, a mailer tying to send mail to Mockapetris@ISI.EDU might ask the resolver for mail information about ISI.EDU, resulting in a query for QNAME=ISI.EDU, QTYPE=MX, QCLASS=IN. The response's answer section would be:

```
    ISI.EDU.        MX      10 VENERA.ISI.EDU.
                    MX      10 VAXA.ISI.EDU.
```

while the additional section might be:

```
    VAXA.ISI.EDU.   A       10.2.0.27
                    A       128.9.0.33
    VENERA.ISI.EDU. A       10.1.0.52
                    A       128.9.0.32
```

Because the server assumes that if the requester wants mail exchange information, it will probably want the addresses of the mail exchanges soon afterward.

Note that the QCLASS=* construct requires special interpretation regarding authority. Since a particular name server may not know all of the classes available in the domain system, it can never know if it is authoritative for all classes. Hence responses to QCLASS=* queries can





never be authoritative.

3.7.2. Inverse queries (Optional)

Name servers may also support inverse queries that map a particular resource to a domain name or domain names that have that resource. For example, while a standard query might map a domain name to a SOA RR, the corresponding inverse query might map the SOA RR back to the domain name.

Implementation of this service is optional in a name server, but all name servers must at least be able to understand an inverse query message and return a not-implemented error response.

The domain system cannot guarantee the completeness or uniqueness of inverse queries because the domain system is organized by domain name rather than by host address or any other resource type. Inverse queries are primarily useful for debugging and database maintenance activities.

Inverse queries may not return the proper TTL, and do not indicate cases where the identified RR is one of a set (for example, one address for a host having multiple addresses). Therefore, the RRs returned in inverse queries should never be cached.

Inverse queries are NOT an acceptable method for mapping host addresses to host names; use the IN-ADDR.ARPA domain instead.

A detailed discussion of inverse queries is contained in [RFC-1035].

3.8. Status queries (Experimental)

To be defined.

3.9. Completion queries (Obsolete)

The optional completion services described in RFCs 882 and 883 have been deleted. Redesigned services may become available in the future, or the opcodes may be reclaimed for other use.

4. NAME SERVERS

4.1. Introduction

Name servers are the repositories of information that make up the domain database. The database is divided up into sections called zones, which are distributed among the name servers. While name servers can have several optional functions and sources of data, the essential task of a name server is to answer queries using data in its zones. By design,

name servers can answer queries in a simple manner; the response can always be generated using only local data, and either contains the answer to the question or a referral to other name servers "closer" to the desired information.

A given zone will be available from several name servers to insure its availability in spite of host or communication link failure. By administrative fiat, we require every zone to be available on at least two servers, and many zones have more redundancy than that.

A given name server will typically support one or more zones, but this gives it authoritative information about only a small section of the domain tree. It may also have some cached non-authoritative data about other parts of the tree. The name server marks its responses to queries so that the requester can tell whether the response comes from authoritative data or not.

4.2. How the database is divided into zones

The domain database is partitioned in two ways: by class, and by "cuts" made in the name space between nodes.

The class partition is simple. The database for any class is organized, delegated, and maintained separately from all other classes. Since, by convention, the name spaces are the same for all classes, the separate classes can be thought of as an array of parallel namespace trees. Note that the data attached to nodes will be different for these different parallel classes. The most common reasons for creating a new class are the necessity for a new data format for existing types or a desire for a separately managed version of the existing name space.

Within a class, "cuts" in the name space can be made between any two adjacent nodes. After all cuts are made, each group of connected name space is a separate zone. The zone is said to be authoritative for all names in the connected region. Note that the "cuts" in the name space may be in different places for different classes, the name servers may be different, etc.

These rules mean that every zone has at least one node, and hence domain name, for which it is authoritative, and all of the nodes in a particular zone are connected. Given, the tree structure, every zone has a highest node which is closer to the root than any other node in the zone. The name of this node is often used to identify the zone.

It would be possible, though not particularly useful, to partition the name space so that each domain name was in a separate zone or so that all nodes were in a single zone. Instead, the database is partitioned at points where a particular organization wants to take over control of







a subtree. Once an organization controls its own zone it can unilaterally change the data in the zone, grow new tree sections connected to the zone, delete existing nodes, or delegate new subzones under its zone.

If the organization has substructure, it may want to make further internal partitions to achieve nested delegations of name space control. In some cases, such divisions are made purely to make database maintenance more convenient.

4.2.1. Technical considerations

The data that describes a zone has four major parts:

- Authoritative data for all nodes within the zone.
- Data that defines the top node of the zone (can be thought of as part of the authoritative data).
- Data that describes delegated subzones, i.e., cuts around the bottom of the zone.
- Data that allows access to name servers for subzones (sometimes called "glue" data).

All of this data is expressed in the form of RRs, so a zone can be completely described in terms of a set of RRs. Whole zones can be transferred between name servers by transferring the RRs, either carried in a series of messages or by FTPing a master file which is a textual representation.

The authoritative data for a zone is simply all of the RRs attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone.

Though logically part of the authoritative data, the RRs that describe the top node of the zone are especially important to the zone's management. These RRs are of two types: name server RRs that list, one per RR, all of the servers for the zone, and a single SOA RR that describes zone management parameters.

The RRs that describe cuts around the bottom of the zone are NS RRs that name the servers for the subzones. Since the cuts are between nodes, these RRs are NOT part of the authoritative data of the zone, and should be exactly the same as the corresponding RRs in the top node of the subzone. Since name servers are always associated with zone boundaries, NS RRs are only found at nodes which are the top node of some zone. In the data that makes up a zone, NS RRs are found at the top node of the

zone (and are authoritative) and at cuts around the bottom of the zone (where they are not authoritative), but never in between.

One of the goals of the zone structure is that any zone have all the data required to set up communications with the name servers for any subzones. That is, parent zones have all the information needed to access servers for their children zones. The NS RRs that name the servers for subzones are often not enough for this task since they name the servers, but do not give their addresses. In particular, if the name of the name server is itself in the subzone, we could be faced with the situation where the NS RRs tell us that in order to learn a name server's address, we should contact the server using the address we wish to learn. To fix this problem, a zone contains "glue" RRs which are not part of the authoritative data, and are address RRs for the servers. These RRs are only necessary if the name server's name is "below" the cut, and are only used as part of a referral response.

4.2.2. Administrative considerations

When some organization wants to control its own domain, the first step is to identify the proper parent zone, and get the parent zone's owners to agree to the delegation of control. While there are no particular technical constraints dealing with where in the tree this can be done, there are some administrative groupings discussed in [RFC-1032] which deal with top level organization, and middle level zones are free to create their own rules. For example, one university might choose to use a single zone, while another might choose to organize by subzones dedicated to individual departments or schools. [RFC-1033] catalogs available DNS software an discusses administration procedures.

Once the proper name for the new subzone is selected, the new owners should be required to demonstrate redundant name server support. Note that there is no requirement that the servers for a zone reside in a host which has a name in that domain. In many cases, a zone will be more accessible to the internet at large if its servers are widely distributed rather than being within the physical facilities controlled by the same organization that manages the zone. For example, in the current DNS, one of the name servers for the United Kingdom, or UK domain, is found in the US. This allows US hosts to get UK data without using limited transatlantic bandwidth.

As the last installation step, the delegation NS RRs and glue RRs necessary to make the delegation effective should be added to the parent zone. The administrators of both zones should insure that the NS and glue RRs which mark both sides of the cut are consistent and remain so.

4.3. Name server internals







4.3.1. Queries and responses

The principal activity of name servers is to answer standard queries. Both the query and its response are carried in a standard message format which is described in [RFC-1035]. The query contains a QTYPE, QCLASS, and QNAME, which describe the types and classes of desired information and the name of interest.

The way that the name server answers the query depends upon whether it is operating in recursive mode or not:

- The simplest mode for the server is non-recursive, since it can answer queries using only local information: the response contains an error, the answer, or a referral to some other server "closer" to the answer. All name servers must implement non-recursive queries.

- The simplest mode for the client is recursive, since in this mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals. This service is optional in a name server, and the name server may also choose to restrict the clients which can use recursive mode.

Recursive service is helpful in several situations:

- a relatively simple requester that lacks the ability to use anything other than a direct answer to the question.

- a request that needs to cross protocol or other boundaries and can be sent to a server which can act as intermediary.

- a network where we want to concentrate the cache rather than having a separate cache for each client.

Non-recursive service is appropriate if the requester is capable of pursuing referrals and interested in information which will aid future requests.

The use of recursive mode is limited to cases where both the client and the name server agree to its use. The agreement is negotiated through the use of two bits in query and response messages:

- The recursion available, or RA bit, is set or cleared by a name server in all responses. The bit is true if the name server is willing to provide recursive service for the client, regardless of whether the client requested recursive service. That is, RA signals availability rather than use.

- Queries contain a bit called recursion desired or RD. This bit specifies specifies whether the requester wants recursive service for this query. Clients may request recursive service from any name server, though they should depend upon receiving it only from servers which have previously sent an RA, or servers which have agreed to provide service through private agreement or some other means outside of the DNS protocol.

The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive mode was used by checking that both RA and RD are set in the reply. Note that the name server should never perform recursive service unless asked via RD, since this interferes with trouble shooting of name servers and their databases.

If recursive service is requested and available, the recursive response to a query will be one of the following:

- The answer to the query, possibly preface by one or more CNAME RRs that specify aliases encountered on the way to an answer.

- A name error indicating that the name does not exist. This may include CNAME RRs that indicate that the original query name was an alias for a name which does not exist.

- A temporary error indication.

If recursive service is not requested or is not available, the non-recursive response will be one of the following:

- An authoritative name error indicating that the name does not exist.

- A temporary error indication.

- Some combination of:

  RRs that answer the question, together with an indication whether the data comes from a zone or is cached.

  A referral to name servers which have zones which are closer ancestors to the name than the server sending the reply.

- RRs that the name server thinks will prove useful to the requester.





4.3.2. Algorithm

The actual algorithm used by the name server will depend on the local OS and data structures used to store RRs. The following algorithm assumes that the RRs are organized in several tree structures, one for each zone, and another for the cache:

1. Set or clear the value of recursion available in the response depending on whether the name server is willing to provide recursive service. If recursive service is available and requested via the RD bit in the query, go to step 5, otherwise step 2.

2. Search the available zones for the zone which is the nearest ancestor to QNAME. If such a zone is found, go to step 3, otherwise step 4.

3. Start matching down, label by label, in the zone. The matching process can terminate several ways:

   a. If the whole of QNAME is matched, we have found the node.

      If the data at the node is a CNAME, and QTYPE doesn't match CNAME, copy the CNAME RR into the answer section of the response, change QNAME to the canonical name in the CNAME RR, and go back to step 1.

      Otherwise, copy all RRs which match QTYPE into the answer section and go to step 6.

   b. If a match would take us out of the authoritative data, we have a referral. This happens when we encounter a node with NS RRs marking cuts along the bottom of a zone.

      Copy the NS RRs for the subzone into the authority section of the reply. Put whatever addresses are available into the additional section, using glue RRs if the addresses are not available from authoritative data or the cache. Go to step 4.

   c. If at some label, a match is impossible (i.e., the corresponding label does not exist), look to see if a the "*" label exists.

      If the "*" label does not exist, check whether the name we are looking for is the original QNAME in the query

or a name we have followed due to a CNAME. If the name is original, set an authoritative name error in the response and exit. Otherwise just exit.

If the "*" label does exist, match RRs at that node against QTYPE. If any match, copy them into the answer section, but set the owner of the RR to be QNAME, and not the node with the "*" label. Go to step 6.

4. Start matching down in the cache. If QNAME is found in the cache, copy all RRs attached to it that match QTYPE into the answer section. If there was no delegation from authoritative data, look for the best one from the cache, and put it in the authority section. Go to step 6.

5. Using the local resolver or a copy of its algorithm (see resolver section of this memo) to answer the query. Store the results, including any intermediate CNAMEs, in the answer section of the response.

6. Using local data only, attempt to add other RRs which may be useful to the additional section of the query. Exit.

4.3.3. Wildcards

In the previous algorithm, special treatment was given to RRs with owner names starting with the label "*". Such RRs are called wildcards. Wildcard RRs can be thought of as instructions for synthesizing RRs. When the appropriate conditions are met, the name server creates RRs with an owner name equal to the query name and contents taken from the wildcard RRs.

This facility is most often used to create a zone which will be used to forward mail from the Internet to some other mail system. The general idea is that any name in that zone which is presented to server in a query will be assumed to exist, with certain properties, unless explicit evidence exists to the contrary. Note that the use of the term zone here, instead of domain, is intentional; such defaults do not propagate across zone boundaries, although a subzone may choose to achieve that appearance by setting up similar defaults.

The contents of the wildcard RRs follows the usual rules and formats for RRs. The wildcards in the zone have an owner name that controls the query names they will match. The owner name of the wildcard RRs is of the form "*.<anydomain>", where <anydomain> is any domain name. <anydomain> should not contain other * labels, and should be in the authoritative data of the zone. The wildcards potentially apply to descendants of <anydomain>, but not to <anydomain> itself. Another way





to look at this is that the "*" label always matches at least one whole label and sometimes more, but always whole labels.

Wildcard RRs do not apply:

- When the query is in another zone. That is, delegation cancels the wildcard defaults.

- When the query name or a name between the wildcard domain and the query name is know to exist. For example, if a wildcard RR has an owner name of "*.X", and the zone also contains RRs attached to B.X, the wildcards would apply to queries for name Z.X (presuming there is no explicit information for Z.X), but not to B.X, A.B.X, or X.

A * label appearing in a query name has no special effect, but can be used to test for wildcards in an authoritative zone; such a query is the only way to get a response containing RRs with an owner name with * in it. The result of such a query should not be cached.

Note that the contents of the wildcard RRs are not modified when used to synthesize RRs.

To illustrate the use of wildcard RRs, suppose a large company with a large, non-IP/TCP, network wanted to create a mail gateway. If the company was called X.COM, and IP/TCP capable gateway machine was called A.X.COM, the following RRs might be entered into the COM zone:

```
X.COM           MX      10      A.X.COM

*.X.COM         MX      10      A.X.COM

A.X.COM         A       1.2.3.4
A.X.COM         MX      10      A.X.COM

*.A.X.COM       MX      10      A.X.COM
```

This would cause any MX query for any domain name ending in X.COM to return an MX RR pointing at A.X.COM. Two wildcard RRs are required since the effect of the wildcard at *.X.COM is inhibited in the A.X.COM subtree by the explicit data for A.X.COM. Note also that the explicit MX data at X.COM and A.X.COM is required, and that none of the RRs above would match a query name of XX.COM.

4.3.4. Negative response caching (Optional)

The DNS provides an optional service which allows name servers to distribute, and resolvers to cache, negative results with TTLs. For

example, a name server can distribute a TTL along with a name error indication, and a resolver receiving such information is allowed to assume that the name does not exist during the TTL period without consulting authoritative data. Similarly, a resolver can make a query with a QTYPE which matches multiple types, and cache the fact that some of the types are not present.

This feature can be particularly important in a system which implements naming shorthands that use search lists beacuse a popular shorthand, which happens to require a suffix toward the end of the search list, will generate multiple name errors whenever it is used.

The method is that a name server may add an SOA RR to the additional section of a response when that response is authoritative. The SOA must be that of the zone which was the source of the authoritative data in the answer section, or name error if applicable. The MINIMUM field of the SOA controls the length of time that the negative result may be cached.

Note that in some circumstances, the answer section may contain multiple owner names. In this case, the SOA mechanism should only be used for the data which matches QNAME, which is the only authoritative data in this section.

Name servers and resolvers should never attempt to add SOAs to the additional section of a non-authoritative response, or attempt to infer results which are not directly stated in an authoritative response. There are several reasons for this, including: cached information isn't usually enough to match up RRs and their zone names, SOA RRs may be cached due to direct SOA queries, and name servers are not required to output the SOAs in the authority section.

This feature is optional, although a refined version is expected to become part of the standard protocol in the future. Name servers are not required to add the SOA RRs in all authoritative responses, nor are resolvers required to cache negative results. Both are recommended. All resolvers and recursive name servers are required to at least be able to ignore the SOA RR when it is present in a response.

Some experiments have also been proposed which will use this feature. The idea is that if cached data is known to come from a particular zone, and if an authoritative copy of the zone's SOA is obtained, and if the zone's SERIAL has not changed since the data was cached, then the TTL of the cached data can be reset to the zone MINIMUM value if it is smaller. This usage is mentioned for planning purposes only, and is not recommended as yet.







### 4.3.5. Zone maintenance and transfers

Part of the job of a zone administrator is to maintain the zones at all of the name servers which are authoritative for the zone. When the inevitable changes are made, they must be distributed to all of the name servers. While this distribution can be accomplished using FTP or some other ad hoc procedure, the preferred method is the zone transfer part of the DNS protocol.

The general model of automatic zone transfer or refreshing is that one of the name servers is the master or primary for the zone. Changes are coordinated at the primary, typically by editing a master file for the zone. After editing, the administrator signals the master server to load the new zone. The other non-master or secondary servers for the zone periodically check for changes (at a selectable interval) and obtain new zone copies when changes have been made.

To detect changes, secondaries just check the SERIAL field of the SOA for the zone. In addition to whatever other changes are made, the SERIAL field in the SOA of the zone is always advanced whenever any change is made to the zone. The advancing can be a simple increment, or could be based on the write date and time of the master file, etc. The purpose is to make it possible to determine which of two copies of a zone is more recent by comparing serial numbers. Serial number advances and comparisons use sequence space arithmetic, so there is a theoretic limit on how fast a zone can be updated, basically that old copies must die out before the serial number covers half of its 32 bit range. In practice, the only concern is that the compare operation deals properly with comparisons around the boundary between the most positive and most negative 32 bit numbers.

The periodic polling of the secondary servers is controlled by parameters in the SOA RR for the zone, which set the minimum acceptable polling intervals. The parameters are called REFRESH, RETRY, and EXPIRE. Whenever a new zone is loaded in a secondary, the secondary waits REFRESH seconds before checking with the primary for a new serial. If this check cannot be completed, new checks are started every RETRY seconds. The check is a simple query to the primary for the SOA RR of the zone. If the serial field in the secondary's zone copy is equal to the serial returned by the primary, then no changes have occurred, and the REFRESH interval wait is restarted. If the secondary finds it impossible to perform a serial check for the EXPIRE interval, it must assume that its copy of the zone is obsolete an discard it.

When the poll shows that the zone has changed, then the secondary server must request a zone transfer via an AXFR request for the zone. The AXFR may cause an error, such as refused, but normally is answered by a sequence of response messages. The first and last messages must contain

the data for the top authoritative node of the zone. Intermediate messages carry all of the other RRs from the zone, including both authoritative and non-authoritative RRs. The stream of messages allows the secondary to construct a copy of the zone. Because accuracy is essential, TCP or some other reliable protocol must be used for AXFR requests.

Each secondary server is required to perform the following operations against the master, but may also optionally perform these operations against other secondary servers. This strategy can improve the transfer process when the primary is unavailable due to host downtime or network problems, or when a secondary server has better network access to an "intermediate" secondary than to the primary.

## 5. RESOLVERS

### 5.1. Introduction

Resolvers are programs that interface user programs to domain name servers. In the simplest case, a resolver receives a request from a user program (e.g., mail programs, TELNET, FTP) in the form of a subroutine call, system call etc., and returns the desired information in a form compatible with the local host's data formats.

The resolver is located on the same machine as the program that requests the resolver's services, but it may need to consult name servers on other hosts. Because a resolver may need to consult several name servers, or may have the requested information in a local cache, the amount of time that a resolver will take to complete can vary quite a bit, from milliseconds to several seconds.

A very important goal of the resolver is to eliminate network delay and name server load from most requests by answering them from its cache of prior results. It follows that caches which are shared by multiple processes, users, machines, etc., are more efficient than non-shared caches.

### 5.2. Client-resolver interface

#### 5.2.1. Typical functions

The client interface to the resolver is influenced by the local host's conventions, but the typical resolver-client interface has three functions:

1. Host name to host address translation.

   This function is often defined to mimic a previous HOSTS.TXT based function. Given a character string, the caller wants one or more 32 bit IP addresses. Under the DNS, it translates into a request for type A RRs. Since the DNS does not preserve the order of RRs, this function may choose to sort the returned addresses or select the "best" address if the service returns only one choice to the client. Note that a multiple address return is recommended, but a single address may be the only way to emulate prior HOSTS.TXT services.

2. Host address to host name translation

   This function will often follow the form of previous functions. Given a 32 bit IP address, the caller wants a character string. The octets of the IP address are reversed, used as name components, and suffixed with "IN-ADDR.ARPA". A type PTR query is used to get the RR with the primary name of the host. For example, a request for the host name corresponding to IP address 1.2.3.4 looks for PTR RRs for domain name "4.3.2.1.IN-ADDR.ARPA".

3. General lookup function

   This function retrieves arbitrary information from the DNS, and has no counterpart in previous systems. The caller supplies a QNAME, QTYPE, and QCLASS, and wants all of the matching RRs. This function will often use the DNS format for all RR data instead of the local host's, and returns all RR content (e.g., TTL) instead of a processed form with local quoting conventions.

When the resolver performs the indicated function, it usually has one of the following results to pass back to the client:

- One or more RRs giving the requested data.

  In this case the resolver returns the answer in the appropriate format.

- A name error (NE).

  This happens when the referenced name does not exist. For example, a user may have mistyped a host name.

- A data not found error.

  This happens when the referenced name exists, but data of the appropriate type does not. For example, a host address

function applied to a mailbox name would return this error since the name exists, but no address RR is present.

It is important to note that the functions for translating between host names and addresses may combine the "name error" and "data not found" error conditions into a single type of error return, but the general function should not. One reason for this is that applications may ask first for one type of information about a name followed by a second request to the same name for some other type of information; if the two errors are combined, then useless queries may slow the application.

5.2.2. Aliases

While attempting to resolve a particular request, the resolver may find that the name in question is an alias. For example, the resolver might find that the name given for host name to address translation is an alias when it finds the CNAME RR. If possible, the alias condition should be signalled back from the resolver to the client.

In most cases a resolver simply restarts the query at the new name when it encounters a CNAME. However, when performing the general function, the resolver should not pursue aliases when the CNAME RR matches the query type. This allows queries which ask whether an alias is present. For example, if the query type is CNAME, the user is interested in the CNAME RR itself, and not the RRs at the name it points to.

Several special conditions can occur with aliases. Multiple levels of aliases should be avoided due to their lack of efficiency, but should not be signalled as an error. Alias loops and aliases which point to non-existent names should be caught and an error condition passed back to the client.

5.2.3. Temporary failures

In a less than perfect world, all resolvers will occasionally be unable to resolve a particular request. This condition can be caused by a resolver which becomes separated from the rest of the network due to a link failure or gateway problem, or less often by coincident failure or unavailability of all servers for a particular domain.

It is essential that this sort of condition should not be signalled as a name or data not present error to applications. This sort of behavior is annoying to humans, and can wreak havoc when mail systems use the DNS.

While in some cases it is possible to deal with such a temporary problem by blocking the request indefinitely, this is usually not a good choice, particularly when the client is a server process that could move on to





other tasks. The recommended solution is to always have temporary failure as one of the possible results of a resolver function, even though this may make emulation of existing HOSTS.TXT functions more difficult.

5.3. Resolver internals

Every resolver implementation uses slightly different algorithms, and typically spends much more logic dealing with errors of various sorts than typical occurances. This section outlines a recommended basic strategy for resolver operation, but leaves details to [RFC-1035].

5.3.1. Stub resolvers

One option for implementing a resolver is to move the resolution function out of the local machine and into a name server which supports recursive queries. This can provide an easy method of providing domain service in a PC which lacks the resources to perform the resolver function, or can centralize the cache for a whole local network or organization.

All that the remaining stub needs is a list of name server addresses that will perform the recursive requests. This type of resolver presumably needs the information in a configuration file, since it probably lacks the sophistication to locate it in the domain database. The user also needs to verify that the listed servers will perform the recursive service; a name server is free to refuse to perform recursive services for any or all clients. The user should consult the local system administrator to find name servers willing to perform the service.

This type of service suffers from some drawbacks. Since the recursive requests may take an arbitrary amount of time to perform, the stub may have difficulty optimizing retransmission intervals to deal with both lost UDP packets and dead servers; the name server can be easily overloaded by too zealous a stub if it interprets retransmissions as new requests. Use of TCP may be an answer, but TCP may well place burdens on the host's capabilities which are similar to those of a real resolver.

5.3.2. Resources

In addition to its own resources, the resolver may also have shared access to zones maintained by a local name server. This gives the resolver the advantage of more rapid access, but the resolver must be careful to never let cached information override zone data. In this discussion the term "local information" is meant to mean the union of the cache and such shared zones, with the understanding that

authoritative data is always used in preference to cached data when both are present.

The following resolver algorithm assumes that all functions have been converted to a general lookup function, and uses the following data structures to represent the state of a request in progress in the resolver:

SNAME — the domain name we are searching for.

STYPE — the QTYPE of the search request.

SCLASS — the QCLASS of the search request.

SLIST — a structure which describes the name servers and the zone which the resolver is currently trying to query. This structure keeps track of the resolver's current best guess about which name servers hold the desired information; it is updated when arriving information changes the guess. This structure includes the equivalent of a zone name, the known name servers for the zone, the known addresses for the name servers, and history information which can be used to suggest which server is likely to be the best one to try next. The zone name equivalent is a match count of the number of labels from the root down which SNAME has in common with the zone being queried; this is used as a measure of how "close" the resolver is to SNAME.

SBELT — a "safety belt" structure of the same form as SLIST, which is initialized from a configuration file, and lists servers which should be used when the resolver doesn't have any local information to guide name server selection. The match count will be -1 to indicate that no labels are known to match.

CACHE — A structure which stores the results from previous responses. Since resolvers are responsible for discarding old RRs whose TTL has expired, most implementations convert the interval specified in arriving RRs to some sort of absolute time when the RR is stored in the cache. Instead of counting the TTLs down individually, the resolver just ignores or discards old RRs when it runs across them in the course of a search, or discards them during periodic sweeps to reclaim the memory consumed by old RRs.





5.3.3. Algorithm

The top level algorithm has four steps:

1. See if the answer is in local information, and if so return it to the client.

2. Find the best servers to ask.

3. Send them queries until one returns a response.

4. Analyze the response, either:

   a. if the response answers the question or contains a name error, cache the data as well as returning it back to the client.

   b. if the response contains a better delegation to other servers, cache the delegation information, and go to step 2.

   c. if the response shows a CNAME and that is not the answer itself, cache the CNAME, change the SNAME to the canonical name in the CNAME RR and go to step 1.

   d. if the response shows a servers failure or other bizarre contents, delete the server from the SLIST and go back to step 3.

Step 1 searches the cache for the desired data. If the data is in the cache, it is assumed to be good enough for normal use. Some resolvers have an option at the user interface which will force the resolver to ignore the cached data and consult with an authoritative server. This is not recommended as the default. If the resolver has direct access to a name server's zones, it should check to see if the desired data is present in authoritative form, and if so, use the authoritative data in preference to cached data.

Step 2 looks for a name server to ask for the required data. The general strategy is to look for locally-available name server RRs, starting at SNAME, then the parent domain name of SNAME, the grandparent, and so on toward the root. Thus if SNAME were Mockapetris.ISI.EDU, this step would look for NS RRs for Mockapetris.ISI.EDU, then ISI.EDU, then EDU, and then . (the root). These NS RRs list the names of hosts for a zone at or above SNAME. Copy the names into SLIST. Set up their addresses using local data. It may be the case that the addresses are not available. The resolver has many choices here; the best is to start parallel resolver processes looking

for the addresses while continuing onward with the addresses which are available. Obviously, the design choices and options are complicated and a function of the local host's capabilities. The recommended priorities for the resolver designer are:

1. Bound the amount of work (packets sent, parallel processes started) so that a request can't get into an infinite loop or start off a chain reaction of requests or queries with other implementations EVEN IF SOMEONE HAS INCORRECTLY CONFIGURED SOME DATA.

2. Get back an answer if at all possible.

3. Avoid unnecessary transmissions.

4. Get the answer as quickly as possible.

If the search for NS RRs fails, then the resolver initializes SLIST from the safety belt SBELT. The basic idea is that when the resolver has no idea what servers to ask, it should use information from a configuration file that lists several servers which are expected to be helpful. Although there are special situations, the usual choice is two of the root servers and two of the servers for the host's domain. The reason for two of each is for redundancy. The root servers will provide eventual access to all of the domain space. The two local servers will allow the resolver to continue to resolve local names if the local network becomes isolated from the internet due to gateway or link failure.

In addition to the names and addresses of the servers, the SLIST data structure can be sorted to use the best servers first, and to insure that all addresses of all servers are used in a round-robin manner. The sorting can be a simple function of preferring addresses on the local network over others, or may involve statistics from past events, such as previous response times and batting averages.

Step 3 sends out queries until a response is received. The strategy is to cycle around all of the addresses for all of the servers with a timeout between each transmission. In practice it is important to use all addresses of a multihomed host, and too aggressive a retransmission policy actually slows response when used by multiple resolvers contending for the same name server and even occasionally for a single resolver. SLIST typically contains data values to control the timeouts and keep track of previous transmissions.

Step 4 involves analyzing responses. The resolver should be highly paranoid in its parsing of responses. It should also check that the response matches the query it sent using the ID field in the response.







The ideal answer is one from a server authoritative for the query which either gives the required data or a name error. The data is passed back to the user and entered in the cache for future use if its TTL is greater than zero.

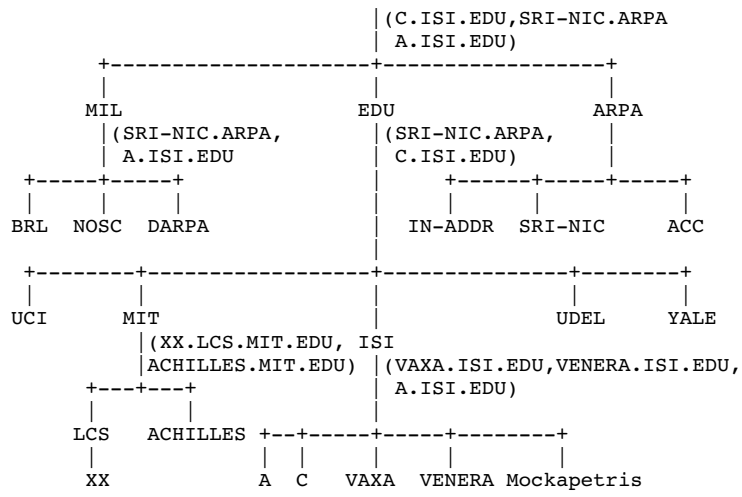
If the response shows a delegation, the resolver should check to see that the delegation is "closer" to the answer than the servers in SLIST are. This can be done by comparing the match count in SLIST with that computed from SNAME and the NS RRs in the delegation. If not, the reply is bogus and should be ignored. If the delegation is valid the NS delegation RRs and any address RRs for the servers should be cached. The name servers are entered in the SLIST, and the search is restarted.

If the response contains a CNAME, the search is restarted at the CNAME unless the response has the data for the canonical name or if the CNAME is the answer itself.

Details and implementation hints can be found in [RFC-1035].

## 6. A SCENARIO

In our sample domain space, suppose we wanted separate administrative control for the root, MIL, EDU, MIT.EDU and ISI.EDU zones. We might allocate name servers as follows:



In this example, the authoritative name server is shown in parentheses at the point in the domain tree at which is assumes control.

Thus the root name servers are on C.ISI.EDU, SRI-NIC.ARPA, and A.ISI.EDU. The MIL domain is served by SRI-NIC.ARPA and A.ISI.EDU. The EDU domain is served by SRI-NIC.ARPA. and C.ISI.EDU. Note that servers may have zones which are contiguous or disjoint. In this scenario, C.ISI.EDU has contiguous zones at the root and EDU domains. A.ISI.EDU has contiguous zones at the root and MIL domains, but also has a non-contiguous zone at ISI.EDU.

6.1. C.ISI.EDU name server

C.ISI.EDU is a name server for the root, MIL, and EDU domains of the IN class, and would have zones for these domains. The zone data for the root domain might be:

```
    .       IN      SOA     SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
                            870611          ;serial
                            1800            ;refresh every 30 min
                            300             ;retry every 5 min
                            604800          ;expire after a week
                            86400)          ;minimum of a day
                    NS      A.ISI.EDU.
                    NS      C.ISI.EDU.
                    NS      SRI-NIC.ARPA.

    MIL.    86400   NS      SRI-NIC.ARPA.
            86400   NS      A.ISI.EDU.

    EDU.    86400   NS      SRI-NIC.ARPA.
            86400   NS      C.ISI.EDU.

    SRI-NIC.ARPA.   A       26.0.0.73
                    A       10.0.0.51
                    MX      0 SRI-NIC.ARPA.
                    HINFO   DEC-2060 TOPS20

    ACC.ARPA.       A       26.6.0.65
                    HINFO   PDP-11/70 UNIX
                    MX      10 ACC.ARPA.

    USC-ISIC.ARPA.  CNAME   C.ISI.EDU.

    73.0.0.26.IN-ADDR.ARPA.  PTR    SRI-NIC.ARPA.
    65.0.6.26.IN-ADDR.ARPA.  PTR    ACC.ARPA.
    51.0.0.10.IN-ADDR.ARPA.  PTR    SRI-NIC.ARPA.
    52.0.0.10.IN-ADDR.ARPA.  PTR    C.ISI.EDU.
```





```
    103.0.3.26.IN-ADDR.ARPA. PTR    A.ISI.EDU.

    A.ISI.EDU. 86400 A      26.3.0.103
    C.ISI.EDU. 86400 A      10.0.0.52
```

This data is represented as it would be in a master file. Most RRs are single line entries; the sole exception here is the SOA RR, which uses "(" to start a multi-line RR and ")" to show the end of a multi-line RR. Since the class of all RRs in a zone must be the same, only the first RR in a zone need specify the class. When a name server loads a zone, it forces the TTL of all authoritative RRs to be at least the MINIMUM field of the SOA, here 86400 seconds, or one day. The NS RRs marking delegation of the MIL and EDU domains, together with the glue RRs for the servers host addresses, are not part of the authoritative data in the zone, and hence have explicit TTLs.

Four RRs are attached to the root node: the SOA which describes the root zone and the 3 NS RRs which list the name servers for the root. The data in the SOA RR describes the management of the zone. The zone data is maintained on host SRI-NIC.ARPA, and the responsible party for the zone is HOSTMASTER@SRI-NIC.ARPA. A key item in the SOA is the 86400 second minimum TTL, which means that all authoritative data in the zone has at least that TTL, although higher values may be explicitly specified.

The NS RRs for the MIL and EDU domains mark the boundary between the root zone and the MIL and EDU zones. Note that in this example, the lower zones happen to be supported by name servers which also support the root zone.

The master file for the EDU zone might be stated relative to the origin EDU. The zone data for the EDU domain might be:

```
    EDU.  IN SOA SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
                            870729 ;serial
                            1800 ;refresh every 30 minutes
                            300 ;retry every 5 minutes
                            604800 ;expire after a week
                            86400 ;minimum of a day
                            )
                    NS SRI-NIC.ARPA.
                    NS C.ISI.EDU.

    UCI 172800 NS ICS.UCI
                    172800 NS ROME.UCI
    ICS.UCI 172800 A 192.5.19.1
    ROME.UCI 172800 A 192.5.19.31
```

```
ISI 172800 NS VAXA.ISI
                172800 NS A.ISI
                172800 NS VENERA.ISI.EDU.
VAXA.ISI 172800 A 10.2.0.27
                172800 A 128.9.0.33
VENERA.ISI.EDU. 172800 A 10.1.0.52
                172800 A 128.9.0.32
A.ISI 172800 A 26.3.0.103

UDEL.EDU.  172800 NS LOUIE.UDEL.EDU.
                172800 NS UMN-REI-UC.ARPA.
LOUIE.UDEL.EDU. 172800 A 10.0.0.96
                172800 A 192.5.39.3

YALE.EDU.  172800 NS YALE.ARPA.
YALE.EDU.  172800 NS YALE-BULLDOG.ARPA.

MIT.EDU.  43200 NS XX.LCS.MIT.EDU.
                 43200 NS ACHILLES.MIT.EDU.
XX.LCS.MIT.EDU.  43200 A 10.0.0.44
ACHILLES.MIT.EDU. 43200 A 18.72.0.8
```

Note the use of relative names here. The owner name for the ISI.EDU. is stated using a relative name, as are two of the name server RR contents. Relative and absolute domain names may be freely intermixed in a master

6.2. Example standard queries

The following queries and responses illustrate name server behavior. Unless otherwise noted, the queries do not have recursion desired (RD) in the header. Note that the answers to non-recursive queries do depend on the server being asked, but do not depend on the identity of the requester.







6.2.1. QNAME=SRI-NIC.ARPA, QTYPE=A

The query would look like:

| | |
|---|---|
| Header | OPCODE=SQUERY |
| Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A |
| Answer | <empty> |
| Authority | <empty> |
| Additional | <empty> |

The response from C.ISI.EDU would be:

| | |
|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA |
| Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A |
| Answer | SRI-NIC.ARPA. 86400 IN A 26.0.0.73<br>86400 IN A 10.0.0.51 |
| Authority | <empty> |
| Additional | <empty> |

The header of the response looks like the header of the query, except that the RESPONSE bit is set, indicating that this message is a response, not a query, and the Authoritative Answer (AA) bit is set indicating that the address RRs in the answer section are from authoritative data. The question section of the response matches the question section of the query.

If the same query was sent to some other server which was not authoritative for SRI-NIC.ARPA, the response might be:

```
            +-------------------------------------------------+
    Header  | OPCODE=SQUERY,RESPONSE                          |
            +-------------------------------------------------+
    Question| QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A         |
            +-------------------------------------------------+
    Answer  | SRI-NIC.ARPA. 1777 IN A 10.0.0.51               |
            |               1777 IN A 26.0.0.73               |
            +-------------------------------------------------+
    Authority | <empty>                                       |
            +-------------------------------------------------+
    Additional | <empty>                                      |
            +-------------------------------------------------+
```

This response is different from the previous one in two ways: the header does not have AA set, and the TTLs are different. The inference is that the data did not come from a zone, but from a cache. The difference between the authoritative TTL and the TTL here is due to aging of the data in a cache. The difference in ordering of the RRs in the answer section is not significant.

6.2.2. QNAME=SRI-NIC.ARPA, QTYPE=*

A query similar to the previous one, but using a QTYPE of *, would receive the following response from C.ISI.EDU:

```
            +-------------------------------------------------+
    Header  | OPCODE=SQUERY, RESPONSE, AA                     |
            +-------------------------------------------------+
    Question| QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=*         |
            +-------------------------------------------------+
    Answer  | SRI-NIC.ARPA. 86400 IN  A     26.0.0.73         |
            |                         A     10.0.0.51         |
            |                         MX    0 SRI-NIC.ARPA.   |
            |                         HINFO DEC-2060 TOPS20   |
            +-------------------------------------------------+
    Authority | <empty>                                       |
            +-------------------------------------------------+
    Additional | <empty>                                      |
            +-------------------------------------------------+
```





If a similar query was directed to two name servers which are not authoritative for SRI-NIC.ARPA, the responses might be:

```
            +-------------------------------------------------+
    Header  | OPCODE=SQUERY, RESPONSE                         |
            +-------------------------------------------------+
    Question| QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=*         |
            +-------------------------------------------------+
    Answer  | SRI-NIC.ARPA. 12345 IN     A       26.0.0.73    |
            |                            A       10.0.0.51    |
            +-------------------------------------------------+
    Authority | <empty>                                       |
            +-------------------------------------------------+
    Additional | <empty>                                      |
            +-------------------------------------------------+
```

and

```
            +-------------------------------------------------+
    Header  | OPCODE=SQUERY, RESPONSE                         |
            +-------------------------------------------------+
    Question| QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=*         |
            +-------------------------------------------------+
    Answer  | SRI-NIC.ARPA. 1290 IN HINFO  DEC-2060 TOPS20    |
            +-------------------------------------------------+
    Authority | <empty>                                       |
            +-------------------------------------------------+
    Additional | <empty>                                      |
            +-------------------------------------------------+
```

Neither of these answers have AA set, so neither response comes from authoritative data. The different contents and different TTLs suggest that the two servers cached data at different times, and that the first server cached the response to a QTYPE=A query and the second cached the response to a HINFO query.

6.2.3. QNAME=SRI-NIC.ARPA, QTYPE=MX

This type of query might be result from a mailer trying to look up routing information for the mail destination HOSTMASTER@SRI-NIC.ARPA. The response from C.ISI.EDU would be:

| | | | | |
|---|---|---|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA | | | |
| Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=MX | | | |
| Answer | SRI-NIC.ARPA. 86400 IN | MX | 0 SRI-NIC.ARPA. | |
| Authority | <empty> | | | |
| Additional | SRI-NIC.ARPA. 86400 IN | A | 26.0.0.73 | |
| | | A | 10.0.0.51 | |

This response contains the MX RR in the answer section of the response. The additional section contains the address RRs because the name server at C.ISI.EDU guesses that the requester will need the addresses in order to properly use the information carried by the MX.

6.2.4. QNAME=SRI-NIC.ARPA, QTYPE=NS

C.ISI.EDU would reply to this query with:

| | |
|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA |
| Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=NS |
| Answer | <empty> |
| Authority | <empty> |
| Additional | <empty> |

The only difference between the response and the query is the AA and RESPONSE bits in the header. The interpretation of this response is that the server is authoritative for the name, and the name exists, but no RRs of type NS are present there.

6.2.5. QNAME=SIR-NIC.ARPA, QTYPE=A

If a user mistyped a host name, we might see this type of query.





C.ISI.EDU would answer it with:

| | |
|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA, RCODE=NE |
| Question | QNAME=SIR-NIC.ARPA., QCLASS=IN, QTYPE=A |
| Answer | <empty> |
| Authority | . SOA SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA.<br>870611 1800 300 604800 86400 |
| Additional | <empty> |

This response states that the name does not exist. This condition is signalled in the response code (RCODE) section of the header.

The SOA RR in the authority section is the optional negative caching information which allows the resolver using this response to assume that the name will not exist for the SOA MINIMUM (86400) seconds.

6.2.6. QNAME=BRL.MIL, QTYPE=A

If this query is sent to C.ISI.EDU, the reply would be:

| | | | | |
|---|---|---|---|---|
| Header | OPCODE=SQUERY, RESPONSE | | | |
| Question | QNAME=BRL.MIL, QCLASS=IN, QTYPE=A | | | |
| Answer | <empty> | | | |
| Authority | MIL. | 86400 IN | NS | SRI-NIC.ARPA. |
| | | 86400 | NS | A.ISI.EDU. |
| Additional | A.ISI.EDU. | | A | 26.3.0.103 |
| | SRI-NIC.ARPA. | | A | 26.0.0.73 |
| | | | A | 10.0.0.51 |

This response has an empty answer section, but is not authoritative, so it is a referral. The name server on C.ISI.EDU, realizing that it is not authoritative for the MIL domain, has referred the requester to servers on A.ISI.EDU and SRI-NIC.ARPA, which it knows are authoritative for the MIL domain.

6.2.7. QNAME=USC-ISIC.ARPA, QTYPE=A

The response to this query from A.ISI.EDU would be:

| | |
|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA |
| Question | QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A |
| Answer | USC-ISIC.ARPA. 86400 IN CNAME C.ISI.EDU.<br>C.ISI.EDU. 86400 IN A 10.0.0.52 |
| Authority | <empty> |
| Additional | <empty> |

Note that the AA bit in the header guarantees that the data matching QNAME is authoritative, but does not say anything about whether the data for C.ISI.EDU is authoritative. This complete reply is possible because A.ISI.EDU happens to be authoritative for both the ARPA domain where USC-ISIC.ARPA is found and the ISI.EDU domain where C.ISI.EDU data is found.

If the same query was sent to C.ISI.EDU, its response might be the same as shown above if it had its own address in its cache, but might also be:





| | | | | |
|---|---|---|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA | | | |
| Question | QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A | | | |
| Answer | USC-ISIC.ARPA. | 86400 IN | CNAME | C.ISI.EDU. |
| Authority | ISI.EDU. | 172800 IN | NS | VAXA.ISI.EDU. |
| | | | NS | A.ISI.EDU. |
| | | | NS | VENERA.ISI.EDU. |
| Additional | VAXA.ISI.EDU. | 172800 | A | 10.2.0.27 |
| | | 172800 | A | 128.9.0.33 |
| | VENERA.ISI.EDU. | 172800 | A | 10.1.0.52 |
| | | 172800 | A | 128.9.0.32 |
| | A.ISI.EDU. | 172800 | A | 26.3.0.103 |

This reply contains an authoritative reply for the alias USC-ISIC.ARPA, plus a referral to the name servers for ISI.EDU. This sort of reply isn't very likely given that the query is for the host name of the name server being asked, but would be common for other aliases.

6.2.8. QNAME=USC-ISIC.ARPA, QTYPE=CNAME

If this query is sent to either A.ISI.EDU or C.ISI.EDU, the reply would be:

| | |
|---|---|
| Header | OPCODE=SQUERY, RESPONSE, AA |
| Question | QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A |
| Answer | USC-ISIC.ARPA. 86400 IN CNAME C.ISI.EDU. |
| Authority | <empty> |
| Additional | <empty> |

Because QTYPE=CNAME, the CNAME RR itself answers the query, and the name server doesn't attempt to look up anything for C.ISI.EDU. (Except possibly for the additional section.)

6.3. Example resolution

The following examples illustrate the operations a resolver must perform for its client. We assume that the resolver is starting without a

cache, as might be the case after system boot. We further assume that the system is not one of the hosts in the data and that the host is located somewhere on net 26, and that its safety belt (SBELT) data structure has the following information:

```
Match count = -1
SRI-NIC.ARPA.   26.0.0.73       10.0.0.51
A.ISI.EDU.      26.3.0.103
```

This information specifies servers to try, their addresses, and a match count of -1, which says that the servers aren't very close to the target. Note that the -1 isn't supposed to be an accurate closeness measure, just a value so that later stages of the algorithm will work.

The following examples illustrate the use of a cache, so each example assumes that previous requests have completed.

6.3.1. Resolve MX for ISI.EDU.

Suppose the first request to the resolver comes from the local mailer, which has mail for PVM@ISI.EDU. The mailer might then ask for type MX RRs for the domain name ISI.EDU.

The resolver would look in its cache for MX RRs at ISI.EDU, but the empty cache wouldn't be helpful. The resolver would recognize that it needed to query foreign servers and try to determine the best servers to query. This search would look for NS RRs for the domains ISI.EDU, EDU, and the root. These searches of the cache would also fail. As a last resort, the resolver would use the information from the SBELT, copying it into its SLIST structure.

At this point the resolver would need to pick one of the three available addresses to try. Given that the resolver is on net 26, it should choose either 26.0.0.73 or 26.3.0.103 as its first choice. It would then send off a query of the form:





| | |
|---|---|
| Header | OPCODE=SQUERY |
| Question | QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX |
| Answer | <empty> |
| Authority | <empty> |
| Additional | <empty> |

The resolver would then wait for a response to its query or a timeout. If the timeout occurs, it would try different servers, then different addresses of the same servers, lastly retrying addresses already tried. It might eventually receive a reply from SRI-NIC.ARPA:

| | | | | | |
|---|---|---|---|---|---|
| Header | OPCODE=SQUERY, RESPONSE | | | | |
| Question | QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX | | | | |
| Answer | <empty> | | | | |
| Authority | ISI.EDU. | 172800 | IN NS | VAXA.ISI.EDU. | |
| | | | NS | A.ISI.EDU. | |
| | | | NS | VENERA.ISI.EDU. | |
| Additional | VAXA.ISI.EDU. | 172800 | A | 10.2.0.27 | |
| | | 172800 | A | 128.9.0.33 | |
| | VENERA.ISI.EDU. | 172800 | A | 10.1.0.52 | |
| | | 172800 | A | 128.9.0.32 | |
| | A.ISI.EDU. | 172800 | A | 26.3.0.103 | |

The resolver would notice that the information in the response gave a closer delegation to ISI.EDU than its existing SLIST (since it matches three labels). The resolver would then cache the information in this response and use it to set up a new SLIST:

```
Match count = 3
A.ISI.EDU.      26.3.0.103
VAXA.ISI.EDU.   10.2.0.27       128.9.0.33
VENERA.ISI.EDU. 10.1.0.52       128.9.0.32
```

A.ISI.EDU appears on this list as well as the previous one, but that is purely coincidental. The resolver would again start transmitting and waiting for responses. Eventually it would get an answer:

```
            +-------------------------------------------------+
    Header  | OPCODE=SQUERY, RESPONSE, AA                     |
            +-------------------------------------------------+
    Question| QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX             |
            +-------------------------------------------------+
    Answer  | ISI.EDU.              MX 10 VENERA.ISI.EDU.     |
            |                       MX 20 VAXA.ISI.EDU.       |
            +-------------------------------------------------+
    Authority | <empty>                                       |
            +-------------------------------------------------+
    Additional | VAXA.ISI.EDU.   172800  A  10.2.0.27         |
            |                   172800  A  128.9.0.33         |
            | VENERA.ISI.EDU. 172800  A  10.1.0.52            |
            |                 172800  A  128.9.0.32           |
            +-------------------------------------------------+
```

The resolver would add this information to its cache, and return the MX RRs to its client.

6.3.2. Get the host name for address 26.6.0.65

The resolver would translate this into a request for PTR RRs for 65.0.6.26.IN-ADDR.ARPA. This information is not in the cache, so the resolver would look for foreign servers to ask. No servers would match, so it would use SBELT again. (Note that the servers for the ISI.EDU domain are in the cache, but ISI.EDU is not an ancestor of 65.0.6.26.IN-ADDR.ARPA, so the SBELT is used.)

Since this request is within the authoritative data of both servers in SBELT, eventually one would return:





```
            +-------------------------------------------------+
    Header  | OPCODE=SQUERY, RESPONSE, AA                     |
            +-------------------------------------------------+
    Question| QNAME=65.0.6.26.IN-ADDR.ARPA.,QCLASS=IN,QTYPE=PTR |
            +-------------------------------------------------+
    Answer  | 65.0.6.26.IN-ADDR.ARPA.    PTR     ACC.ARPA.    |
            +-------------------------------------------------+
    Authority | <empty>                                       |
            +-------------------------------------------------+
    Additional | <empty>                                      |
            +-------------------------------------------------+
```

6.3.3. Get the host address of poneria.ISI.EDU

This request would translate into a type A request for poneria.ISI.EDU. The resolver would not find any cached data for this name, but would find the NS RRs in the cache for ISI.EDU when it looks for foreign servers to ask. Using this data, it would construct a SLIST of the form:

```
    Match count = 3

    A.ISI.EDU.      26.3.0.103
    VAXA.ISI.EDU.   10.2.0.27       128.9.0.33
    VENERA.ISI.EDU. 10.1.0.52
```

A.ISI.EDU is listed first on the assumption that the resolver orders its choices by preference, and A.ISI.EDU is on the same network.

One of these servers would answer the query.

7. REFERENCES and BIBLIOGRAPHY

[Dyer 87] Dyer, S., and F. Hsu, "Hesiod", Project Athena Technical Plan - Name Service, April 1987, version 1.9.

Describes the fundamentals of the Hesiod name service.

[IEN-116] J. Postel, "Internet Name Server", IEN-116, USC/Information Sciences Institute, August 1979.

A name service obsoleted by the Domain Name System, but still in use.

[Quarterman 86] Quarterman, J., and J. Hoskins, "Notable Computer Networks",Communications of the ACM, October 1986, volume 29, number 10.

[RFC-742] K. Harrenstien, "NAME/FINGER", RFC-742, Network Information Center, SRI International, December 1977.

[RFC-768] J. Postel, "User Datagram Protocol", RFC-768, USC/Information Sciences Institute, August 1980.

[RFC-793] J. Postel, "Transmission Control Protocol", RFC-793, USC/Information Sciences Institute, September 1981.

[RFC-799] D. Mills, "Internet Name Domains", RFC-799, COMSAT, September 1981.

Suggests introduction of a hierarchy in place of a flat name space for the Internet.

[RFC-805] J. Postel, "Computer Mail Meeting Notes", RFC-805, USC/Information Sciences Institute, February 1982.

[RFC-810] E. Feinler, K. Harrenstien, Z. Su, and V. White, "DOD Internet Host Table Specification", RFC-810, Network Information Center, SRI International, March 1982.

Obsolete. See RFC-952.

[RFC-811] K. Harrenstien, V. White, and E. Feinler, "Hostnames Server", RFC-811, Network Information Center, SRI International, March 1982.

Obsolete. See RFC-953.

[RFC-812] K. Harrenstien, and V. White, "NICNAME/WHOIS", RFC-812, Network Information Center, SRI International, March 1982.

[RFC-819] Z. Su, and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC-819, Network Information Center, SRI International, August 1982.

Early thoughts on the design of the domain system. Current implementation is completely different.

[RFC-821] J. Postel, "Simple Mail Transfer Protocol", RFC-821, USC/Information Sciences Institute, August 1980.





[RFC-830] Z. Su, "A Distributed System for Internet Name Service", RFC-830, Network Information Center, SRI International, October 1982.

Early thoughts on the design of the domain system. Current implementation is completely different.

[RFC-882] P. Mockapetris, "Domain names - Concepts and Facilities," RFC-882, USC/Information Sciences Institute, November 1983.

Superceeded by this memo.

[RFC-883] P. Mockapetris, "Domain names - Implementation and Specification," RFC-883, USC/Information Sciences Institute, November 1983.

Superceeded by this memo.

[RFC-920] J. Postel and J. Reynolds, "Domain Requirements", RFC-920, USC/Information Sciences Institute October 1984.

Explains the naming scheme for top level domains.

[RFC-952] K. Harrenstien, M. Stahl, E. Feinler, "DoD Internet Host Table Specification", RFC-952, SRI, October 1985.

Specifies the format of HOSTS.TXT, the host/address table replaced by the DNS.

[RFC-953] K. Harrenstien, M. Stahl, E. Feinler, "HOSTNAME Server", RFC-953, SRI, October 1985.

This RFC contains the official specification of the hostname server protocol, which is obsoleted by the DNS. This TCP based protocol accesses information stored in the RFC-952 format, and is used to obtain copies of the host table.

[RFC-973] P. Mockapetris, "Domain System Changes and Observations", RFC-973, USC/Information Sciences Institute, January 1986.

Describes changes to RFC-882 and RFC-883 and reasons for them. Now obsolete.

[RFC-974] C. Partridge, "Mail routing and the domain system", RFC-974, CSNET CIC BBN Labs, January 1986.

Describes the transition from HOSTS.TXT based mail addressing to the more powerful MX system used with the domain system.

[RFC-1001] NetBIOS Working Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and Methods", RFC-1001, March 1987.

This RFC and RFC-1002 are a preliminary design for NETBIOS on top of TCP/IP which proposes to base NetBIOS name service on top of the DNS.

[RFC-1002] NetBIOS Working Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed Specifications", RFC-1002, March 1987.

[RFC-1010] J. Reynolds and J. Postel, "Assigned Numbers", RFC-1010, USC/Information Sciences Institute, May 1987

Contains socket numbers and mnemonics for host names, operating systems, etc.

[RFC-1031] W. Lazear, "MILNET Name Domain Transition", RFC-1031, November 1987.

Describes a plan for converting the MILNET to the DNS.

[RFC-1032] M. K. Stahl, "Establishing a Domain - Guidelines for Administrators", RFC-1032, November 1987.

Describes the registration policies used by the NIC to administer the top level domains and delegate subzones.

[RFC-1033] M. K. Lottor, "Domain Administrators Operations Guide", RFC-1033, November 1987.

A cookbook for domain administrators.

[Solomon 82] M. Solomon, L. Landweber, and D. Neuhengen, "The CSNET Name Server", Computer Networks, vol 6, nr 3, July 1982.

Describes a name service for CSNET which is independent from the DNS and DNS use in the CSNET.







Index

Network Working Group
Request for Comments: 2821
Obsoletes: 821, 974, 1869
Updates: 1123
Category: Standards Track

J. Klensin, Editor
AT&T Laboratories

April 2001

# Simple Mail Transfer Protocol

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## Copyright Notice



## Abstract

This document is a self-contained specification of the basic protocol for the Internet electronic mail transport. It consolidates, updates and clarifies, but doesn't add new or change existing functionality of the following:

- the original SMTP (Simple Mail Transfer Protocol) specification of RFC 821 [30],

- domain name system requirements and implications for mail transport from RFC 1035 [22] and RFC 974 [27],

- the clarifications and applicability statements in RFC 1123 [2], and

- material drawn from the SMTP Extension mechanisms [19].

It obsoletes RFC 821, RFC 974, and updates RFC 1123 (replaces the mail transport materials of RFC 1123). However, RFC 821 specifies some features that were not in significant use in the Internet by the mid-1990s and (in appendices) some additional transport models. Those sections are omitted here in the interest of clarity and brevity; readers needing them should refer to RFC 821.





It also includes some additional material from RFC 1123 that required amplification. This material has been identified in multiple ways, mostly by tracking flaming on various lists and newsgroups and problems of unusual readings or interpretations that have appeared as the SMTP extensions have been deployed. Where this specification moves beyond consolidation and actually differs from earlier documents, it supersedes them technically as well as textually.

Although SMTP was designed as a mail transport and delivery protocol, this specification also contains information that is important to its use as a 'mail submission' protocol, as recommended for POP [3, 26] and IMAP [6]. Additional submission issues are discussed in RFC 2476 [15].

Section 2.3 provides definitions of terms specific to this document. Except when the historical terminology is necessary for clarity, this document uses the current 'client' and 'server' terminology to identify the sending and receiving SMTP processes, respectively.

A companion document [32] discusses message headers, message bodies and formats and structures for them, and their relationship.

## Table of Contents

## 1. Introduction

The objective of the Simple Mail Transfer Protocol (SMTP) is to transfer mail reliably and efficiently.

SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel. While this document specifically discusses transport over TCP, other transports are possible. Appendices to RFC 821 describe some of them.

An important feature of SMTP is its capability to transport mail across networks, usually referred to as "SMTP mail relaying" (see section 3.8). A network consists of the mutually-TCP-accessible hosts on the public Internet, the mutually-TCP-accessible hosts on a firewall-isolated TCP/IP Intranet, or hosts in some other LAN or WAN environment utilizing a non-TCP transport-level protocol. Using

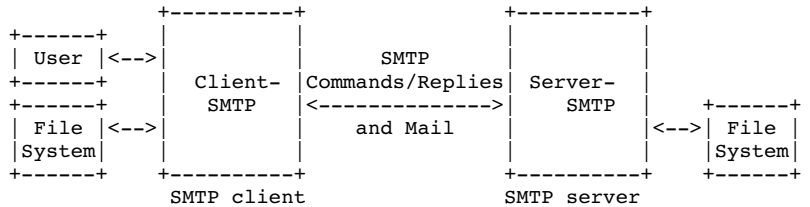
SMTP, a process can transfer mail to another process on the same network or to some other network via a relay or gateway process accessible to both networks.

In this way, a mail message may pass through a number of intermediate relay or gateway hosts on its path from sender to ultimate recipient. The Mail eXchanger mechanisms of the domain name system [22, 27] (and section 5 of this document) are used to identify the appropriate next-hop destination for a message being transported.

## 2. The SMTP Model

### 2.1 Basic Structure

The SMTP design can be pictured as:




When an SMTP client has a message to transmit, it establishes a two-way transmission channel to an SMTP server. The responsibility of an SMTP client is to transfer mail messages to one or more SMTP servers, or report its failure to do so.

The means by which a mail message is presented to an SMTP client, and how that client determines the domain name(s) to which mail messages are to be transferred is a local matter, and is not addressed by this document. In some cases, the domain name(s) transferred to, or determined by, an SMTP client will identify the final destination(s) of the mail message. In other cases, common with SMTP clients associated with implementations of the POP [3, 26] or IMAP [6] protocols, or when the SMTP client is inside an isolated transport service environment, the domain name determined will identify an intermediate destination through which all mail messages are to be relayed. SMTP clients that transfer all traffic, regardless of the target domain names associated with the individual messages, or that do not maintain queues for retrying message transmissions that initially cannot be completed, may otherwise conform to this specification but are not considered fully-capable. Fully-capable SMTP implementations, including the relays used by these less capable





ones, and their destinations, are expected to support all of the queuing, retrying, and alternate address functions discussed in this specification.

The means by which an SMTP client, once it has determined a target domain name, determines the identity of an SMTP server to which a copy of a message is to be transferred, and then performs that transfer, is covered by this document. To effect a mail transfer to an SMTP server, an SMTP client establishes a two-way transmission channel to that SMTP server. An SMTP client determines the address of an appropriate host running an SMTP server by resolving a destination domain name to either an intermediate Mail eXchanger host or a final target host.

An SMTP server may be either the ultimate destination or an intermediate "relay" (that is, it may assume the role of an SMTP client after receiving the message) or "gateway" (that is, it may transport the message further using some protocol other than SMTP). SMTP commands are generated by the SMTP client and sent to the SMTP server. SMTP replies are sent from the SMTP server to the SMTP client in response to the commands.

In other words, message transfer can occur in a single connection between the original SMTP-sender and the final SMTP-recipient, or can occur in a series of hops through intermediary systems. In either case, a formal handoff of responsibility for the message occurs: the protocol requires that a server accept responsibility for either delivering a message or properly reporting the failure to do so.

Once the transmission channel is established and initial handshaking completed, the SMTP client normally initiates a mail transaction. Such a transaction consists of a series of commands to specify the originator and destination of the mail and transmission of the message content (including any headers or other structure) itself. When the same message is sent to multiple recipients, this protocol encourages the transmission of only one copy of the data for all recipients at the same destination (or intermediate relay) host.

The server responds to each command with a reply; replies may indicate that the command was accepted, that additional commands are expected, or that a temporary or permanent error condition exists. Commands specifying the sender or recipients may include server-permitted SMTP service extension requests as discussed in section 2.2. The dialog is purposely lock-step, one-at-a-time, although this can be modified by mutually-agreed extension requests such as command pipelining [13].

Once a given mail message has been transmitted, the client may either request that the connection be shut down or may initiate other mail transactions. In addition, an SMTP client may use a connection to an SMTP server for ancillary services such as verification of email addresses or retrieval of mailing list subscriber addresses.

As suggested above, this protocol provides mechanisms for the transmission of mail. This transmission normally occurs directly from the sending user's host to the receiving user's host when the two hosts are connected to the same transport service. When they are not connected to the same transport service, transmission occurs via one or more relay SMTP servers. An intermediate host that acts as either an SMTP relay or as a gateway into some other transmission environment is usually selected through the use of the domain name service (DNS) Mail eXchanger mechanism.

Usually, intermediate hosts are determined via the DNS MX record, not by explicit "source" routing (see section 5 and appendices C and F.2).

## 2.2 The Extension Model

### 2.2.1 Background

In an effort that started in 1990, approximately a decade after RFC 821 was completed, the protocol was modified with a "service extensions" model that permits the client and server to agree to utilize shared functionality beyond the original SMTP requirements. The SMTP extension mechanism defines a means whereby an extended SMTP client and server may recognize each other, and the server can inform the client as to the service extensions that it supports.

Contemporary SMTP implementations MUST support the basic extension mechanisms. For instance, servers MUST support the EHLO command even if they do not implement any specific extensions and clients SHOULD preferentially utilize EHLO rather than HELO. (However, for compatibility with older conforming implementations, SMTP clients and servers MUST support the original HELO mechanisms as a fallback.) Unless the different characteristics of HELO must be identified for interoperability purposes, this document discusses only EHLO.

SMTP is widely deployed and high-quality implementations have proven to be very robust. However, the Internet community now considers some services to be important that were not anticipated when the protocol was first designed. If support for those services is to be added, it must be done in a way that permits older implementations to continue working acceptably. The extension framework consists of:





- The SMTP command EHLO, superseding the earlier HELO,
- a registry of SMTP service extensions,
- additional parameters to the SMTP MAIL and RCPT commands, and
- optional replacements for commands defined in this protocol, such as for DATA in non-ASCII transmissions [33].

SMTP's strength comes primarily from its simplicity. Experience with many protocols has shown that protocols with few options tend towards ubiquity, whereas protocols with many options tend towards obscurity.

Each and every extension, regardless of its benefits, must be carefully scrutinized with respect to its implementation, deployment, and interoperability costs. In many cases, the cost of extending the SMTP service will likely outweigh the benefit.

### 2.2.2 Definition and Registration of Extensions

The IANA maintains a registry of SMTP service extensions. A corresponding EHLO keyword value is associated with each extension. Each service extension registered with the IANA must be defined in a formal standards-track or IESG-approved experimental protocol document. The definition must include:

- the textual name of the SMTP service extension;
- the EHLO keyword value associated with the extension;
- the syntax and possible values of parameters associated with the EHLO keyword value;
- any additional SMTP verbs associated with the extension (additional verbs will usually be, but are not required to be, the same as the EHLO keyword value);
- any new parameters the extension associates with the MAIL or RCPT verbs;
- a description of how support for the extension affects the behavior of a server and client SMTP; and,
- the increment by which the extension is increasing the maximum length of the commands MAIL and/or RCPT, over that specified in this standard.

In addition, any EHLO keyword value starting with an upper or lower case "X" refers to a local SMTP service extension used exclusively through bilateral agreement. Keywords beginning with "X" MUST NOT be used in a registered service extension. Conversely, keyword values presented in the EHLO response that do not begin with "X" MUST correspond to a standard, standards-track, or IESG-approved experimental SMTP service extension registered with IANA. A conforming server MUST NOT offer non-"X"-prefixed keyword values that are not described in a registered extension.

Additional verbs and parameter names are bound by the same rules as EHLO keywords; specifically, verbs beginning with "X" are local extensions that may not be registered or standardized. Conversely, verbs not beginning with "X" must always be registered.

## 2.3 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 below.

1. MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

2. MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

4. SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which





does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

### 2.3.1 Mail Objects

SMTP transports a mail object. A mail object contains an envelope and content.

The SMTP envelope is sent as a series of SMTP protocol units (described in section 3). It consists of an originator address (to which error reports should be directed); one or more recipient addresses; and optional protocol extension material. Historically, variations on the recipient address specification command (RCPT TO) could be used to specify alternate delivery modes, such as immediate display; those variations have now been deprecated (see appendix F, section F.6).

The SMTP content is sent in the SMTP DATA protocol unit and has two parts: the headers and the body. If the content conforms to other contemporary standards, the headers form a collection of field/value pairs structured as in the message format specification [32]; the body, if structured, is defined according to MIME [12]. The content is textual in nature, expressed using the US-ASCII repertoire [1]. Although SMTP extensions (such as "8BITMIME" [20]) may relax this restriction for the content body, the content headers are always encoded using the US-ASCII repertoire. A MIME extension [23] defines an algorithm for representing header values outside the US-ASCII repertoire, while still encoding them using the US-ASCII repertoire.

### 2.3.2 Senders and Receivers

In RFC 821, the two hosts participating in an SMTP transaction were described as the "SMTP-sender" and "SMTP-receiver". This document has been changed to reflect current industry terminology and hence refers to them as the "SMTP client" (or sometimes just "the client") and "SMTP server" (or just "the server"), respectively. Since a given host may act both as server and client in a relay situation, "receiver" and "sender" terminology is still used where needed for clarity.

### 2.3.3 Mail Agents and Message Stores

Additional mail system terminology became common after RFC 821 was published and, where convenient, is used in this specification. In particular, SMTP servers and clients provide a mail transport service and therefore act as "Mail Transfer Agents" (MTAs). "Mail User Agents" (MUAs or UAs) are normally thought of as the sources and

targets of mail. At the source, an MUA might collect mail to be transmitted from a user and hand it off to an MTA; the final ("delivery") MTA would be thought of as handing the mail off to an MUA (or at least transferring responsibility to it, e.g., by depositing the message in a "message store"). However, while these terms are used with at least the appearance of great precision in other environments, the implied boundaries between MUAs and MTAs often do not accurately match common, and conforming, practices with Internet mail. Hence, the reader should be cautious about inferring the strong relationships and responsibilities that might be implied if these terms were used elsewhere.

### 2.3.4 Host

For the purposes of this specification, a host is a computer system attached to the Internet (or, in some cases, to a private TCP/IP network) and supporting the SMTP protocol. Hosts are known by names (see "domain"); identifying them by numerical address is discouraged.

### 2.3.5 Domain

A domain (or domain name) consists of one or more dot-separated components. These components ("labels" in DNS terminology [22]) are restricted for SMTP purposes to consist of a sequence of letters, digits, and hyphens drawn from the ASCII character set [1]. Domain names are used as names of hosts and of other entities in the domain name hierarchy. For example, a domain may refer to an alias (label of a CNAME RR) or the label of Mail eXchanger records to be used to deliver mail instead of representing a host name. See [22] and section 5 of this specification.

The domain name, as described in this document and in [22], is the entire, fully-qualified name (often referred to as an "FQDN"). A domain name that is not in FQDN form is no more than a local alias. Local aliases MUST NOT appear in any SMTP transaction.

### 2.3.6 Buffer and State Table

SMTP sessions are stateful, with both parties carefully maintaining a common view of the current state. In this document we model this state by a virtual "buffer" and a "state table" on the server which may be used by the client to, for example, "clear the buffer" or "reset the state table," causing the information in the buffer to be discarded and the state to be returned to some previous state.







### 2.3.7 Lines

SMTP commands and, unless altered by a service extension, message data, are transmitted in "lines". Lines consist of zero or more data characters terminated by the sequence ASCII character "CR" (hex value 0D) followed immediately by ASCII character "LF" (hex value 0A). This termination sequence is denoted as <CRLF> in this document. Conforming implementations MUST NOT recognize or generate any other character or character sequence as a line terminator. Limits MAY be imposed on line lengths by servers (see section 4.5.3).

In addition, the appearance of "bare" "CR" or "LF" characters in text (i.e., either without the other) has a long history of causing problems in mail implementations and applications that use the mail system as a tool. SMTP client implementations MUST NOT transmit these characters except when they are intended as line terminators and then MUST, as indicated above, transmit them only as a <CRLF> sequence.

### 2.3.8 Originator, Delivery, Relay, and Gateway Systems

This specification makes a distinction among four types of SMTP systems, based on the role those systems play in transmitting electronic mail. An "originating" system (sometimes called an SMTP originator) introduces mail into the Internet or, more generally, into a transport service environment. A "delivery" SMTP system is one that receives mail from a transport service environment and passes it to a mail user agent or deposits it in a message store which a mail user agent is expected to subsequently access. A "relay" SMTP system (usually referred to just as a "relay") receives mail from an SMTP client and transmits it, without modification to the message data other than adding trace information, to another SMTP server for further relaying or for delivery.

A "gateway" SMTP system (usually referred to just as a "gateway") receives mail from a client system in one transport environment and transmits it to a server system in another transport environment. Differences in protocols or message semantics between the transport environments on either side of a gateway may require that the gateway system perform transformations to the message that are not permitted to SMTP relay systems. For the purposes of this specification, firewalls that rewrite addresses should be considered as gateways, even if SMTP is used on both sides of them (see [11]).

### 2.3.9 Message Content and Mail Data

The terms "message content" and "mail data" are used interchangeably in this document to describe the material transmitted after the DATA command is accepted and before the end of data indication is transmitted. Message content includes message headers and the possibly-structured message body. The MIME specification [12] provides the standard mechanisms for structured message bodies.

### 2.3.10 Mailbox and Address

As used in this specification, an "address" is a character string that identifies a user to whom mail will be sent or a location into which mail will be deposited. The term "mailbox" refers to that depository. The two terms are typically used interchangeably unless the distinction between the location in which mail is placed (the mailbox) and a reference to it (the address) is important. An address normally consists of user and domain specifications. The standard mailbox naming convention is defined to be "local-part@domain": contemporary usage permits a much broader set of applications than simple "user names". Consequently, and due to a long history of problems when intermediate hosts have attempted to optimize transport by modifying them, the local-part MUST be interpreted and assigned semantics only by the host specified in the domain part of the address.

### 2.3.11 Reply

An SMTP reply is an acknowledgment (positive or negative) sent from receiver to sender via the transmission channel in response to a command. The general form of a reply is a numeric completion code (indicating failure or success) usually followed by a text string. The codes are for use by programs and the text is usually intended for human users. Recent work [34] has specified further structuring of the reply strings, including the use of supplemental and more specific completion codes.

## 2.4 General Syntax Principles and Transaction Model

SMTP commands and replies have a rigid syntax. All commands begin with a command verb. All Replies begin with a three digit numeric code. In some commands and replies, arguments MUST follow the verb or reply code. Some commands do not accept arguments (after the verb), and some reply codes are followed, sometimes optionally, by free form text. In both cases, where text appears, it is separated from the verb or reply code by a space character. Complete definitions of commands and replies appear in section 4.





Verbs and argument values (e.g., "TO:" or "to:" in the RCPT command and extension name keywords) are not case sensitive, with the sole exception in this specification of a mailbox local-part (SMTP Extensions may explicitly specify case-sensitive elements). That is, a command verb, an argument value other than a mailbox local-part, and free form text MAY be encoded in upper case, lower case, or any mixture of upper and lower case with no impact on its meaning. This is NOT true of a mailbox local-part. The local-part of a mailbox MUST BE treated as case sensitive. Therefore, SMTP implementations MUST take care to preserve the case of mailbox local-parts. Mailbox domains are not case sensitive. In particular, for some hosts the user "smith" is different from the user "Smith". However, exploiting the case sensitivity of mailbox local-parts impedes interoperability and is discouraged.

A few SMTP servers, in violation of this specification (and RFC 821) require that command verbs be encoded by clients in upper case. Implementations MAY wish to employ this encoding to accommodate those servers.

The argument field consists of a variable length character string ending with the end of the line, i.e., with the character sequence <CRLF>. The receiver will take no action until this sequence is received.

The syntax for each command is shown with the discussion of that command. Common elements and parameters are shown in section 4.1.2.

Commands and replies are composed of characters from the ASCII character set [1]. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero. More specifically, the unextended SMTP service provides seven bit transport only. An originating SMTP client which has not successfully negotiated an appropriate extension with a particular server MUST NOT transmit messages with information in the high-order bit of octets. If such messages are transmitted in violation of this rule, receiving SMTP servers MAY clear the high-order bit or reject the message as invalid. In general, a relay SMTP SHOULD assume that the message content it has received is valid and, assuming that the envelope permits doing so, relay it without inspecting that content. Of course, if the content is mislabeled and the data path cannot accept the actual content, this may result in ultimate delivery of a severely garbled message to the recipient. Delivery SMTP systems MAY reject ("bounce") such messages rather than deliver them. No sending SMTP system is permitted to send envelope commands in any character

set other than US-ASCII; receiving systems SHOULD reject such commands, normally using "500 syntax error - invalid character" replies.

Eight-bit message content transmission MAY be requested of the server by a client using extended SMTP facilities, notably the "8BITMIME" extension [20]. 8BITMIME SHOULD be supported by SMTP servers. However, it MUST not be construed as authorization to transmit unrestricted eight bit material. 8BITMIME MUST NOT be requested by senders for material with the high bit on that is not in MIME format with an appropriate content-transfer encoding; servers MAY reject such messages.

The metalinguistic notation used in this document corresponds to the "Augmented BNF" used in other Internet mail system documents. The reader who is not familiar with that syntax should consult the ABNF specification [8]. Metalanguage terms used in running text are surrounded by pointed brackets (e.g., <CRLF>) for clarity.

## 3. The SMTP Procedures: An Overview

This section contains descriptions of the procedures used in SMTP: session initiation, the mail transaction, forwarding mail, verifying mailbox names and expanding mailing lists, and the opening and closing exchanges. Comments on relaying, a note on mail domains, and a discussion of changing roles are included at the end of this section. Several complete scenarios are presented in appendix D.

### 3.1 Session Initiation

An SMTP session is initiated when a client opens a connection to a server and the server responds with an opening message.

SMTP server implementations MAY include identification of their software and version information in the connection greeting reply after the 220 code, a practice that permits more efficient isolation and repair of any problems. Implementations MAY make provision for SMTP servers to disable the software and version announcement where it causes security concerns. While some systems also identify their contact point for mail problems, this is not a substitute for maintaining the required "postmaster" address (see section 4.5.1).

The SMTP protocol allows a server to formally reject a transaction while still allowing the initial connection as follows: a 554 response MAY be given in the initial connection opening message instead of the 220. A server taking this approach MUST still wait for the client to send a QUIT (see section 4.1.1.10) before closing the connection and SHOULD respond to any intervening commands with





"503 bad sequence of commands". Since an attempt to make an SMTP connection to such a system is probably in error, a server returning a 554 response on connection opening SHOULD provide enough information in the reply text to facilitate debugging of the sending system.

### 3.2 Client Initiation

Once the server has sent the welcoming message and the client has received it, the client normally sends the EHLO command to the server, indicating the client's identity. In addition to opening the session, use of EHLO indicates that the client is able to process service extensions and requests that the server provide a list of the extensions it supports. Older SMTP systems which are unable to support service extensions and contemporary clients which do not require service extensions in the mail session being initiated, MAY use HELO instead of EHLO. Servers MUST NOT return the extended EHLO-style response to a HELO command. For a particular connection attempt, if the server returns a "command not recognized" response to EHLO, the client SHOULD be able to fall back and send HELO.

In the EHLO command the host sending the command identifies itself; the command may be interpreted as saying "Hello, I am <domain>" (and, in the case of EHLO, "and I support service extension requests").

### 3.3 Mail Transactions

There are three steps to SMTP mail transactions. The transaction starts with a MAIL command which gives the sender identification. (In general, the MAIL command may be sent only when no mail transaction is in progress; see section 4.1.4.) A series of one or more RCPT commands follows giving the receiver information. Then a DATA command initiates transfer of the mail data and is terminated by the "end of mail" data indicator, which also confirms the transaction.

The first step in the procedure is the MAIL command.

MAIL FROM:<reverse-path> [SP <mail-parameters> ] <CRLF>

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. The <reverse-path> portion of the first or only argument contains the source mailbox (between "<" and ">" brackets), which can be used to report errors (see section 4.2 for a discussion of error reporting). If accepted, the SMTP server returns a 250 OK reply. If the mailbox specification is not acceptable for some reason, the server MUST return a reply indicating whether the

failure is permanent (i.e., will occur again if the client tries to send the same address again) or temporary (i.e., the address might be accepted if the client tries again later). Despite the apparent scope of this requirement, there are circumstances in which the acceptability of the reverse-path may not be determined until one or more forward-paths (in RCPT commands) can be examined. In those cases, the server MAY reasonably accept the reverse-path (with a 250 reply) and then report problems after the forward-paths are received and examined. Normally, failures produce 550 or 553 replies.

Historically, the <reverse-path> can contain more than just a mailbox, however, contemporary systems SHOULD NOT use source routing (see appendix C).

The optional <mail-parameters> are associated with negotiated SMTP service extensions (see section 2.2).

The second step in the procedure is the RCPT command.

```
RCPT TO:<forward-path> [ SP <rcpt-parameters> ] <CRLF>
```

The first or only argument to this command includes a forward-path (normally a mailbox and domain, always surrounded by "<" and ">" brackets) identifying one recipient. If accepted, the SMTP server returns a 250 OK reply and stores the forward-path. If the recipient is known not to be a deliverable address, the SMTP server returns a 550 reply, typically with a string such as "no such user - " and the mailbox name (other circumstances and reply codes are possible). This step of the procedure can be repeated any number of times.

The <forward-path> can contain more than just a mailbox. Historically, the <forward-path> can be a source routing list of hosts and the destination mailbox, however, contemporary SMTP clients SHOULD NOT utilize source routes (see appendix C). Servers MUST be prepared to encounter a list of source routes in the forward path, but SHOULD ignore the routes or MAY decline to support the relaying they imply. Similarly, servers MAY decline to accept mail that is destined for other hosts or systems. These restrictions make a server useless as a relay for clients that do not support full SMTP functionality. Consequently, restricted-capability clients MUST NOT assume that any SMTP server on the Internet can be used as their mail processing (relaying) site. If a RCPT command appears without a previous MAIL command, the server MUST return a 503 "Bad sequence of commands" response. The optional <rcpt-parameters> are associated with negotiated SMTP service extensions (see section 2.2).

The third step in the procedure is the DATA command (or some alternative specified in a service extension).







```
DATA <CRLF>
```

If accepted, the SMTP server returns a 354 Intermediate reply and considers all succeeding lines up to but not including the end of mail data indicator to be the message text. When the end of text is successfully received and stored the SMTP-receiver sends a 250 OK reply.

Since the mail data is sent on the transmission channel, the end of mail data must be indicated so that the command and reply dialog can be resumed. SMTP indicates the end of the mail data by sending a line containing only a "." (period or full stop). A transparency procedure is used to prevent this from interfering with the user's text (see section 4.5.2).

The end of mail data indicator also confirms the mail transaction and tells the SMTP server to now process the stored recipients and mail data. If accepted, the SMTP server returns a 250 OK reply. The DATA command can fail at only two points in the protocol exchange:

- If there was no MAIL, or no RCPT, command, or all such commands were rejected, the server MAY return a "command out of sequence" (503) or "no valid recipients" (554) reply in response to the DATA command. If one of those replies (or any other 5yz reply) is received, the client MUST NOT send the message data; more generally, message data MUST NOT be sent unless a 354 reply is received.

- If the verb is initially accepted and the 354 reply issued, the DATA command should fail only if the mail transaction was incomplete (for example, no recipients), or if resources were unavailable (including, of course, the server unexpectedly becoming unavailable), or if the server determines that the message should be rejected for policy or other reasons.

However, in practice, some servers do not perform recipient verification until after the message text is received. These servers SHOULD treat a failure for one or more recipients as a "subsequent failure" and return a mail message as discussed in section 6. Using a "550 mailbox not found" (or equivalent) reply code after the data are accepted makes it difficult or impossible for the client to determine which recipients failed.

When RFC 822 format [7, 32] is being used, the mail data include the memo header items such as Date, Subject, To, Cc, From. Server SMTP systems SHOULD NOT reject messages based on perceived defects in the RFC 822 or MIME [12] message header or message body. In particular,

they MUST NOT reject messages in which the numbers of Resent-fields do not match or Resent-to appears without Resent-from and/or Resent-date.

Mail transaction commands MUST be used in the order discussed above.

## 3.4 Forwarding for Address Correction or Updating

Forwarding support is most often required to consolidate and simplify addresses within, or relative to, some enterprise and less frequently to establish addresses to link a person's prior address with current one. Silent forwarding of messages (without server notification to the sender), for security or non-disclosure purposes, is common in the contemporary Internet.

In both the enterprise and the "new address" cases, information hiding (and sometimes security) considerations argue against exposure of the "final" address through the SMTP protocol as a side-effect of the forwarding activity. This may be especially important when the final address may not even be reachable by the sender. Consequently, the "forwarding" mechanisms described in section 3.2 of RFC 821, and especially the 251 (corrected destination) and 551 reply codes from RCPT must be evaluated carefully by implementers and, when they are available, by those configuring systems.

In particular:

* Servers MAY forward messages when they are aware of an address change. When they do so, they MAY either provide address-updating information with a 251 code, or may forward "silently" and return a 250 code. But, if a 251 code is used, they MUST NOT assume that the client will actually update address information or even return that information to the user.

Alternately,

* Servers MAY reject or bounce messages when they are not deliverable when addressed. When they do so, they MAY either provide address-updating information with a 551 code, or may reject the message as undeliverable with a 550 code and no address-specific information. But, if a 551 code is used, they MUST NOT assume that the client will actually update address information or even return that information to the user.

SMTP server implementations that support the 251 and/or 551 reply codes are strongly encouraged to provide configuration mechanisms so that sites which conclude that they would undesirably disclose information can disable or restrict their use.







## 3.5 Commands for Debugging Addresses

### 3.5.1 Overview

SMTP provides commands to verify a user name or obtain the content of a mailing list. This is done with the VRFY and EXPN commands, which have character string arguments. Implementations SHOULD support VRFY and EXPN (however, see section 3.5.2 and 7.3).

For the VRFY command, the string is a user name or a user name and domain (see below). If a normal (i.e., 250) response is returned, the response MAY include the full name of the user and MUST include the mailbox of the user. It MUST be in either of the following forms:

```
User Name <local-part@domain>
local-part@domain
```

When a name that is the argument to VRFY could identify more than one mailbox, the server MAY either note the ambiguity or identify the alternatives. In other words, any of the following are legitimate response to VRFY:

```
553 User ambiguous
```

or

```
553- Ambiguous;  Possibilities are
553-Joe Smith <jsmith@foo.com>
553-Harry Smith <hsmith@foo.com>
553 Melvin Smith <dweep@foo.com>
```

or

```
553-Ambiguous;  Possibilities
553- <jsmith@foo.com>
553- <hsmith@foo.com>
553 <dweep@foo.com>
```

Under normal circumstances, a client receiving a 553 reply would be expected to expose the result to the user. Use of exactly the forms given, and the "user ambiguous" or "ambiguous" keywords, possibly supplemented by extended reply codes such as those described in [34], will facilitate automated translation into other languages as needed. Of course, a client that was highly automated or that was operating in another language than English, might choose to try to translate the response, to return some other indication to the user than the

literal text of the reply, or to take some automated action such as consulting a directory service for additional information before reporting to the user.

For the EXPN command, the string identifies a mailing list, and the successful (i.e., 250) multiline response MAY include the full name of the users and MUST give the mailboxes on the mailing list.

In some hosts the distinction between a mailing list and an alias for a single mailbox is a bit fuzzy, since a common data structure may hold both types of entries, and it is possible to have mailing lists containing only one mailbox. If a request is made to apply VRFY to a mailing list, a positive response MAY be given if a message so addressed would be delivered to everyone on the list, otherwise an error SHOULD be reported (e.g., "550 That is a mailing list, not a user" or "252 Unable to verify members of mailing list"). If a request is made to expand a user name, the server MAY return a positive response consisting of a list containing one name, or an error MAY be reported (e.g., "550 That is a user name, not a mailing list").

In the case of a successful multiline reply (normal for EXPN) exactly one mailbox is to be specified on each line of the reply. The case of an ambiguous request is discussed above.

"User name" is a fuzzy term and has been used deliberately. An implementation of the VRFY or EXPN commands MUST include at least recognition of local mailboxes as "user names". However, since current Internet practice often results in a single host handling mail for multiple domains, hosts, especially hosts that provide this functionality, SHOULD accept the "local-part@domain" form as a "user name"; hosts MAY also choose to recognize other strings as "user names".

The case of expanding a mailbox list requires a multiline reply, such as:

```
C: EXPN Example-People
S: 250-Jon Postel <Postel@isi.edu>
S: 250-Fred Fonebone <Fonebone@physics.foo-u.edu>
S: 250 Sam Q. Smith <SQSmith@specific.generic.com>
```

or

```
C: EXPN Executive-Washroom-List
S: 550 Access Denied to You.
```







The character string arguments of the VRFY and EXPN commands cannot be further restricted due to the variety of implementations of the user name and mailbox list concepts. On some systems it may be appropriate for the argument of the EXPN command to be a file name for a file containing a mailing list, but again there are a variety of file naming conventions in the Internet. Similarly, historical variations in what is returned by these commands are such that the response SHOULD be interpreted very carefully, if at all, and SHOULD generally only be used for diagnostic purposes.

### 3.5.2 VRFY Normal Response

When normal (2yz or 551) responses are returned from a VRFY or EXPN request, the reply normally includes the mailbox name, i.e., "<local-part@domain>", where "domain" is a fully qualified domain name, MUST appear in the syntax. In circumstances exceptional enough to justify violating the intent of this specification, free-form text MAY be returned. In order to facilitate parsing by both computers and people, addresses SHOULD appear in pointed brackets. When addresses, rather than free-form debugging information, are returned, EXPN and VRFY MUST return only valid domain addresses that are usable in SMTP RCPT commands. Consequently, if an address implies delivery to a program or other system, the mailbox name used to reach that target MUST be given. Paths (explicit source routes) MUST NOT be returned by VRFY or EXPN.

Server implementations SHOULD support both VRFY and EXPN. For security reasons, implementations MAY provide local installations a way to disable either or both of these commands through configuration options or the equivalent. When these commands are supported, they are not required to work across relays when relaying is supported. Since they were both optional in RFC 821, they MUST be listed as service extensions in an EHLO response, if they are supported.

### 3.5.3 Meaning of VRFY or EXPN Success Response

A server MUST NOT return a 250 code in response to a VRFY or EXPN command unless it has actually verified the address. In particular, a server MUST NOT return 250 if all it has done is to verify that the syntax given is valid. In that case, 502 (Command not implemented) or 500 (Syntax error, command unrecognized) SHOULD be returned. As stated elsewhere, implementation (in the sense of actually validating addresses and returning information) of VRFY and EXPN are strongly recommended. Hence, implementations that return 500 or 502 for VRFY are not in full compliance with this specification.

There may be circumstances where an address appears to be valid but cannot reasonably be verified in real time, particularly when a server is acting as a mail exchanger for another server or domain. "Apparent validity" in this case would normally involve at least syntax checking and might involve verification that any domains specified were ones to which the host expected to be able to relay mail. In these situations, reply code 252 SHOULD be returned. These cases parallel the discussion of RCPT verification discussed in section 2.1. Similarly, the discussion in section 3.4 applies to the use of reply codes 251 and 551 with VRFY (and EXPN) to indicate addresses that are recognized but that would be forwarded or bounced were mail received for them. Implementations generally SHOULD be more aggressive about address verification in the case of VRFY than in the case of RCPT, even if it takes a little longer to do so.

### 3.5.4 Semantics and Applications of EXPN

EXPN is often very useful in debugging and understanding problems with mailing lists and multiple-target-address aliases. Some systems have attempted to use source expansion of mailing lists as a means of eliminating duplicates. The propagation of aliasing systems with mail on the Internet, for hosts (typically with MX and CNAME DNS records), for mailboxes (various types of local host aliases), and in various proxying arrangements, has made it nearly impossible for these strategies to work consistently, and mail systems SHOULD NOT attempt them.

## 3.6 Domains

Only resolvable, fully-qualified, domain names (FQDNs) are permitted when domain names are used in SMTP. In other words, names that can be resolved to MX RRs or A RRs (as discussed in section 5) are permitted, as are CNAME RRs whose targets can be resolved, in turn, to MX or A RRs. Local nicknames or unqualified names MUST NOT be used. There are two exceptions to the rule requiring FQDNs:

- The domain name given in the EHLO command MUST BE either a primary host name (a domain name that resolves to an A RR) or, if the host has no name, an address literal as described in section 4.1.1.1.

- The reserved mailbox name "postmaster" may be used in a RCPT command without domain qualification (see section 4.1.1.3) and MUST be accepted if so used.





## 3.7 Relaying

In general, the availability of Mail eXchanger records in the domain name system [22, 27] makes the use of explicit source routes in the Internet mail system unnecessary. Many historical problems with their interpretation have made their use undesirable. SMTP clients SHOULD NOT generate explicit source routes except under unusual circumstances. SMTP servers MAY decline to act as mail relays or to accept addresses that specify source routes. When route information is encountered, SMTP servers are also permitted to ignore the route information and simply send to the final destination specified as the last element in the route and SHOULD do so. There has been an invalid practice of using names that do not appear in the DNS as destination names, with the senders counting on the intermediate hosts specified in source routing to resolve any problems. If source routes are stripped, this practice will cause failures. This is one of several reasons why SMTP clients MUST NOT generate invalid source routes or depend on serial resolution of names.

When source routes are not used, the process described in RFC 821 for constructing a reverse-path from the forward-path is not applicable and the reverse-path at the time of delivery will simply be the address that appeared in the MAIL command.

A relay SMTP server is usually the target of a DNS MX record that designates it, rather than the final delivery system. The relay server may accept or reject the task of relaying the mail in the same way it accepts or rejects mail for a local user. If it accepts the task, it then becomes an SMTP client, establishes a transmission channel to the next SMTP server specified in the DNS (according to the rules in section 5), and sends it the mail. If it declines to relay mail to a particular address for policy reasons, a 550 response SHOULD be returned.

Many mail-sending clients exist, especially in conjunction with facilities that receive mail via POP3 or IMAP, that have limited capability to support some of the requirements of this specification, such as the ability to queue messages for subsequent delivery attempts. For these clients, it is common practice to make private arrangements to send all messages to a single server for processing and subsequent distribution. SMTP, as specified here, is not ideally suited for this role, and work is underway on standardized mail submission protocols that might eventually supercede the current practices. In any event, because these arrangements are private and fall outside the scope of this specification, they are not described here.

It is important to note that MX records can point to SMTP servers which act as gateways into other environments, not just SMTP relays and final delivery systems; see sections 3.8 and 5.

If an SMTP server has accepted the task of relaying the mail and later finds that the destination is incorrect or that the mail cannot be delivered for some other reason, then it MUST construct an "undeliverable mail" notification message and send it to the originator of the undeliverable mail (as indicated by the reverse-path). Formats specified for non-delivery reports by other standards (see, for example, [24, 25]) SHOULD be used if possible.

This notification message must be from the SMTP server at the relay host or the host that first determines that delivery cannot be accomplished. Of course, SMTP servers MUST NOT send notification messages about problems transporting notification messages. One way to prevent loops in error reporting is to specify a null reverse-path in the MAIL command of a notification message. When such a message is transmitted the reverse-path MUST be set to null (see section 4.5.5 for additional discussion). A MAIL command with a null reverse-path appears as follows:

```
MAIL FROM:<>
```

As discussed in section 2.4.1, a relay SMTP has no need to inspect or act upon the headers or body of the message data and MUST NOT do so except to add its own "Received:" header (section 4.4) and, optionally, to attempt to detect looping in the mail system (see section 6.2).

## 3.8 Mail Gatewaying

While the relay function discussed above operates within the Internet SMTP transport service environment, MX records or various forms of explicit routing may require that an intermediate SMTP server perform a translation function between one transport service and another. As discussed in section 2.3.8, when such a system is at the boundary between two transport service environments, we refer to it as a "gateway" or "gateway SMTP".

Gatewaying mail between different mail environments, such as different mail formats and protocols, is complex and does not easily yield to standardization. However, some general requirements may be given for a gateway between the Internet and another mail environment.





### 3.8.1 Header Fields in Gatewaying

Header fields MAY be rewritten when necessary as messages are gatewayed across mail environment boundaries. This may involve inspecting the message body or interpreting the local-part of the destination address in spite of the prohibitions in section 2.4.1.

Other mail systems gatewayed to the Internet often use a subset of RFC 822 headers or provide similar functionality with a different syntax, but some of these mail systems do not have an equivalent to the SMTP envelope. Therefore, when a message leaves the Internet environment, it may be necessary to fold the SMTP envelope information into the message header. A possible solution would be to create new header fields to carry the envelope information (e.g., "X-SMTP-MAIL:" and "X-SMTP-RCPT:"); however, this would require changes in mail programs in foreign environments and might risk disclosure of private information (see section 7.2).

### 3.8.2 Received Lines in Gatewaying

When forwarding a message into or out of the Internet environment, a gateway MUST prepend a Received: line, but it MUST NOT alter in any way a Received: line that is already in the header.

"Received:" fields of messages originating from other environments may not conform exactly to this specification. However, the most important use of Received: lines is for debugging mail faults, and this debugging can be severely hampered by well-meaning gateways that try to "fix" a Received: line. As another consequence of trace fields arising in non-SMTP environments, receiving systems MUST NOT reject mail based on the format of a trace field and SHOULD be extremely robust in the light of unexpected information or formats in those fields.

The gateway SHOULD indicate the environment and protocol in the "via" clauses of Received field(s) that it supplies.

### 3.8.3 Addresses in Gatewaying

From the Internet side, the gateway SHOULD accept all valid address formats in SMTP commands and in RFC 822 headers, and all valid RFC 822 messages. Addresses and headers generated by gateways MUST conform to applicable Internet standards (including this one and RFC 822). Gateways are, of course, subject to the same rules for handling source routes as those described for other SMTP systems in section 3.3.

### 3.8.4 Other Header Fields in Gatewaying

The gateway MUST ensure that all header fields of a message that it forwards into the Internet mail environment meet the requirements for Internet mail. In particular, all addresses in "From:", "To:", "Cc:", etc., fields MUST be transformed (if necessary) to satisfy RFC 822 syntax, MUST reference only fully-qualified domain names, and MUST be effective and useful for sending replies. The translation algorithm used to convert mail from the Internet protocols to another environment's protocol SHOULD ensure that error messages from the foreign mail environment are delivered to the return path from the SMTP envelope, not to the sender listed in the "From:" field (or other fields) of the RFC 822 message.

### 3.8.5 Envelopes in Gatewaying

Similarly, when forwarding a message from another environment into the Internet, the gateway SHOULD set the envelope return path in accordance with an error message return address, if supplied by the foreign environment. If the foreign environment has no equivalent concept, the gateway must select and use a best approximation, with the message originator's address as the default of last resort.

## 3.9 Terminating Sessions and Connections

An SMTP connection is terminated when the client sends a QUIT command. The server responds with a positive reply code, after which it closes the connection.

An SMTP server MUST NOT intentionally close the connection except:

- After receiving a QUIT command and responding with a 221 reply.
- After detecting the need to shut down the SMTP service and returning a 421 response code. This response code can be issued after the server receives any command or, if necessary, asynchronously from command receipt (on the assumption that the client will receive it after the next command is issued).

In particular, a server that closes connections in response to commands that are not understood is in violation of this specification. Servers are expected to be tolerant of unknown commands, issuing a 500 reply and awaiting further instructions from the client.







An SMTP server which is forcibly shut down via external means SHOULD attempt to send a line containing a 421 response code to the SMTP client before exiting. The SMTP client will normally read the 421 response code after sending its next command.

SMTP clients that experience a connection close, reset, or other communications failure due to circumstances not under their control (in violation of the intent of this specification but sometimes unavoidable) SHOULD, to maintain the robustness of the mail system, treat the mail transaction as if a 451 response had been received and act accordingly.

## 3.10 Mailing Lists and Aliases

An SMTP-capable host SHOULD support both the alias and the list models of address expansion for multiple delivery. When a message is delivered or forwarded to each address of an expanded list form, the return address in the envelope ("MAIL FROM:") MUST be changed to be the address of a person or other entity who administers the list. However, in this case, the message header [32] MUST be left unchanged; in particular, the "From" field of the message header is unaffected.

An important mail facility is a mechanism for multi-destination delivery of a single message, by transforming (or "expanding" or "exploding") a pseudo-mailbox address into a list of destination mailbox addresses. When a message is sent to such a pseudo-mailbox (sometimes called an "exploder"), copies are forwarded or redistributed to each mailbox in the expanded list. Servers SHOULD simply utilize the addresses on the list; application of heuristics or other matching rules to eliminate some addresses, such as that of the originator, is strongly discouraged. We classify such a pseudo-mailbox as an "alias" or a "list", depending upon the expansion rules.

### 3.10.1 Alias

To expand an alias, the recipient mailer simply replaces the pseudo-mailbox address in the envelope with each of the expanded addresses in turn; the rest of the envelope and the message body are left unchanged. The message is then delivered or forwarded to each expanded address.

### 3.10.2 List

A mailing list may be said to operate by "redistribution" rather than by "forwarding". To expand a list, the recipient mailer replaces the pseudo-mailbox address in the envelope with all of the expanded

addresses. The return address in the envelope is changed so that all error messages generated by the final deliveries will be returned to a list administrator, not to the message originator, who generally has no control over the contents of the list and will typically find error messages annoying.

# 4. The SMTP Specifications

## 4.1 SMTP Commands

### 4.1.1 Command Semantics and Syntax

The SMTP commands define the mail transfer or the mail system function requested by the user. SMTP commands are character strings terminated by <CRLF>. The commands themselves are alphabetic characters terminated by <SP> if parameters follow and <CRLF> otherwise. (In the interest of improved interoperability, SMTP receivers are encouraged to tolerate trailing white space before the terminating <CRLF>.) The syntax of the local part of a mailbox must conform to receiver site conventions and the syntax specified in section 4.1.2. The SMTP commands are discussed below. The SMTP replies are discussed in section 4.2.

A mail transaction involves several data objects which are communicated as arguments to different commands. The reverse-path is the argument of the MAIL command, the forward-path is the argument of the RCPT command, and the mail data is the argument of the DATA command. These arguments or data objects must be transmitted and held pending the confirmation communicated by the end of mail data indication which finalizes the transaction. The model for this is that distinct buffers are provided to hold the types of data objects, that is, there is a reverse-path buffer, a forward-path buffer, and a mail data buffer. Specific commands cause information to be appended to a specific buffer, or cause one or more buffers to be cleared.

Several commands (RSET, DATA, QUIT) are specified as not permitting parameters. In the absence of specific extensions offered by the server and accepted by the client, clients MUST NOT send such parameters and servers SHOULD reject commands containing them as having invalid syntax.

#### 4.1.1.1 Extended HELLO (EHLO) or HELLO (HELO)

These commands are used to identify the SMTP client to the SMTP server. The argument field contains the fully-qualified domain name of the SMTP client if one is available. In situations in which the SMTP client system does not have a meaningful domain name (e.g., when its address is dynamically allocated and no reverse mapping record is







available), the client SHOULD send an address literal (see section 4.1.3), optionally followed by information that will help to identify the client system. y The SMTP server identifies itself to the SMTP client in the connection greeting reply and in the response to this command.

A client SMTP SHOULD start an SMTP session by issuing the EHLO command. If the SMTP server supports the SMTP service extensions it will give a successful response, a failure response, or an error response. If the SMTP server, in violation of this specification, does not support any SMTP service extensions it will generate an error response. Older client SMTP systems MAY, as discussed above, use HELO (as specified in RFC 821) instead of EHLO, and servers MUST support the HELO command and reply properly to it. In any event, a client MUST issue HELO or EHLO before starting a mail transaction.

These commands, and a "250 OK" reply to one of them, confirm that both the SMTP client and the SMTP server are in the initial state, that is, there is no transaction in progress and all state tables and buffers are cleared.

Syntax:

```
ehlo            = "EHLO" SP Domain CRLF
helo            = "HELO" SP Domain CRLF
```

Normally, the response to EHLO will be a multiline reply. Each line of the response contains a keyword and, optionally, one or more parameters. Following the normal syntax for multiline replies, these keyworks follow the code (250) and a hyphen for all but the last line, and the code and a space for the last line. The syntax for a positive response, using the ABNF notation and terminal symbols of [8], is:

```
ehlo-ok-rsp  =    ( "250"    domain [ SP ehlo-greet ] CRLF )
              / (    "250-"   domain [ SP ehlo-greet ] CRLF
                  *( "250-"   ehlo-line                CRLF )
                     "250"    SP ehlo-line             CRLF  )

ehlo-greet   = 1*(%d0-9 / %d11-12 / %d14-127)
             ; string of any characters other than CR or LF

ehlo-line    = ehlo-keyword *( SP ehlo-param )

ehlo-keyword = (ALPHA / DIGIT) *(ALPHA / DIGIT / "-")
             ; additional syntax of ehlo-params depends on
             ; ehlo-keyword
```

```
ehlo-param   = 1*(%d33-127)
               ; any CHAR excluding <SP> and all
               ; control characters (US-ASCII 0-31 inclusive)
```

Although EHLO keywords may be specified in upper, lower, or mixed case, they MUST always be recognized and processed in a case-insensitive manner. This is simply an extension of practices specified in RFC 821 and section 2.4.1.

4.1.1.2 MAIL (MAIL)

This command is used to initiate a mail transaction in which the mail data is delivered to an SMTP server which may, in turn, deliver it to one or more mailboxes or pass it on to another system (possibly using SMTP). The argument field contains a reverse-path and may contain optional parameters. In general, the MAIL command may be sent only when no mail transaction is in progress, see section 4.1.4.

The reverse-path consists of the sender mailbox. Historically, that mailbox might optionally have been preceded by a list of hosts, but that behavior is now deprecated (see appendix C). In some types of reporting messages for which a reply is likely to cause a mail loop (for example, mail delivery and nondelivery notifications), the reverse-path may be null (see section 3.7).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

If service extensions were negotiated, the MAIL command may also carry parameters associated with a particular service extension.

Syntax:

```
"MAIL FROM:" ("<>" / Reverse-Path)
                [SP Mail-parameters] CRLF
```

4.1.1.3 RECIPIENT (RCPT)

This command is used to identify an individual recipient of the mail data; multiple recipients are specified by multiple use of this command. The argument field contains a forward-path and may contain optional parameters.

The forward-path normally consists of the required destination mailbox. Sending systems SHOULD not generate the optional list of hosts known as a source route. Receiving systems MUST recognize





source route syntax but SHOULD strip off the source route specification and utilize the domain name associated with the mailbox as if the source route had not been provided.

Similarly, relay hosts SHOULD strip or ignore source routes, and names MUST NOT be copied into the reverse-path. When mail reaches its ultimate destination (the forward-path contains only a destination mailbox), the SMTP server inserts it into the destination mailbox in accordance with its host mail conventions.

For example, mail received at relay host xyz.com with envelope commands

```
MAIL FROM:<userx@y.foo.org>
RCPT TO:<@hosta.int,@jkl.org:userc@d.bar.org>
```

will normally be sent directly on to host d.bar.org with envelope commands

```
MAIL FROM:<userx@y.foo.org>
RCPT TO:<userc@d.bar.org>
```

As provided in appendix C, xyz.com MAY also choose to relay the message to hosta.int, using the envelope commands

```
MAIL FROM:<userx@y.foo.org>
RCPT TO:<@hosta.int,@jkl.org:userc@d.bar.org>
```

or to jkl.org, using the envelope commands

```
MAIL FROM:<userx@y.foo.org>
RCPT TO:<@jkl.org:userc@d.bar.org>
```

Of course, since hosts are not required to relay mail at all, xyz.com may also reject the message entirely when the RCPT command is received, using a 550 code (since this is a "policy reason").

If service extensions were negotiated, the RCPT command may also carry parameters associated with a particular service extension offered by the server. The client MUST NOT transmit parameters other than those associated with a service extension offered by the server in its EHLO response.

Syntax:

```
"RCPT TO:" ("<Postmaster@" domain ">" / "<Postmaster>" / Forward-Path)
                [SP Rcpt-parameters] CRLF
```

4.1.1.4 DATA (DATA)

The receiver normally sends a 354 response to DATA, and then treats the lines (strings ending in <CRLF> sequences, as described in section 2.3.7) following the command as mail data from the sender. This command causes the mail data to be appended to the mail data buffer. The mail data may contain any of the 128 ASCII character codes, although experience has indicated that use of control characters other than SP, HT, CR, and LF may cause problems and SHOULD be avoided when possible.

The mail data is terminated by a line containing only a period, that is, the character sequence "<CRLF>.<CRLF>" (see section 4.5.2). This is the end of mail data indication. Note that the first <CRLF> of this terminating sequence is also the <CRLF> that ends the final line of the data (message text) or, if there was no data, ends the DATA command itself. An extra <CRLF> MUST NOT be added, as that would cause an empty line to be added to the message. The only exception to this rule would arise if the message body were passed to the originating SMTP-sender with a final "line" that did not end in <CRLF>; in that case, the originating SMTP system MUST either reject the message as invalid or add <CRLF> in order to have the receiving SMTP server recognize the "end of data" condition.

The custom of accepting lines ending only in <LF>, as a concession to non-conforming behavior on the part of some UNIX systems, has proven to cause more interoperability problems than it solves, and SMTP server systems MUST NOT do this, even in the name of improved robustness. In particular, the sequence "<LF>.<LF>" (bare line feeds, without carriage returns) MUST NOT be treated as equivalent to <CRLF>.<CRLF> as the end of mail data indication.

Receipt of the end of mail data indication requires the server to process the stored mail transaction information. This processing consumes the information in the reverse-path buffer, the forward-path buffer, and the mail data buffer, and on the completion of this command these buffers are cleared. If the processing is successful, the receiver MUST send an OK reply. If the processing fails the receiver MUST send a failure reply. The SMTP model does not allow for partial failures at this point: either the message is accepted by the server for delivery and a positive response is returned or it is not accepted and a failure reply is returned. In sending a positive completion reply to the end of data indication, the receiver takes full responsibility for the message (see section 6.1). Errors that are diagnosed subsequently MUST be reported in a mail message, as discussed in section 4.4.







When the SMTP server accepts a message either for relaying or for final delivery, it inserts a trace record (also referred to interchangeably as a "time stamp line" or "Received" line) at the top of the mail data. This trace record indicates the identity of the host that sent the message, the identity of the host that received the message (and is inserting this time stamp), and the date and time the message was received. Relayed messages will have multiple time stamp lines. Details for formation of these lines, including their syntax, is specified in section 4.4.

Additional discussion about the operation of the DATA command appears in section 3.3.

Syntax:
"DATA" CRLF

4.1.1.5 RESET (RSET)

This command specifies that the current mail transaction will be aborted. Any stored sender, recipients, and mail data MUST be discarded, and all buffers and state tables cleared. The receiver MUST send a "250 OK" reply to a RSET command with no arguments. A reset command may be issued by the client at any time. It is effectively equivalent to a NOOP (i.e., if has no effect) if issued immediately after EHLO, before EHLO is issued in the session, after an end-of-data indicator has been sent and acknowledged, or immediately before a QUIT. An SMTP server MUST NOT close the connection as the result of receiving a RSET; that action is reserved for QUIT (see section 4.1.1.10).

Since EHLO implies some additional processing and response by the server, RSET will normally be more efficient than reissuing that command, even though the formal semantics are the same.

There are circumstances, contrary to the intent of this specification, in which an SMTP server may receive an indication that the underlying TCP connection has been closed or reset. To preserve the robustness of the mail system, SMTP servers SHOULD be prepared for this condition and SHOULD treat it as if a QUIT had been received before the connection disappeared.

Syntax:
"RSET" CRLF

4.1.1.6 VERIFY (VRFY)

This command asks the receiver to confirm that the argument identifies a user or mailbox. If it is a user name, information is returned as specified in section 3.5.

This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer.

Syntax:
"VRFY" SP String CRLF

4.1.1.7 EXPAND (EXPN)

This command asks the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list. If the command is successful, a reply is returned containing information as described in section 3.5. This reply will have multiple lines except in the trivial case of a one-member list.

This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer and may be issued at any time.

Syntax:
"EXPN" SP String CRLF

4.1.1.8 HELP (HELP)

This command causes the server to send helpful information to the client. The command MAY take an argument (e.g., any command name) and return more specific information as a response.

This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer and may be issued at any time.

SMTP servers SHOULD support HELP without arguments and MAY support it with arguments.

Syntax:
"HELP" [ SP String ] CRLF

4.1.1.9 NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the receiver send an OK reply.







This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer and may be issued at any time. If a parameter string is specified, servers SHOULD ignore it.

Syntax:
"NOOP" [ SP String ] CRLF

4.1.1.10 QUIT (QUIT)

This command specifies that the receiver MUST send an OK reply, and then close the transmission channel.

The receiver MUST NOT intentionally close the transmission channel until it receives and replies to a QUIT command (even if there was an error). The sender MUST NOT intentionally close the transmission channel until it sends a QUIT command and SHOULD wait until it receives the reply (even if there was an error response to a previous command). If the connection is closed prematurely due to violations of the above or system or network failure, the server MUST cancel any pending transaction, but not undo any previously completed transaction, and generally MUST act as if the command or transaction in progress had received a temporary error (i.e., a 4yz response).

The QUIT command may be issued at any time.

Syntax:
"QUIT" CRLF

4.1.2 Command Argument Syntax

The syntax of the argument fields of the above commands (using the syntax specified in [8] where applicable) is given below. Some of the productions given below are used only in conjunction with source routes as described in appendix C. Terminals not defined in this document, such as ALPHA, DIGIT, SP, CR, LF, CRLF, are as defined in the "core" syntax [8 (section 6)] or in the message format syntax [32].

```
Reverse-path = Path
Forward-path = Path
Path = "<" [ A-d-l ":" ] Mailbox ">"
A-d-l = At-domain *( "," A-d-l )
      ; Note that this form, the so-called "source route",
      ; MUST BE accepted, SHOULD NOT be generated, and SHOULD be
      ; ignored.
At-domain = "@" domain
Mail-parameters = esmtp-param *(SP esmtp-param)
Rcpt-parameters = esmtp-param *(SP esmtp-param)
```

```
esmtp-param     = esmtp-keyword ["=" esmtp-value]
esmtp-keyword   = (ALPHA / DIGIT) *(ALPHA / DIGIT / "-")
esmtp-value     = 1*(%d33-60 / %d62-127)
      ; any CHAR excluding "=", SP, and control characters
Keyword  = Ldh-str
Argument = Atom
Domain = (sub-domain 1*("." sub-domain)) / address-literal
sub-domain = Let-dig [Ldh-str]

address-literal = "[" IPv4-address-literal /
                      IPv6-address-literal /
                      General-address-literal "]"
      ; See section 4.1.3

Mailbox = Local-part "@" Domain

Local-part = Dot-string / Quoted-string
      ; MAY be case-sensitive

Dot-string = Atom *("." Atom)

Atom = 1*atext

Quoted-string = DQUOTE *qcontent DQUOTE

String = Atom / Quoted-string
```

While the above definition for Local-part is relatively permissive, for maximum interoperability, a host that expects to receive mail SHOULD avoid defining mailboxes where the Local-part requires (or uses) the Quoted-string form or where the Local-part is case-sensitive. For any purposes that require generating or comparing Local-parts (e.g., to specific mailbox names), all quoted forms MUST be treated as equivalent and the sending system SHOULD transmit the form that uses the minimum quoting possible.

Systems MUST NOT define mailboxes in such a way as to require the use in SMTP of non-ASCII characters (octets with the high order bit set to one) or ASCII "control characters" (decimal value 0-31 and 127). These characters MUST NOT be used in MAIL or RCPT commands or other commands that require mailbox names.

Note that the backslash, "\", is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe\,Smith" indicates a single nine character user field with the comma being the fourth character of the field.







To promote interoperability and consistent with long-standing guidance about conservative use of the DNS in naming and applications (e.g., see section 2.3.1 of the base DNS document, RFC1035 [22]), characters outside the set of alphas, digits, and hyphen MUST NOT appear in domain name labels for SMTP clients or servers. In particular, the underscore character is not permitted. SMTP servers that receive a command in which invalid character codes have been employed, and for which there are no other reasons for rejection, MUST reject that command with a 501 response.

### 4.1.3 Address Literals

Sometimes a host is not known to the domain name system and communication (and, in particular, communication to report and repair the error) is blocked. To bypass this barrier a special literal form of the address is allowed as an alternative to a domain name. For IPv4 addresses, this form uses four small decimal integers separated by dots and enclosed by brackets such as [123.255.37.2], which indicates an (IPv4) Internet Address in sequence-of-octets form. For IPv6 and other forms of addressing that might eventually be standardized, the form consists of a standardized "tag" that identifies the address syntax, a colon, and the address itself, in a format specified as part of the IPv6 standards [17].

Specifically:

```
IPv4-address-literal = Snum 3("." Snum)
IPv6-address-literal = "IPv6:" IPv6-addr
General-address-literal = Standardized-tag ":" 1*dcontent
Standardized-tag = Ldh-str
      ; MUST be specified in a standards-track RFC
      ; and registered with IANA

Snum = 1*3DIGIT  ; representing a decimal integer
      ; value in the range 0 through 255
Let-dig = ALPHA / DIGIT
Ldh-str = *( ALPHA / DIGIT / "-" ) Let-dig

IPv6-addr = IPv6-full / IPv6-comp / IPv6v4-full / IPv6v4-comp
IPv6-hex  = 1*4HEXDIG
IPv6-full = IPv6-hex 7(":" IPv6-hex)
IPv6-comp = [IPv6-hex *5(":" IPv6-hex)] "::" [IPv6-hex *5(":"
           IPv6-hex)]
      ; The "::" represents at least 2 16-bit groups of zeros
      ; No more than 6 groups in addition to the "::" may be
      ; present
IPv6v4-full = IPv6-hex 5(":" IPv6-hex) ":" IPv4-address-literal
IPv6v4-comp = [IPv6-hex *3(":" IPv6-hex)] "::"
```

```
                [IPv6-hex *3(":" IPv6-hex) ":"] IPv4-address-literal
        ; The "::" represents at least 2 16-bit groups of zeros
        ; No more than 4 groups in addition to the "::" and
        ; IPv4-address-literal may be present
```

#### 4.1.4 Order of Commands

There are restrictions on the order in which these commands may be used.

A session that will contain mail transactions MUST first be initialized by the use of the EHLO command. An SMTP server SHOULD accept commands for non-mail transactions (e.g., VRFY or EXPN) without this initialization.

An EHLO command MAY be issued by a client later in the session. If it is issued after the session begins, the SMTP server MUST clear all buffers and reset the state exactly as if a RSET command had been issued. In other words, the sequence of RSET followed immediately by EHLO is redundant, but not harmful other than in the performance cost of executing unnecessary commands.

If the EHLO command is not acceptable to the SMTP server, 501, 500, or 502 failure replies MUST be returned as appropriate. The SMTP server MUST stay in the same state after transmitting these replies that it was in before the EHLO was received.

The SMTP client MUST, if possible, ensure that the domain parameter to the EHLO command is a valid principal host name (not a CNAME or MX name) for its host. If this is not possible (e.g., when the client's address is dynamically assigned and the client does not have an obvious name), an address literal SHOULD be substituted for the domain name and supplemental information provided that will assist in identifying the client.

An SMTP server MAY verify that the domain name parameter in the EHLO command actually corresponds to the IP address of the client. However, the server MUST NOT refuse to accept a message for this reason if the verification fails: the information about verification failure is for logging and tracing only.

The NOOP, HELP, EXPN, VRFY, and RSET commands can be used at any time during a session, or without previously initializing a session. SMTP servers SHOULD process these normally (that is, not return a 503 code) even if no EHLO command has yet been received; clients SHOULD open a session with EHLO before sending these commands.







If these rules are followed, the example in RFC 821 that shows "550 access denied to you" in response to an EXPN command is incorrect unless an EHLO command precedes the EXPN or the denial of access is based on the client's IP address or other authentication or authorization-determining mechanisms.

The MAIL command (or the obsolete SEND, SOML, or SAML commands) begins a mail transaction. Once started, a mail transaction consists of a transaction beginning command, one or more RCPT commands, and a DATA command, in that order. A mail transaction may be aborted by the RSET (or a new EHLO) command. There may be zero or more transactions in a session. MAIL (or SEND, SOML, or SAML) MUST NOT be sent if a mail transaction is already open, i.e., it should be sent only if no mail transaction had been started in the session, or it the previous one successfully concluded with a successful DATA command, or if the previous one was aborted with a RSET.

If the transaction beginning command argument is not acceptable, a 501 failure reply MUST be returned and the SMTP server MUST stay in the same state. If the commands in a transaction are out of order to the degree that they cannot be processed by the server, a 503 failure reply MUST be returned and the SMTP server MUST stay in the same state.

The last command in a session MUST be the QUIT command. The QUIT command cannot be used at any other time in a session, but SHOULD be used by the client SMTP to request connection closure, even when no session opening command was sent and accepted.

#### 4.1.5 Private-use Commands

As specified in section 2.2.2, commands starting in "X" may be used by bilateral agreement between the client (sending) and server (receiving) SMTP agents. An SMTP server that does not recognize such a command is expected to reply with "500 Command not recognized". An extended SMTP server MAY list the feature names associated with these private commands in the response to the EHLO command.

Commands sent or accepted by SMTP systems that do not start with "X" MUST conform to the requirements of section 2.2.2.

### 4.2 SMTP Replies

Replies to SMTP commands serve to ensure the synchronization of requests and actions in the process of mail transfer and to guarantee that the SMTP client always knows the state of the SMTP server. Every command MUST generate exactly one reply.

The details of the command-reply sequence are described in section 4.3.

An SMTP reply consists of a three digit number (transmitted as three numeric characters) followed by some text unless specified otherwise in this document. The number is for use by automata to determine what state to enter next; the text is for the human user. The three digits contain enough encoded information that the SMTP client need not examine the text and may either discard it or pass it on to the user, as appropriate. Exceptions are as noted elsewhere in this document. In particular, the 220, 221, 251, 421, and 551 reply codes are associated with message text that must be parsed and interpreted by machines. In the general case, the text may be receiver dependent and context dependent, so there are likely to be varying texts for each reply code. A discussion of the theory of reply codes is given in section 4.2.1. Formally, a reply is defined to be the sequence: a three-digit code, <SP>, one line of text, and <CRLF>, or a multiline reply (as defined in section 4.2.1). Since, in violation of this specification, the text is sometimes not sent, clients which do not receive it SHOULD be prepared to process the code alone (with or without a trailing space character). Only the EHLO, EXPN, and HELP commands are expected to result in multiline replies in normal circumstances, however, multiline replies are allowed for any command.

In ABNF, server responses are:

```
Greeting = "220 " Domain [ SP text ] CRLF
Reply-line = Reply-code [ SP text ] CRLF
```

where "Greeting" appears only in the 220 response that announces that the server is opening its part of the connection.

An SMTP server SHOULD send only the reply codes listed in this document. An SMTP server SHOULD use the text shown in the examples whenever appropriate.

An SMTP client MUST determine its actions only by the reply code, not by the text (except for the "change of address" 251 and 551 and, if necessary, 220, 221, and 421 replies); in the general case, any text, including no text at all (although senders SHOULD NOT send bare codes), MUST be acceptable. The space (blank) following the reply code is considered part of the text. Whenever possible, a receiver-SMTP SHOULD test the first digit (severity indication) of the reply code.







The list of codes that appears below MUST NOT be construed as permanent. While the addition of new codes should be a rare and significant activity, with supplemental information in the textual part of the response being preferred, new codes may be added as the result of new Standards or Standards-track specifications. Consequently, a sender-SMTP MUST be prepared to handle codes not specified in this document and MUST do so by interpreting the first digit only.

### 4.2.1 Reply Code Severities and Theory

The three digits of the reply each have a special significance. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated SMTP client, or one that receives an unexpected code, will be able to determine its next action (proceed as planned, redo, retrench, etc.) by examining this first digit. An SMTP client that wants to know approximately what kind of error occurred (e.g., mail system error, command syntax error) may examine the second digit. The third digit and any supplemental information that may be present is reserved for the finest gradation of information.

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply
The command has been accepted, but the requested action is being held in abeyance, pending confirmation of the information in this reply. The SMTP client should send another command specifying whether to continue or abort the action. Note: unextended SMTP does not have any commands that allow this type of reply, and so does not have continue or abort commands.

2yz Positive Completion reply
The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply
The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The SMTP client should send another command specifying this information. This reply is used in command sequence groups (i.e., in DATA).

4yz Transient Negative Completion reply
The command was not accepted, and the requested action did not occur. However, the error condition is temporary and the action may be requested again. The sender should return to the beginning of the command sequence (if any). It is difficult to assign a meaning to "transient" when two different sites (receiver- and

sender-SMTP agents) must agree on the interpretation. Each reply in this category might have a different time value, but the SMTP client is encouraged to try again. A rule of thumb to determine whether a reply fits into the 4yz or the 5yz category (see below) is that replies are 4yz if they can be successful if repeated without any change in command form or in properties of the sender or receiver (that is, the command is repeated identically and the receiver does not put up a new implementation.)

5yz Permanent Negative Completion reply
The command was not accepted and the requested action did not occur. The SMTP client is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct the SMTP client to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered the account status).

The second digit encodes responses in specific categories:

x0z Syntax: These replies refer to syntax errors, syntactically correct commands that do not fit any functional category, and unimplemented or superfluous commands.

x1z Information: These are replies to requests for information, such as status or help.

x2z Connections: These are replies referring to the transmission channel.

x3z Unspecified.

x4z Unspecified.

x5z Mail system: These replies indicate the status of the receiver mail system vis-a-vis the requested transfer or other mail system action.

The third digit gives a finer gradation of meaning in each category specified by the second digit. The list of replies illustrates this. Each reply text is recommended rather than mandatory, and may even change according to the command with which it is associated. On the other hand, the reply codes must strictly follow the specifications in this section. Receiver implementations should not invent new codes for slightly different situations from the ones described here, but rather adapt codes already defined.







For example, a command such as NOOP, whose successful execution does not offer the SMTP client any new information, will return a 250 reply. The reply is 502 when the command requests an unimplemented non-site-specific action. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

The reply text may be longer than a single line; in these cases the complete text must be marked so the SMTP client knows when it can stop reading the reply. This requires a special format to indicate a multiple line reply.

The format for multiline replies requires that every line, except the last, begin with the reply code, followed immediately by a hyphen, "-" (also known as minus), followed by text. The last line will begin with the reply code, followed immediately by <SP>, optionally some text, and <CRLF>. As noted above, servers SHOULD send the <SP> if subsequent text is not sent, but clients MUST be prepared for it to be omitted.

For example:

```
123-First line
123-Second line
123-234 text beginning with numbers
123 The last line
```

In many cases the SMTP client then simply needs to search for a line beginning with the reply code followed by <SP> or <CRLF> and ignore all preceding lines. In a few cases, there is important data for the client in the reply "text". The client will be able to identify these cases from the current context.

### 4.2.2 Reply Codes by Function Groups

```
500 Syntax error, command unrecognized
   (This may include errors such as command line too long)
501 Syntax error in parameters or arguments
502 Command not implemented  (see section 4.2.4)
503 Bad sequence of commands
504 Command parameter not implemented

211 System status, or system help reply
214 Help message
   (Information on how to use the receiver or the meaning of a
   particular non-standard command; this reply is useful only
   to the human user)
```

```
220 <domain> Service ready
221 <domain> Service closing transmission channel
421 <domain> Service not available, closing transmission channel
   (This may be a reply to any command if the service knows it
   must shut down)

250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
   (See section 3.4)
252 Cannot VRFY user, but will accept message and attempt
    delivery
   (See section 3.5.3)
450 Requested mail action not taken: mailbox unavailable
   (e.g., mailbox busy)
550 Requested action not taken: mailbox unavailable
   (e.g., mailbox not found, no access, or command rejected
   for policy reasons)
451 Requested action aborted: error in processing
551 User not local; please try <forward-path>
   (See section 3.4)
452 Requested action not taken: insufficient system storage
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed
   (e.g., mailbox syntax incorrect)
354 Start mail input; end with <CRLF>.<CRLF>
554 Transaction failed (Or, in the case of a connection-opening
    response, "No SMTP service here")
```

#### 4.2.3 Reply Codes in Numeric Order

```
211 System status, or system help reply
214 Help message
   (Information on how to use the receiver or the meaning of a
   particular non-standard command; this reply is useful only
   to the human user)
220 <domain> Service ready
221 <domain> Service closing transmission channel
250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
   (See section 3.4)
252 Cannot VRFY user, but will accept message and attempt
   delivery
   (See section 3.5.3)

354 Start mail input; end with <CRLF>.<CRLF>
421 <domain> Service not available, closing transmission channel
   (This may be a reply to any command if the service knows it
   must shut down)
450 Requested mail action not taken: mailbox unavailable
   (e.g., mailbox busy)
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage
500 Syntax error, command unrecognized
   (This may include errors such as command line too long)
501 Syntax error in parameters or arguments
502 Command not implemented (see section 4.2.4)
503 Bad sequence of commands
504 Command parameter not implemented
550 Requested action not taken: mailbox unavailable
   (e.g., mailbox not found, no access, or command rejected
   for policy reasons)
551 User not local; please try <forward-path>
   (See section 3.4)
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed
   (e.g., mailbox syntax incorrect)
554 Transaction failed  (Or, in the case of a connection-opening
    response, "No SMTP service here")
```

#### 4.2.4 Reply Code 502

Questions have been raised as to when reply code 502 (Command not implemented) SHOULD be returned in preference to other codes. 502 SHOULD be used when the command is actually recognized by the SMTP server, but not implemented. If the command is not recognized, code 500 SHOULD be returned. Extended SMTP systems MUST NOT list capabilities in response to EHLO for which they will return 502 (or 500) replies.

#### 4.2.5 Reply Codes After DATA and the Subsequent <CRLF>.<CRLF>

When an SMTP server returns a positive completion status (2yz code) after the DATA command is completed with <CRLF>.<CRLF>, it accepts responsibility for:

- delivering the message (if the recipient mailbox exists), or
- if attempts to deliver the message fail due to transient conditions, retrying delivery some reasonable number of times at intervals as specified in section 4.5.4.

- if attempts to deliver the message fail due to permanent conditions, or if repeated attempts to deliver the message fail due to transient conditions, returning appropriate notification to the sender of the original message (using the address in the SMTP MAIL command).

When an SMTP server returns a permanent error status (5yz) code after the DATA command is completed with <CRLF>.<CRLF>, it MUST NOT make any subsequent attempt to deliver that message. The SMTP client retains responsibility for delivery of that message and may either return it to the user or requeue it for a subsequent attempt (see section 4.5.4.1).

The user who originated the message SHOULD be able to interpret the return of a transient failure status (by mail message or otherwise) as a non-delivery indication, just as a permanent failure would be interpreted. I.e., if the client SMTP successfully handles these conditions, the user will not receive such a reply.

When an SMTP server returns a permanent error status (5yz) code after the DATA command is completely with <CRLF>.<CRLF>, it MUST NOT make any subsequent attempt to deliver the message. As with temporary error status codes, the SMTP client retains responsibility for the message, but SHOULD not again attempt delivery to the same server without user review and intervention of the message.

## 4.3 Sequencing of Commands and Replies

### 4.3.1 Sequencing Overview

The communication between the sender and receiver is an alternating dialogue, controlled by the sender. As such, the sender issues a command and the receiver responds with a reply. Unless other arrangements are negotiated through service extensions, the sender MUST wait for this response before sending further commands.

One important reply is the connection greeting. Normally, a receiver will send a 220 "Service ready" reply when the connection is completed. The sender SHOULD wait for this greeting message before sending any commands.

Note: all the greeting-type replies have the official name (the fully-qualified primary domain name) of the server host as the first word following the reply code. Sometimes the host will have no meaningful name. See 4.1.3 for a discussion of alternatives in these situations.





For example,

```
   220 ISIF.USC.EDU Service ready
or
   220 mail.foo.com SuperSMTP v 6.1.2 Service ready
or
   220 [10.0.0.1] Clueless host service ready
```

The table below lists alternative success and failure replies for each command. These SHOULD be strictly adhered to: a receiver may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

### 4.3.2 Command-Reply Sequences

Each command is listed with its usual possible replies. The prefixes used before the possible replies are "I" for intermediate, "S" for success, and "E" for error. Since some servers may generate other replies under special circumstances, and to allow for future extension, SMTP clients SHOULD, when possible, interpret only the first digit of the reply and MUST be prepared to deal with unrecognized reply codes by interpreting the first digit only. Unless extended using the mechanisms described in section 2.2, SMTP servers MUST NOT transmit reply codes to an SMTP client that are other than three digits or that do not start in a digit between 2 and 5 inclusive.

These sequencing rules and, in principle, the codes themselves, can be extended or modified by SMTP extensions offered by the server and accepted (requested) by the client.

In addition to the codes listed below, any SMTP command can return any of the following codes if the corresponding unusual circumstances are encountered:

500 For the "command line too long" case or if the command name was not recognized. Note that producing a "command not recognized" error in response to the required subset of these commands is a violation of this specification.

501 Syntax error in command or arguments. In order to provide for future extensions, commands that are specified in this document as not accepting arguments (DATA, RSET, QUIT) SHOULD return a 501 message if arguments are supplied in the absence of EHLO-advertised extensions.

421 Service shutting down and closing transmission channel

Specific sequences are:

```
CONNECTION ESTABLISHMENT
   S: 220
   E: 554
EHLO or HELO
   S: 250
   E: 504, 550
MAIL
   S: 250
   E: 552, 451, 452, 550, 553, 503
RCPT
   S: 250, 251 (but see section 3.4 for discussion of 251 and 551)
   E: 550, 551, 552, 553, 450, 451, 452, 503, 550
DATA
   I: 354 -> data -> S: 250
                     E: 552, 554, 451, 452
   E: 451, 554, 503
RSET
   S: 250
VRFY
   S: 250, 251, 252
   E: 550, 551, 553, 502, 504
EXPN
   S: 250, 252
   E: 550, 500, 502, 504
HELP
   S: 211, 214
   E: 502, 504
NOOP
   S: 250
QUIT
   S: 221
```

## 4.4 Trace Information

When an SMTP server receives a message for delivery or further processing, it MUST insert trace ("time stamp" or "Received") information at the beginning of the message content, as discussed in section 4.1.1.4.

This line MUST be structured as follows:

- The FROM field, which MUST be supplied in an SMTP environment, SHOULD contain both (1) the name of the source host as presented in the EHLO command and (2) an address literal containing the IP address of the source, determined from the TCP connection.





- The ID field MAY contain an "@" as suggested in RFC 822, but this is not required.

- The FOR field MAY contain a list of <path> entries when multiple RCPT commands have been given. This may raise some security issues and is usually not desirable; see section 7.2.

An Internet mail program MUST NOT change a Received: line that was previously added to the message header. SMTP servers MUST prepend Received lines to messages; they MUST NOT change the order of existing lines or insert Received lines in any other location.

As the Internet grows, comparability of Received fields is important for detecting problems, especially slow relays. SMTP servers that create Received fields SHOULD use explicit offsets in the dates (e.g., -0800), rather than time zone names of any type. Local time (with an offset) is preferred to UT when feasible. This formulation allows slightly more information about local circumstances to be specified. If UT is needed, the receiver need merely do some simple arithmetic to convert the values. Use of UT loses information about the time zone-location of the server. If it is desired to supply a time zone name, it SHOULD be included in a comment.

When the delivery SMTP server makes the "final delivery" of a message, it inserts a return-path line at the beginning of the mail data. This use of return-path is required; mail systems MUST support it. The return-path line preserves the information in the <reverse-path> from the MAIL command. Here, final delivery means the message has left the SMTP environment. Normally, this would mean it had been delivered to the destination user or an associated mail drop, but in some cases it may be further processed and transmitted by another mail system.

It is possible for the mailbox in the return path to be different from the actual sender's mailbox, for example, if error responses are to be delivered to a special error handling mailbox rather than to the message sender. When mailing lists are involved, this arrangement is common and useful as a means of directing errors to the list maintainer rather than the message originator.

The text above implies that the final mail data will begin with a return path line, followed by one or more time stamp lines. These lines will be followed by the mail data headers and body [32].

It is sometimes difficult for an SMTP server to determine whether or not it is making final delivery since forwarding or other operations may occur after the message is accepted for delivery. Consequently,

any further (forwarding, gateway, or relay) systems MAY remove the return path and rebuild the MAIL command as needed to ensure that exactly one such line appears in a delivered message.

A message-originating SMTP system SHOULD NOT send a message that already contains a Return-path header. SMTP servers performing a relay function MUST NOT inspect the message data, and especially not to the extent needed to determine if Return-path headers are present. SMTP servers making final delivery MAY remove Return-path headers before adding their own.

The primary purpose of the Return-path is to designate the address to which messages indicating non-delivery or other mail system failures are to be sent. For this to be unambiguous, exactly one return path SHOULD be present when the message is delivered. Systems using RFC 822 syntax with non-SMTP transports SHOULD designate an unambiguous address, associated with the transport envelope, to which error reports (e.g., non-delivery messages) should be sent.

Historical note: Text in RFC 822 that appears to contradict the use of the Return-path header (or the envelope reverse path address from the MAIL command) as the destination for error messages is not applicable on the Internet. The reverse path address (as copied into the Return-path) MUST be used as the target of any mail containing delivery error messages.

In particular:

- a gateway from SMTP->elsewhere SHOULD insert a return-path header, unless it is known that the "elsewhere" transport also uses Internet domain addresses and maintains the envelope sender address separately.

- a gateway from elsewhere->SMTP SHOULD delete any return-path header present in the message, and either copy that information to the SMTP envelope or combine it with information present in the envelope of the other transport system to construct the reverse path argument to the MAIL command in the SMTP envelope.

The server must give special treatment to cases in which the processing following the end of mail data indication is only partially successful. This could happen if, after accepting several recipients and the mail data, the SMTP server finds that the mail data could be successfully delivered to some, but not all, of the recipients. In such cases, the response to the DATA command MUST be an OK reply. However, the SMTP server MUST compose and send an "undeliverable mail" notification message to the originator of the message.







A single notification listing all of the failed recipients or separate notification messages MUST be sent for each failed recipient. For economy of processing by the sender, the former is preferred when possible. All undeliverable mail notification messages are sent using the MAIL command (even if they result from processing the obsolete SEND, SOML, or SAML commands) and use a null return path as discussed in section 3.7.

The time stamp line and the return path line are formally defined as follows:

```
Return-path-line = "Return-Path:" FWS Reverse-path <CRLF>

Time-stamp-line = "Received:" FWS Stamp <CRLF>

Stamp = From-domain By-domain Opt-info ";"  FWS date-time

      ; where "date-time" is as defined in [32]
      ; but the "obs-" forms, especially two-digit
      ; years, are prohibited in SMTP and MUST NOT be used.

From-domain = "FROM" FWS Extended-Domain CFWS

By-domain = "BY" FWS Extended-Domain CFWS

Extended-Domain = Domain /
           ( Domain FWS "(" TCP-info ")" ) /
           ( Address-literal FWS "(" TCP-info ")" )

TCP-info = Address-literal / ( Domain FWS Address-literal )
      ; Information derived by server from TCP connection
      ; not client EHLO.

Opt-info = [Via] [With] [ID] [For]

Via = "VIA" FWS Link CFWS

With = "WITH" FWS Protocol CFWS

ID = "ID" FWS String / msg-id CFWS

For = "FOR" FWS 1*( Path / Mailbox ) CFWS

Link = "TCP" / Addtl-Link
Addtl-Link = Atom
      ; Additional standard names for links are registered with the
         ; Internet Assigned Numbers Authority (IANA).  "Via" is
         ; primarily of value with non-Internet transports.  SMTP
```

```
        ; servers SHOULD NOT use unregistered names.
Protocol = "ESMTP" / "SMTP" / Attdl-Protocol
Attdl-Protocol = Atom
      ; Additional standard names for protocols are registered with the
         ; Internet Assigned Numbers Authority (IANA).  SMTP servers
         ; SHOULD NOT use unregistered names.
```

## 4.5 Additional Implementation Issues

### 4.5.1 Minimum Implementation

In order to make SMTP workable, the following minimum implementation is required for all receivers. The following commands MUST be supported to conform to this specification:

EHLO
HELO
MAIL
RCPT
DATA
RSET
NOOP
QUIT
VRFY

Any system that includes an SMTP server supporting mail relaying or delivery MUST support the reserved mailbox "postmaster" as a case-insensitive local name. This postmaster address is not strictly necessary if the server always returns 554 on connection opening (as described in section 3.1). The requirement to accept mail for postmaster implies that RCPT commands which specify a mailbox for postmaster at any of the domains for which the SMTP server provides mail service, as well as the special case of "RCPT TO:<Postmaster>" (with no domain specification), MUST be supported.

SMTP systems are expected to make every reasonable effort to accept mail directed to Postmaster from any other system on the Internet. In extreme cases --such as to contain a denial of service attack or other breach of security-- an SMTP server may block mail directed to Postmaster. However, such arrangements SHOULD be narrowly tailored so as to avoid blocking messages which are not part of such attacks.

### 4.5.2 Transparency

Without some provision for data transparency, the character sequence "<CRLF>.<CRLF>" ends the mail text and cannot be sent by the user. In general, users are not aware of such "forbidden" sequences. To







allow all user composed text to be transmitted transparently, the following procedures are used:

- Before sending a line of mail text, the SMTP client checks the first character of the line. If it is a period, one additional period is inserted at the beginning of the line.

- When a line of mail text is received by the SMTP server, it checks the line. If the line is composed of a single period, it is treated as the end of mail indicator. If the first character is a period and there are other characters on the line, the first character is deleted.

The mail data may contain any of the 128 ASCII characters. All characters are to be delivered to the recipient's mailbox, including spaces, vertical and horizontal tabs, and other control characters. If the transmission channel provides an 8-bit byte (octet) data stream, the 7-bit ASCII codes are transmitted right justified in the octets, with the high order bits cleared to zero. See 3.7 for special treatment of these conditions in SMTP systems serving a relay function.

In some systems it may be necessary to transform the data as it is received and stored. This may be necessary for hosts that use a different character set than ASCII as their local character set, that store data in records rather than strings, or which use special character sequences as delimiters inside mailboxes. If such transformations are necessary, they MUST be reversible, especially if they are applied to mail being relayed.

### 4.5.3 Sizes and Timeouts

#### 4.5.3.1 Size limits and minimums

There are several objects that have required minimum/maximum sizes. Every implementation MUST be able to receive objects of at least these sizes. Objects larger than these sizes SHOULD be avoided when possible. However, some Internet mail constructs such as encoded X.400 addresses [16] will often require larger objects: clients MAY attempt to transmit these, but MUST be prepared for a server to reject them if they cannot be handled by it. To the maximum extent possible, implementation techniques which impose no limits on the length of these objects should be used.

local-part
The maximum total length of a user name or other local-part is 64 characters.

domain
The maximum total length of a domain name or number is 255 characters.

path
The maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators).

command line
The maximum total length of a command line including the command word and the <CRLF> is 512 characters. SMTP extensions may be used to increase this limit.

reply line
The maximum total length of a reply line including the reply code and the <CRLF> is 512 characters. More information may be conveyed through multiple-line replies.

text line
The maximum total length of a text line including the <CRLF> is 1000 characters (not counting the leading dot duplicated for transparency). This number may be increased by the use of SMTP Service Extensions.

message content
The maximum total length of a message content (including any message headers as well as the message body) MUST BE at least 64K octets. Since the introduction of Internet standards for multimedia mail [12], message lengths on the Internet have grown dramatically, and message size restrictions should be avoided if at all possible. SMTP server systems that must impose restrictions SHOULD implement the "SIZE" service extension [18], and SMTP client systems that will send large messages SHOULD utilize it when possible.

recipients buffer
The minimum total number of recipients that must be buffered is 100 recipients. Rejection of messages (for excessive recipients) with fewer than 100 RCPT commands is a violation of this specification. The general principle that relaying SMTP servers MUST NOT, and delivery SMTP servers SHOULD NOT, perform validation tests on message headers suggests that rejecting a message based on the total number of recipients shown in header fields is to be discouraged. A server which imposes a limit on the number of recipients MUST behave in an orderly fashion, such as to reject additional addresses over its limit rather than silently discarding addresses previously accepted. A client that needs to







deliver a message containing over 100 RCPT commands SHOULD be prepared to transmit in 100-recipient "chunks" if the server declines to accept more than 100 recipients in a single message.

Errors due to exceeding these limits may be reported by using the reply codes. Some examples of reply codes are:

500 Line too long.
or
501 Path too long
or
452 Too many recipients (see below)
or
552 Too much mail data.

RFC 821 [30] incorrectly listed the error where an SMTP server exhausts its implementation limit on the number of RCPT commands ("too many recipients") as having reply code 552. The correct reply code for this condition is 452. Clients SHOULD treat a 552 code in this case as a temporary, rather than permanent, failure so the logic below works.

When a conforming SMTP server encounters this condition, it has at least 100 successful RCPT commands in its recipients buffer. If the server is able to accept the message, then at least these 100 addresses will be removed from the SMTP client's queue. When the client attempts retransmission of those addresses which received 452 responses, at least 100 of these will be able to fit in the SMTP server's recipients buffer. Each retransmission attempt which is able to deliver anything will be able to dispose of at least 100 of these recipients.

If an SMTP server has an implementation limit on the number of RCPT commands and this limit is exhausted, it MUST use a response code of 452 (but the client SHOULD also be prepared for a 552, as noted above). If the server has a configured site-policy limitation on the number of RCPT commands, it MAY instead use a 5XX response code. This would be most appropriate if the policy limitation was intended to apply if the total recipient count for a particular message body were enforced even if that message body was sent in multiple mail transactions.

4.5.3.2 Timeouts

An SMTP client MUST provide a timeout mechanism. It MUST use per-command timeouts rather than somehow trying to time the entire mail transaction. Timeouts SHOULD be easily reconfigurable, preferably without recompiling the SMTP code. To implement this, a timer is set

for each SMTP command and for each buffer of the data transfer. The latter means that the overall timeout is inherently proportional to the size of the message.

Based on extensive experience with busy mail-relay hosts, the minimum per-command timeout values SHOULD be as follows:

Initial 220 Message: 5 minutes
An SMTP client process needs to distinguish between a failed TCP connection and a delay in receiving the initial 220 greeting message. Many SMTP servers accept a TCP connection but delay delivery of the 220 message until their system load permits more mail to be processed.

MAIL Command: 5 minutes

RCPT Command: 5 minutes
A longer timeout is required if processing of mailing lists and aliases is not deferred until after the message was accepted.

DATA Initiation: 2 minutes
This is while awaiting the "354 Start Input" reply to a DATA command.

Data Block: 3 minutes
This is while awaiting the completion of each TCP SEND call transmitting a chunk of data.

DATA Termination: 10 minutes.
This is while awaiting the "250 OK" reply. When the receiver gets the final period terminating the message data, it typically performs processing to deliver the message to a user mailbox. A spurious timeout at this point would be very wasteful and would typically result in delivery of multiple copies of the message, since it has been successfully sent and the server has accepted responsibility for delivery. See section 6.1 for additional discussion.

An SMTP server SHOULD have a timeout of at least 5 minutes while it is awaiting the next command from the sender.

### 4.5.4 Retry Strategies

The common structure of a host SMTP implementation includes user mailboxes, one or more areas for queuing messages in transit, and one or more daemon processes for sending and receiving mail. The exact structure will vary depending on the needs of the users on the host







and the number and size of mailing lists supported by the host. We describe several optimizations that have proved helpful, particularly for mailers supporting high traffic levels.

Any queuing strategy MUST include timeouts on all activities on a per-command basis. A queuing strategy MUST NOT send error messages in response to error messages under any circumstances.

#### 4.5.4.1 Sending Strategy

The general model for an SMTP client is one or more processes that periodically attempt to transmit outgoing mail. In a typical system, the program that composes a message has some method for requesting immediate attention for a new piece of outgoing mail, while mail that cannot be transmitted immediately MUST be queued and periodically retried by the sender. A mail queue entry will include not only the message itself but also the envelope information.

The sender MUST delay retrying a particular destination after one attempt has failed. In general, the retry interval SHOULD be at least 30 minutes; however, more sophisticated and variable strategies will be beneficial when the SMTP client can determine the reason for non-delivery.

Retries continue until the message is transmitted or the sender gives up; the give-up time generally needs to be at least 4-5 days. The parameters to the retry algorithm MUST be configurable.

A client SHOULD keep a list of hosts it cannot reach and corresponding connection timeouts, rather than just retrying queued mail items.

Experience suggests that failures are typically transient (the target system or its connection has crashed), favoring a policy of two connection attempts in the first hour the message is in the queue, and then backing off to one every two or three hours.

The SMTP client can shorten the queuing delay in cooperation with the SMTP server. For example, if mail is received from a particular address, it is likely that mail queued for that host can now be sent. Application of this principle may, in many cases, eliminate the requirement for an explicit "send queues now" function such as ETRN [9].

The strategy may be further modified as a result of multiple addresses per host (see below) to optimize delivery time vs. resource usage.

An SMTP client may have a large queue of messages for each unavailable destination host. If all of these messages were retried in every retry cycle, there would be excessive Internet overhead and the sending system would be blocked for a long period. Note that an SMTP client can generally determine that a delivery attempt has failed only after a timeout of several minutes and even a one-minute timeout per connection will result in a very large delay if retries are repeated for dozens, or even hundreds, of queued messages to the same host.

At the same time, SMTP clients SHOULD use great care in caching negative responses from servers. In an extreme case, if EHLO is issued multiple times during the same SMTP connection, different answers may be returned by the server. More significantly, 5yz responses to the MAIL command MUST NOT be cached.

When a mail message is to be delivered to multiple recipients, and the SMTP server to which a copy of the message is to be sent is the same for multiple recipients, then only one copy of the message SHOULD be transmitted. That is, the SMTP client SHOULD use the command sequence: MAIL, RCPT, RCPT,... RCPT, DATA instead of the sequence: MAIL, RCPT, DATA, ..., MAIL, RCPT, DATA. However, if there are very many addresses, a limit on the number of RCPT commands per MAIL command MAY be imposed. Implementation of this efficiency feature is strongly encouraged.

Similarly, to achieve timely delivery, the SMTP client MAY support multiple concurrent outgoing mail transactions. However, some limit may be appropriate to protect the host from devoting all its resources to mail.

#### 4.5.4.2 Receiving Strategy

The SMTP server SHOULD attempt to keep a pending listen on the SMTP port at all times. This requires the support of multiple incoming TCP connections for SMTP. Some limit MAY be imposed but servers that cannot handle more than one SMTP transaction at a time are not in conformance with the intent of this specification.

As discussed above, when the SMTP server receives mail from a particular host address, it could activate its own SMTP queuing mechanisms to retry any mail pending for that host address.

### 4.5.5 Messages with a null reverse-path

There are several types of notification messages which are required by existing and proposed standards to be sent with a null reverse path, namely non-delivery notifications as discussed in section 3.7,





other kinds of Delivery Status Notifications (DSNs) [24], and also Message Disposition Notifications (MDNs) [10]. All of these kinds of messages are notifications about a previous message, and they are sent to the reverse-path of the previous mail message. (If the delivery of such a notification message fails, that usually indicates a problem with the mail system of the host to which the notification message is addressed. For this reason, at some hosts the MTA is set up to forward such failed notification messages to someone who is able to fix problems with the mail system, e.g., via the postmaster alias.)

All other types of messages (i.e., any message which is not required by a standards-track RFC to have a null reverse-path) SHOULD be sent with with a valid, non-null reverse-path.

Implementors of automated email processors should be careful to make sure that the various kinds of messages with null reverse-path are handled correctly, in particular such systems SHOULD NOT reply to messages with null reverse-path.

## 5. Address Resolution and Mail Handling

Once an SMTP client lexically identifies a domain to which mail will be delivered for processing (as described in sections 3.6 and 3.7), a DNS lookup MUST be performed to resolve the domain name [22]. The names are expected to be fully-qualified domain names (FQDNs): mechanisms for inferring FQDNs from partial names or local aliases are outside of this specification and, due to a history of problems, are generally discouraged. The lookup first attempts to locate an MX record associated with the name. If a CNAME record is found instead, the resulting name is processed as if it were the initial name. If no MX records are found, but an A RR is found, the A RR is treated as if it was associated with an implicit MX RR, with a preference of 0, pointing to that host. If one or more MX RRs are found for a given name, SMTP systems MUST NOT utilize any A RRs associated with that name unless they are located using the MX RRs; the "implicit MX" rule above applies only if there are no MX records present. If MX records are present, but none of them are usable, this situation MUST be reported as an error.

When the lookup succeeds, the mapping can result in a list of alternative delivery addresses rather than a single address, because of multiple MX records, multihoming, or both. To provide reliable mail transmission, the SMTP client MUST be able to try (and retry) each of the relevant addresses in this list in order, until a delivery attempt succeeds. However, there MAY also be a configurable limit on the number of alternate addresses that can be tried. In any case, the SMTP client SHOULD try at least two addresses.

Two types of information is used to rank the host addresses: multiple MX records, and multihomed hosts.

Multiple MX records contain a preference indication that MUST be used in sorting (see below). Lower numbers are more preferred than higher ones. If there are multiple destinations with the same preference and there is no clear reason to favor one (e.g., by recognition of an easily-reached address), then the sender-SMTP MUST randomize them to spread the load across multiple mail exchangers for a specific organization.

The destination host (perhaps taken from the preferred MX record) may be multihomed, in which case the domain name resolver will return a list of alternative IP addresses. It is the responsibility of the domain name resolver interface to have ordered this list by decreasing preference if necessary, and SMTP MUST try them in the order presented.

Although the capability to try multiple alternative addresses is required, specific installations may want to limit or disable the use of alternative addresses. The question of whether a sender should attempt retries using the different addresses of a multihomed host has been controversial. The main argument for using the multiple addresses is that it maximizes the probability of timely delivery, and indeed sometimes the probability of any delivery; the counter-argument is that it may result in unnecessary resource use. Note that resource use is also strongly determined by the sending strategy discussed in section 4.5.4.1.

If an SMTP server receives a message with a destination for which it is a designated Mail eXchanger, it MAY relay the message (potentially after having rewritten the MAIL FROM and/or RCPT TO addresses), make final delivery of the message, or hand it off using some mechanism outside the SMTP-provided transport environment. Of course, neither of the latter require that the list of MX records be examined further.

If it determines that it should relay the message without rewriting the address, it MUST sort the MX records to determine candidates for delivery. The records are first ordered by preference, with the lowest-numbered records being most preferred. The relay host MUST then inspect the list for any of the names or addresses by which it might be known in mail transactions. If a matching record is found, all records at that preference level and higher-numbered ones MUST be discarded from consideration. If there are no records left at that point, it is an error condition, and the message MUST be returned as undeliverable. If records do remain, they SHOULD be tried, best preference first, as described above.







# 6. Problem Detection and Handling

## 6.1 Reliable Delivery and Replies by Email

When the receiver-SMTP accepts a piece of mail (by sending a "250 OK" message in response to DATA), it is accepting responsibility for delivering or relaying the message. It must take this responsibility seriously. It MUST NOT lose the message for frivolous reasons, such as because the host later crashes or because of a predictable resource shortage.

If there is a delivery failure after acceptance of a message, the receiver-SMTP MUST formulate and mail a notification message. This notification MUST be sent using a null ("<>") reverse path in the envelope. The recipient of this notification MUST be the address from the envelope return path (or the Return-Path: line). However, if this address is null ("<>"), the receiver-SMTP MUST NOT send a notification. Obviously, nothing in this section can or should prohibit local decisions (i.e., as part of the same system environment as the receiver-SMTP) to log or otherwise transmit information about null address events locally if that is desired. If the address is an explicit source route, it MUST be stripped down to its final hop.

For example, suppose that an error notification must be sent for a message that arrived with:

```
MAIL FROM:<@a,@b:user@d>
```

The notification message MUST be sent using:

```
RCPT TO:<user@d>
```

Some delivery failures after the message is accepted by SMTP will be unavoidable. For example, it may be impossible for the receiving SMTP server to validate all the delivery addresses in RCPT command(s) due to a "soft" domain system error, because the target is a mailing list (see earlier discussion of RCPT), or because the server is acting as a relay and has no immediate access to the delivering system.

To avoid receiving duplicate messages as the result of timeouts, a receiver-SMTP MUST seek to minimize the time required to respond to the final <CRLF>.<CRLF> end of data indicator. See RFC 1047 [28] for a discussion of this problem.

### 6.2 Loop Detection

Simple counting of the number of "Received:" headers in a message has proven to be an effective, although rarely optimal, method of detecting loops in mail systems. SMTP servers using this technique SHOULD use a large rejection threshold, normally at least 100 Received entries. Whatever mechanisms are used, servers MUST contain provisions for detecting and stopping trivial loops.

### 6.3 Compensating for Irregularities

Unfortunately, variations, creative interpretations, and outright violations of Internet mail protocols do occur; some would suggest that they occur quite frequently. The debate as to whether a well-behaved SMTP receiver or relay should reject a malformed message, attempt to pass it on unchanged, or attempt to repair it to increase the odds of successful delivery (or subsequent reply) began almost with the dawn of structured network mail and shows no signs of abating. Advocates of rejection claim that attempted repairs are rarely completely adequate and that rejection of bad messages is the only way to get the offending software repaired. Advocates of "repair" or "deliver no matter what" argue that users prefer that mail go through it if at all possible and that there are significant market pressures in that direction. In practice, these market pressures may be more important to particular vendors than strict conformance to the standards, regardless of the preference of the actual developers.

The problems associated with ill-formed messages were exacerbated by the introduction of the split-UA mail reading protocols [3, 26, 5, 21]. These protocols have encouraged the use of SMTP as a posting protocol, and SMTP servers as relay systems for these client hosts (which are often only intermittently connected to the Internet). Historically, many of those client machines lacked some of the mechanisms and information assumed by SMTP (and indeed, by the mail format protocol [7]). Some could not keep adequate track of time; others had no concept of time zones; still others could not identify their own names or addresses; and, of course, none could satisfy the assumptions that underlay RFC 822's conception of authenticated addresses.

In response to these weak SMTP clients, many SMTP systems now complete messages that are delivered to them in incomplete or incorrect form. This strategy is generally considered appropriate when the server can identify or authenticate the client, and there are prior agreements between them. By contrast, there is at best great concern about fixes applied by a relay or delivery SMTP server that has little or no knowledge of the user or client machine.







The following changes to a message being processed MAY be applied when necessary by an originating SMTP server, or one used as the target of SMTP as an initial posting protocol:

- Addition of a message-id field when none appears
- Addition of a date, time or time zone when none appears
- Correction of addresses to proper FQDN format

The less information the server has about the client, the less likely these changes are to be correct and the more caution and conservatism should be applied when considering whether or not to perform fixes and how. These changes MUST NOT be applied by an SMTP server that provides an intermediate relay function.

In all cases, properly-operating clients supplying correct information are preferred to corrections by the SMTP server. In all cases, documentation of actions performed by the servers (in trace fields and/or header comments) is strongly encouraged.

## 7. Security Considerations

### 7.1 Mail Security and Spoofing

SMTP mail is inherently insecure in that it is feasible for even fairly casual users to negotiate directly with receiving and relaying SMTP servers and create messages that will trick a naive recipient into believing that they came from somewhere else. Constructing such a message so that the "spoofed" behavior cannot be detected by an expert is somewhat more difficult, but not sufficiently so as to be a deterrent to someone who is determined and knowledgeable. Consequently, as knowledge of Internet mail increases, so does the knowledge that SMTP mail inherently cannot be authenticated, or integrity checks provided, at the transport level. Real mail security lies only in end-to-end methods involving the message bodies, such as those which use digital signatures (see [14] and, e.g., PGP [4] or S/MIME [31]).

Various protocol extensions and configuration options that provide authentication at the transport level (e.g., from an SMTP client to an SMTP server) improve somewhat on the traditional situation described above. However, unless they are accompanied by careful handoffs of responsibility in a carefully-designed trust environment, they remain inherently weaker than end-to-end mechanisms which use digitally signed messages rather than depending on the integrity of the transport system.

Efforts to make it more difficult for users to set envelope return path and header "From" fields to point to valid addresses other than their own are largely misguided: they frustrate legitimate applications in which mail is sent by one user on behalf of another or in which error (or normal) replies should be directed to a special address. (Systems that provide convenient ways for users to alter these fields on a per-message basis should attempt to establish a primary and permanent mailbox address for the user so that Sender fields within the message data can be generated sensibly.)

This specification does not further address the authentication issues associated with SMTP other than to advocate that useful functionality not be disabled in the hope of providing some small margin of protection against an ignorant user who is trying to fake mail.

### 7.2 "Blind" Copies

Addresses that do not appear in the message headers may appear in the RCPT commands to an SMTP server for a number of reasons. The two most common involve the use of a mailing address as a "list exploder" (a single address that resolves into multiple addresses) and the appearance of "blind copies". Especially when more than one RCPT command is present, and in order to avoid defeating some of the purpose of these mechanisms, SMTP clients and servers SHOULD NOT copy the full set of RCPT command arguments into the headers, either as part of trace headers or as informational or private-extension headers. Since this rule is often violated in practice, and cannot be enforced, sending SMTP systems that are aware of "bcc" use MAY find it helpful to send each blind copy as a separate message transaction containing only a single RCPT command.

There is no inherent relationship between either "reverse" (from MAIL, SAML, etc., commands) or "forward" (RCPT) addresses in the SMTP transaction ("envelope") and the addresses in the headers. Receiving systems SHOULD NOT attempt to deduce such relationships and use them to alter the headers of the message for delivery. The popular "Apparently-to" header is a violation of this principle as well as a common source of unintended information disclosure and SHOULD NOT be used.

### 7.3 VRFY, EXPN, and Security

As discussed in section 3.5, individual sites may want to disable either or both of VRFY or EXPN for security reasons. As a corollary to the above, implementations that permit this MUST NOT appear to have verified addresses that are not, in fact, verified. If a site







disables these commands for security reasons, the SMTP server MUST return a 252 response, rather than a code that could be confused with successful or unsuccessful verification.

Returning a 250 reply code with the address listed in the VRFY command after having checked it only for syntax violates this rule. Of course, an implementation that "supports" VRFY by always returning 550 whether or not the address is valid is equally not in conformance.

Within the last few years, the contents of mailing lists have become popular as an address information source for so-called "spammers." The use of EXPN to "harvest" addresses has increased as list administrators have installed protections against inappropriate uses of the lists themselves. Implementations SHOULD still provide support for EXPN, but sites SHOULD carefully evaluate the tradeoffs. As authentication mechanisms are introduced into SMTP, some sites may choose to make EXPN available only to authenticated requestors.

### 7.4 Information Disclosure in Announcements

There has been an ongoing debate about the tradeoffs between the debugging advantages of announcing server type and version (and, sometimes, even server domain name) in the greeting response or in response to the HELP command and the disadvantages of exposing information that might be useful in a potential hostile attack. The utility of the debugging information is beyond doubt. Those who argue for making it available point out that it is far better to actually secure an SMTP server rather than hope that trying to conceal known vulnerabilities by hiding the server's precise identity will provide more protection. Sites are encouraged to evaluate the tradeoff with that issue in mind; implementations are strongly encouraged to minimally provide for making type and version information available in some way to other network hosts.

### 7.5 Information Disclosure in Trace Fields

In some circumstances, such as when mail originates from within a LAN whose hosts are not directly on the public Internet, trace ("Received") fields produced in conformance with this specification may disclose host names and similar information that would not normally be available. This ordinarily does not pose a problem, but sites with special concerns about name disclosure should be aware of it. Also, the optional FOR clause should be supplied with caution or not at all when multiple recipients are involved lest it inadvertently disclose the identities of "blind copy" recipients to others.

### 7.6 Information Disclosure in Message Forwarding

As discussed in section 3.4, use of the 251 or 551 reply codes to identify the replacement address associated with a mailbox may inadvertently disclose sensitive information. Sites that are concerned about those issues should ensure that they select and configure servers appropriately.

### 7.7 Scope of Operation of SMTP Servers

It is a well-established principle that an SMTP server may refuse to accept mail for any operational or technical reason that makes sense to the site providing the server. However, cooperation among sites and installations makes the Internet possible. If sites take excessive advantage of the right to reject traffic, the ubiquity of email availability (one of the strengths of the Internet) will be threatened; considerable care should be taken and balance maintained if a site decides to be selective about the traffic it will accept and process.

In recent years, use of the relay function through arbitrary sites has been used as part of hostile efforts to hide the actual origins of mail. Some sites have decided to limit the use of the relay function to known or identifiable sources, and implementations SHOULD provide the capability to perform this type of filtering. When mail is rejected for these or other policy reasons, a 550 code SHOULD be used in response to EHLO, MAIL, or RCPT as appropriate.

## 8. IANA Considerations

IANA will maintain three registries in support of this specification. The first consists of SMTP service extensions with the associated keywords, and, as needed, parameters and verbs. As specified in section 2.2.2, no entry may be made in this registry that starts in an "X". Entries may be made only for service extensions (and associated keywords, parameters, or verbs) that are defined in standards-track or experimental RFCs specifically approved by the IESG for this purpose.

The second registry consists of "tags" that identify forms of domain literals other than those for IPv4 addresses (specified in RFC 821 and in this document) and IPv6 addresses (specified in this document). Additional literal types require standardization before being used; none are anticipated at this time.

The third, established by RFC 821 and renewed by this specification, is a registry of link and protocol identifiers to be used with the "via" and "with" subclauses of the time stamp ("Received: header")





described in section 4.4. Link and protocol identifiers in addition to those specified in this document may be registered only by standardization or by way of an RFC-documented, IESG-approved, Experimental protocol extension.

## 9. References

[1] American National Standards Institute (formerly United States of America Standards Institute), X3.4, 1968, "USA Code for Information Interchange". ANSI X3.4-1968 has been replaced by newer versions with slight modifications, but the 1968 version remains definitive for the Internet.

[2] Braden, R., "Requirements for Internet hosts - application and support", STD 3, RFC 1123, October 1989.

[3] Butler, M., Chase, D., Goldberger, J., Postel, J. and J. Reynolds, "Post Office Protocol - version 2", RFC 937, February 1985.

[4] Callas, J., Donnerhacke, L., Finney, H. and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[5] Crispin, M., "Interactive Mail Access Protocol - Version 2", RFC 1176, August 1990.

[6] Crispin, M., "Internet Message Access Protocol - Version 4", RFC 2060, December 1996.

[7] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", RFC 822, August 1982.

[8] Crocker, D. and P. Overell, Eds., "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[9] De Winter, J., "SMTP Service Extension for Remote Message Queue Starting", RFC 1985, August 1996.

[10] Fajman, R., "An Extensible Message Format for Message Disposition Notifications", RFC 2298, March 1998.

[11] Freed, N, "Behavior of and Requirements for Internet Firewalls", RFC 2979, October 2000.

[12] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, December 1996.

[13] Freed, N., "SMTP Service Extension for Command Pipelining", RFC 2920, September 2000.

[14] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[15] Gellens, R. and J. Klensin, "Message Submission", RFC 2476, December 1998.

[16] Kille, S., "Mapping between X.400 and RFC822/MIME", RFC 2156, January 1998.

[17] Hinden, R and S. Deering, Eds. "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[18] Klensin, J., Freed, N. and K. Moore, "SMTP Service Extension for Message Size Declaration", STD 10, RFC 1870, November 1995.

[19] Klensin, J., Freed, N., Rose, M., Stefferud, E. and D. Crocker, "SMTP Service Extensions", STD 10, RFC 1869, November 1995.

[20] Klensin, J., Freed, N., Rose, M., Stefferud, E. and D. Crocker, "SMTP Service Extension for 8bit-MIMEtransport", RFC 1652, July 1994.

[21] Lambert, M., "PCMAIL: A distributed mail system for personal computers", RFC 1056, July 1988.

[22] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[23] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, December 1996.

[24] Moore, K., "SMTP Service Extension for Delivery Status Notifications", RFC 1891, January 1996.

[25] Moore, K., and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 1894, January 1996.

[26] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.







[27] Partridge, C., "Mail routing and the domain system", RFC 974, January 1986.

[28] Partridge, C., "Duplicate messages and SMTP", RFC 1047, February 1988.

[29] Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", STD 7, RFC 793, September 1981.

[30] Postel, J., "Simple Mail Transfer Protocol", RFC 821, August 1982.

[31] Ramsdell, B., Ed., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[32] Resnick, P., Ed., "Internet Message Format", RFC 2822, April 2001.

[33] Vaudreuil, G., "SMTP Service Extensions for Transmission of Large and Binary MIME Messages", RFC 1830, August 1995.

[34] Vaudreuil, G., "Enhanced Mail System Status Codes", RFC 1893, January 1996.

## 10. Editor's Address

John C. Klensin
AT&T Laboratories
99 Bedford St
Boston, MA 02111 USA

Phone: 617-574-3076
EMail: klensin@research.att.com

## 11. Acknowledgments

Many people worked long and hard on the many iterations of this document. There was wide-ranging debate in the IETF DRUMS Working Group, both on its mailing list and in face to face discussions, about many technical issues and the role of a revised standard for Internet mail transport, and many contributors helped form the wording in this specification. The hundreds of participants in the many discussions since RFC 821 was produced are too numerous to mention, but they all helped this document become what it is.

APPENDICES

A. TCP Transport Service

The TCP connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as an 8-bit byte with the high-order bit cleared to zero. Service extensions may modify this rule to permit transmission of full 8-bit data bytes as part of the message body, but not in SMTP commands or responses.

B. Generating SMTP Commands from RFC 822 Headers

Some systems use RFC 822 headers (only) in a mail submission protocol, or otherwise generate SMTP commands from RFC 822 headers when such a message is handed to an MTA from a UA. While the MTA-UA protocol is a private matter, not covered by any Internet Standard, there are problems with this approach. For example, there have been repeated problems with proper handling of "bcc" copies and redistribution lists when information that conceptually belongs to a mail envelopes is not separated early in processing from header information (and kept separate).

It is recommended that the UA provide its initial ("submission client") MTA with an envelope separate from the message itself. However, if the envelope is not supplied, SMTP commands SHOULD be generated as follows:

1. Each recipient address from a TO, CC, or BCC header field SHOULD be copied to a RCPT command (generating multiple message copies if that is required for queuing or delivery). This includes any addresses listed in a RFC 822 "group". Any BCC fields SHOULD then be removed from the headers. Once this process is completed, the remaining headers SHOULD be checked to verify that at least one To:, Cc:, or Bcc: header remains. If none do, then a bcc: header with no additional information SHOULD be inserted as specified in [32].

2. The return address in the MAIL command SHOULD, if possible, be derived from the system's identity for the submitting (local) user, and the "From:" header field otherwise. If there is a system identity available, it SHOULD also be copied to the Sender header field if it is different from the address in the From header field. (Any Sender field that was already there SHOULD be removed.) Systems may provide a way for submitters to override the envelope return address, but may want to restrict its use to privileged users. This will not prevent mail forgery, but may lessen its incidence; see section 7.1.







When an MTA is being used in this way, it bears responsibility for ensuring that the message being transmitted is valid. The mechanisms for checking that validity, and for handling (or returning) messages that are not valid at the time of arrival, are part of the MUA-MTA interface and not covered by this specification.

A submission protocol based on Standard RFC 822 information alone MUST NOT be used to gateway a message from a foreign (non-SMTP) mail system into an SMTP environment. Additional information to construct an envelope must come from some source in the other environment, whether supplemental headers or the foreign system's envelope.

Attempts to gateway messages using only their header "to" and "cc" fields have repeatedly caused mail loops and other behavior adverse to the proper functioning of the Internet mail environment. These problems have been especially common when the message originates from an Internet mailing list and is distributed into the foreign environment using envelope information. When these messages are then processed by a header-only remailer, loops back to the Internet environment (and the mailing list) are almost inevitable.

C. Source Routes

Historically, the <reverse-path> was a reverse source routing list of hosts and a source mailbox. The first host in the <reverse-path> SHOULD be the host sending the MAIL command. Similarly, the <forward-path> may be a source routing lists of hosts and a destination mailbox. However, in general, the <forward-path> SHOULD contain only a mailbox and domain name, relying on the domain name system to supply routing information if required. The use of source routes is deprecated; while servers MUST be prepared to receive and handle them as discussed in section 3.3 and F.2, clients SHOULD NOT transmit them and this section was included only to provide context.

For relay purposes, the forward-path may be a source route of the form "@ONE,@TWO:JOE@THREE", where ONE, TWO, and THREE MUST BE fully-qualified domain names. This form is used to emphasize the distinction between an address and a route. The mailbox is an absolute address, and the route is information about how to get there. The two concepts should not be confused.

If source routes are used, RFC 821 and the text below should be consulted for the mechanisms for constructing and updating the forward- and reverse-paths.

The SMTP server transforms the command arguments by moving its own identifier (its domain name or that of any domain for which it is acting as a mail exchanger), if it appears, from the forward-path to the beginning of the reverse-path.

Notice that the forward-path and reverse-path appear in the SMTP commands and replies, but not necessarily in the message. That is, there is no need for these paths and especially this syntax to appear in the "To:" , "From:", "CC:", etc. fields of the message header. Conversely, SMTP servers MUST NOT derive final message delivery information from message header fields.

When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it MUST use its name as known in the transport environment to which it is relaying the mail rather than that of the transport environment from which the mail came (if they are different).

## D. Scenarios

This section presents complete scenarios of several types of SMTP sessions. In the examples, "C:" indicates what is said by the SMTP client, and "S:" indicates what is said by the SMTP server.

### D.1 A Typical SMTP Transaction Scenario

This SMTP example shows mail sent by Smith at host bar.com, to Jones, Green, and Brown at host foo.com. Here we assume that host bar.com contacts host foo.com directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host foo.com.

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>
```





```
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

### D.2 Aborted SMTP Transaction Scenario

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RSET
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

### D.3 Relayed Mail Scenario

Step 1 -- Source Host to Relay Host

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<JQP@bar.com>
S: 250 OK
C: RCPT TO:<@foo.com:Jones@XYZ.COM>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Date: Thu, 21 May 1998 05:33:29 -0700
```

```
C: From: John Q. Public <JQP@bar.com>
C: Subject:  The Next Meeting of the Board
C: To: Jones@xyz.com
C:
C: Bill:
C: The next meeting of the board of directors will be
C: on Tuesday.
C:                          John.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

Step 2 -- Relay Host to Destination Host

```
S: 220 xyz.com Simple Mail Transfer Service Ready
C: EHLO foo.com
S: 250 xyz.com is on the air
C: MAIL FROM:<@foo.com:JQP@bar.com>
S: 250 OK
C: RCPT TO:<Jones@XYZ.COM>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Received: from bar.com by foo.com ; Thu, 21 May 1998
C:     05:33:29 -0700
C: Date: Thu, 21 May 1998 05:33:22 -0700
C: From: John Q. Public <JQP@bar.com>
C: Subject:  The Next Meeting of the Board
C: To: Jones@xyz.com
C:
C: Bill:
C: The next meeting of the board of directors will be
C: on Tuesday.
C:                          John.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

### D.4 Verifying and Sending Scenario

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250-VRFY
S: 250 HELP
C: VRFY Crispin
S: 250 Mark Crispin <Admin.MRC@foo.com>
C: SEND FROM:<EAK@bar.com>
S: 250 OK
C: RCPT TO:<Admin.MRC@foo.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

## E. Other Gateway Issues

In general, gateways between the Internet and other mail systems SHOULD attempt to preserve any layering semantics across the boundaries between the two mail systems involved. Gateway-translation approaches that attempt to take shortcuts by mapping, (such as envelope information from one system to the message headers or body of another) have generally proven to be inadequate in important ways. Systems translating between environments that do not support both envelopes and headers and Internet mail must be written with the understanding that some information loss is almost inevitable.

## F. Deprecated Features of RFC 821

A few features of RFC 821 have proven to be problematic and SHOULD NOT be used in Internet mail.

### F.1 TURN

This command, described in RFC 821, raises important security issues since, in the absence of strong authentication of the host requesting that the client and server switch roles, it can easily be used to divert mail from its correct destination. Its use is deprecated; SMTP systems SHOULD NOT use it unless the server can authenticate the client.

### F.2 Source Routing

RFC 821 utilized the concept of explicit source routing to get mail from one host to another via a series of relays. The requirement to utilize source routes in regular mail traffic was eliminated by the introduction of the domain name system "MX" record and the last significant justification for them was eliminated by the introduction, in RFC 1123, of a clear requirement that addresses following an "@" must all be fully-qualified domain names. Consequently, the only remaining justifications for the use of source routes are support for very old SMTP clients or MUAs and in mail system debugging. They can, however, still be useful in the latter circumstance and for routing mail around serious, but temporary, problems such as problems with the relevant DNS records.

SMTP servers MUST continue to accept source route syntax as specified in the main body of this document and in RFC 1123. They MAY, if necessary, ignore the routes and utilize only the target domain in the address. If they do utilize the source route, the message MUST be sent to the first domain shown in the address. In particular, a server MUST NOT guess at shortcuts within the source route.

Clients SHOULD NOT utilize explicit source routing except under unusual circumstances, such as debugging or potentially relaying around firewall or mail system configuration errors.

### F.3 HELO

As discussed in sections 3.1 and 4.1.1, EHLO is strongly preferred to HELO when the server will accept the former. Servers must continue to accept and process HELO in order to support older clients.

### F.4 #-literals

RFC 821 provided for specifying an Internet address as a decimal integer host number prefixed by a pound sign, "#". In practice, that form has been obsolete since the introduction of TCP/IP. It is deprecated and MUST NOT be used.

### F.5 Dates and Years

When dates are inserted into messages by SMTP clients or servers (e.g., in trace fields), four-digit years MUST BE used. Two-digit years are deprecated; three-digit years were never permitted in the Internet mail system.





### F.6 Sending versus Mailing

In addition to specifying a mechanism for delivering messages to user's mailboxes, RFC 821 provided additional, optional, commands to deliver messages directly to the user's terminal screen. These commands (SEND, SAML, SOML) were rarely implemented, and changes in workstation technology and the introduction of other protocols may have rendered them obsolete even where they are implemented.

Clients SHOULD NOT provide SEND, SAML, or SOML as services. Servers MAY implement them. If they are implemented by servers, the implementation model specified in RFC 821 MUST be used and the command names MUST be published in the response to the EHLO command.

Full Copyright Statement



Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Network Working Group
Request for Comments: 2822
Obsoletes: 822
Category: Standards Track

P. Resnick, Editor
QUALCOMM Incorporated

April 2001

# Internet Message Format

## Status of this Memo



## Copyright Notice



## Abstract

This standard specifies a syntax for text messages that are sent between computer users, within the framework of "electronic mail" messages. This standard supersedes the one specified in Request For Comments (RFC) 822, "Standard for the Format of ARPA Internet Text Messages", updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs.

## Table of Contents

# 1. Introduction

## 1.1. Scope

This standard specifies a syntax for text messages that are sent between computer users, within the framework of "electronic mail" messages. This standard supersedes the one specified in Request For Comments (RFC) 822, "Standard for the Format of ARPA Internet Text Messages" [RFC822], updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs [STD3].

This standard specifies a syntax only for text messages. In particular, it makes no provision for the transmission of images, audio, or other sorts of structured data in electronic mail messages. There are several extensions published, such as the MIME document series [RFC2045, RFC2046, RFC2049], which describe mechanisms for the transmission of such data through electronic mail, either by extending the syntax provided here or by structuring such messages to conform to this syntax. Those mechanisms are outside of the scope of this standard.

In the context of electronic mail, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. (See [RFC2821] for a discussion of the envelope.) The contents comprise the object to be delivered to the recipient. This standard applies only to the format and some of the semantics of message contents. It contains no specification of the information in the envelope.

However, some message systems may use information from the contents to create the envelope. It is intended that this standard facilitate the acquisition of such information by programs.

This specification is intended as a definition of what message content format is to be passed between systems. Though some message systems locally store messages in this format (which eliminates the need for translation between formats) and others use formats that differ from the one specified in this standard, local storage is outside of the scope of this standard.





Note: This standard is not intended to dictate the internal formats used by sites, the specific message system features that they are expected to support, or any of the characteristics of user interface programs that create or read messages. In addition, this standard does not specify an encoding of the characters for either transport or storage; that is, it does not specify the number of bits used or how those bits are specifically transferred over the wire or stored on disk.

## 1.2. Notational conventions

### 1.2.1. Requirements notation

This document occasionally uses terms that appear in capital letters. When the terms "MUST", "SHOULD", "RECOMMENDED", "MUST NOT", "SHOULD NOT", and "MAY" appear capitalized, they are being used to indicate particular requirements of this specification. A discussion of the meanings of these terms appears in [RFC2119].

### 1.2.2. Syntactic notation

This standard uses the Augmented Backus-Naur Form (ABNF) notation specified in [RFC2234] for the formal definitions of the syntax of messages. Characters will be specified either by a decimal value (e.g., the value %d65 for uppercase A and %d97 for lowercase A) or by a case-insensitive literal value enclosed in quotation marks (e.g., "A" for either uppercase or lowercase A). See [RFC2234] for the full description of the notation.

## 1.3. Structure of this document

This document is divided into several sections.

This section, section 1, is a short introduction to the document.

Section 2 lays out the general description of a message and its constituent parts. This is an overview to help the reader understand some of the general principles used in the later portions of this document. Any examples in this section MUST NOT be taken as specification of the formal syntax of any part of a message.

Section 3 specifies formal ABNF rules for the structure of each part of a message (the syntax) and describes the relationship between those parts and their meaning in the context of a message (the semantics). That is, it describes the actual rules for the structure of each part of a message (the syntax) as well as a description of the parts and instructions on how they ought to be interpreted (the semantics). This includes analysis of the syntax and semantics of

subparts of messages that have specific structure. The syntax included in section 3 represents messages as they MUST be created. There are also notes in section 3 to indicate if any of the options specified in the syntax SHOULD be used over any of the others.

Both sections 2 and 3 describe messages that are legal to generate for purposes of this standard.

Section 4 of this document specifies an "obsolete" syntax. There are references in section 3 to these obsolete syntactic elements. The rules of the obsolete syntax are elements that have appeared in earlier revisions of this standard or have previously been widely used in Internet messages. As such, these elements MUST be interpreted by parsers of messages in order to be conformant to this standard. However, since items in this syntax have been determined to be non-interoperable or to cause significant problems for recipients of messages, they MUST NOT be generated by creators of conformant messages.

Section 5 details security considerations to take into account when implementing this standard.

Section 6 is a bibliography of references in this document.

Section 7 contains the editor's address.

Section 8 contains acknowledgements.

Appendix A lists examples of different sorts of messages. These examples are not exhaustive of the types of messages that appear on the Internet, but give a broad overview of certain syntactic forms.

Appendix B lists the differences between this standard and earlier standards for Internet messages.

Appendix C has copyright and intellectual property notices.

## 2. Lexical Analysis of Messages

### 2.1. General Description

At the most basic level, a message is a series of characters. A message that is conformant with this standard is comprised of characters with values in the range 1 through 127 and interpreted as US-ASCII characters [ASCII]. For brevity, this document sometimes refers to this range of characters as simply "US-ASCII characters".





Note: This standard specifies that messages are made up of characters in the US-ASCII range of 1 through 127. There are other documents, specifically the MIME document series [RFC2045, RFC2046, RFC2047, RFC2048, RFC2049], that extend this standard to allow for values outside of that range. Discussion of those mechanisms is not within the scope of this standard.

Messages are divided into lines of characters. A line is a series of characters that is delimited with the two characters carriage-return and line-feed; that is, the carriage return (CR) character (ASCII value 13) followed immediately by the line feed (LF) character (ASCII value 10). (The carriage-return/line-feed pair is usually written in this document as "CRLF".)

A message consists of header fields (collectively called "the header of the message") followed, optionally, by a body. The header is a sequence of lines of characters with special syntax as defined in this standard. The body is simply a sequence of characters that follows the header and is separated from the header by an empty line (i.e., a line with nothing preceding the CRLF).

#### 2.1.1. Line Length Limits

There are two limits that this standard places on the number of characters in a line. Each line of characters MUST be no more than 998 characters, and SHOULD be no more than 78 characters, excluding the CRLF.

The 998 character limit is due to limitations in many implementations which send, receive, or store Internet Message Format messages that simply cannot handle more than 998 characters on a line. Receiving implementations would do well to handle an arbitrarily large number of characters in a line for robustness sake. However, there are so many implementations which (in compliance with the transport requirements of [RFC2821]) do not accept messages containing more than 1000 character including the CR and LF per line, it is important for implementations not to create such messages.

The more conservative 78 character recommendation is to accommodate the many implementations of user interfaces that display these messages which may truncate, or disastrously wrap, the display of more than 78 characters per line, in spite of the fact that such implementations are non-conformant to the intent of this specification (and that of [RFC2821] if they actually cause information to be lost). Again, even though this limitation is put on messages, it is encumbant upon implementations which display messages

to handle an arbitrarily large number of characters in a line (certainly at least up to the 998 character limit) for the sake of robustness.

### 2.2. Header Fields

Header fields are lines composed of a field name, followed by a colon (":"), followed by a field body, and terminated by CRLF. A field name MUST be composed of printable US-ASCII characters (i.e., characters that have values between 33 and 126, inclusive), except colon. A field body may be composed of any US-ASCII characters, except for CR and LF. However, a field body may contain CRLF when used in header "folding" and "unfolding" as described in section 2.2.3. All field bodies MUST conform to the syntax described in sections 3 and 4 of this standard.

#### 2.2.1. Unstructured Header Field Bodies

Some field bodies in this standard are defined simply as "unstructured" (which is specified below as any US-ASCII characters, except for CR and LF) with no further restrictions. These are referred to as unstructured field bodies. Semantically, unstructured field bodies are simply to be treated as a single line of characters with no further processing (except for header "folding" and "unfolding" as described in section 2.2.3).

#### 2.2.2. Structured Header Field Bodies

Some field bodies in this standard have specific syntactical structure more restrictive than the unstructured field bodies described above. These are referred to as "structured" field bodies. Structured field bodies are sequences of specific lexical tokens as described in sections 3 and 4 of this standard. Many of these tokens are allowed (according to their syntax) to be introduced or end with comments (as described in section 3.2.3) as well as the space (SP, ASCII value 32) and horizontal tab (HTAB, ASCII value 9) characters (together known as the white space characters, WSP), and those WSP characters are subject to header "folding" and "unfolding" as described in section 2.2.3. Semantic analysis of structured field bodies is given along with their syntax.

#### 2.2.3. Long Header Fields

Each header field is logically a single line of characters comprising the field name, the colon, and the field body. For convenience however, and to deal with the 998/78 character limitations per line, the field body portion of a header field can be split into a multiple line representation; this is called "folding". The general rule is







that wherever this standard allows for folding white space (not simply WSP characters), a CRLF may be inserted before any WSP. For example, the header field:

```
Subject: This is a test
```

can be represented as:

```
Subject: This
 is a test
```

Note: Though structured field bodies are defined in such a way that folding can take place between many of the lexical tokens (and even within some of the lexical tokens), folding SHOULD be limited to placing the CRLF at higher-level syntactic breaks. For instance, if a field body is defined as comma-separated values, it is recommended that folding occur after the comma separating the structured items in preference to other places where the field could be folded, even if it is allowed elsewhere.

The process of moving from this folded multiple-line representation of a header field to its single line representation is called "unfolding". Unfolding is accomplished by simply removing any CRLF that is immediately followed by WSP. Each header field should be treated in its unfolded form for further syntactic and semantic evaluation.

### 2.3. Body

The body of a message is simply lines of US-ASCII characters. The only two limitations on the body are as follows:

- CR and LF MUST only occur together as CRLF; they MUST NOT appear independently in the body.

- Lines of characters in the body MUST be limited to 998 characters, and SHOULD be limited to 78 characters, excluding the CRLF.

Note: As was stated earlier, there are other standards documents, specifically the MIME documents [RFC2045, RFC2046, RFC2048, RFC2049] that extend this standard to allow for different sorts of message bodies. Again, these mechanisms are beyond the scope of this document.

# 3. Syntax

## 3.1. Introduction

The syntax as given in this section defines the legal syntax of Internet messages. Messages that are conformant to this standard MUST conform to the syntax in this section. If there are options in this section where one option SHOULD be generated, that is indicated either in the prose or in a comment next to the syntax.

For the defined expressions, a short description of the syntax and use is given, followed by the syntax in ABNF, followed by a semantic analysis. Primitive tokens that are used but otherwise unspecified come from [RFC2234].

In some of the definitions, there will be nonterminals whose names start with "obs-". These "obs-" elements refer to tokens defined in the obsolete syntax in section 4. In all cases, these productions are to be ignored for the purposes of generating legal Internet messages and MUST NOT be used as part of such a message. However, when interpreting messages, these tokens MUST be honored as part of the legal syntax. In this sense, section 3 defines a grammar for generation of messages, with "obs-" elements that are to be ignored, while section 4 adds grammar for interpretation of messages.

## 3.2. Lexical Tokens

The following rules are used to define an underlying lexical analyzer, which feeds tokens to the higher-level parsers. This section defines the tokens used in structured header field bodies.

Note: Readers of this standard need to pay special attention to how these lexical tokens are used in both the lower-level and higher-level syntax later in the document. Particularly, the white space tokens and the comment tokens defined in section 3.2.3 get used in the lower-level tokens defined here, and those lower-level tokens are in turn used as parts of the higher-level tokens defined later. Therefore, the white space and comments may be allowed in the higher-level tokens even though they may not explicitly appear in a particular definition.

### 3.2.1. Primitive Tokens

The following are primitive tokens referred to elsewhere in this standard, but not otherwise defined in [RFC2234]. Some of them will not appear anywhere else in the syntax, but they are convenient to refer to in other parts of this document.





Note: The "specials" below are just such an example. Though the specials token does not appear anywhere else in this standard, it is useful for implementers who use tools that lexically analyze messages. Each of the characters in specials can be used to indicate a tokenization point in lexical analysis.

```
NO-WS-CTL       =       %d1-8 /         ; US-ASCII control characters
                        %d11 /          ;  that do not include the
                        %d12 /          ;  carriage return, line feed,
                        %d14-31 /       ;  and white space characters
                        %d127

text            =       %d1-9 /         ; Characters excluding CR and LF
                        %d11 /
                        %d12 /
                        %d14-127 /
                        obs-text

specials        =       "(" / ")" /     ; Special characters used in
                        "<" / ">" /     ;  other parts of the syntax
                        "[" / "]" /
                        ":" / ";" /
                        "@" / "\" /
                        "," / "." /
                        DQUOTE
```

No special semantics are attached to these tokens. They are simply single characters.

### 3.2.2. Quoted characters

Some characters are reserved for special interpretation, such as delimiting lexical tokens. To permit use of these characters as uninterpreted data, a quoting mechanism is provided.

```
quoted-pair     =       ("\" text) / obs-qp
```

Where any quoted-pair appears, it is to be interpreted as the text character alone. That is to say, the "\" character that appears as part of a quoted-pair is semantically "invisible".

Note: The "\" character may appear in a message where it is not part of a quoted-pair. A "\" character that does not appear in a quoted-pair is not semantically invisible. The only places in this standard where quoted-pair currently appears are ccontent, qcontent, dcontent, no-fold-quote, and no-fold-literal.

### 3.2.3. Folding white space and comments

White space characters, including white space used in folding (described in section 2.2.3), may appear between many elements in header field bodies. Also, strings of characters that are treated as comments may be included in structured field bodies as characters enclosed in parentheses. The following defines the folding white space (FWS) and comment constructs.

Strings of characters enclosed in parentheses are considered comments so long as they do not appear within a "quoted-string", as defined in section 3.2.5. Comments may nest.

There are several places in this standard where comments and FWS may be freely inserted. To accommodate that syntax, an additional token for "CFWS" is defined for places where comments and/or FWS can occur. However, where CFWS occurs in this standard, it MUST NOT be inserted in such a way that any line of a folded header field is made up entirely of WSP characters and nothing else.

```
FWS             =       ([*WSP CRLF] 1*WSP) /   ; Folding white space
                        obs-FWS

ctext           =       NO-WS-CTL /     ; Non white space controls

                        %d33-39 /       ; The rest of the US-ASCII
                        %d42-91 /       ;  characters not including "(",
                        %d93-126        ;  ")", or "\"

ccontent        =       ctext / quoted-pair / comment

comment         =       "(" *([FWS] ccontent) [FWS] ")"

CFWS            =       *([FWS] comment) (([FWS] comment) / FWS)
```

Throughout this standard, where FWS (the folding white space token) appears, it indicates a place where header folding, as discussed in section 2.2.3, may take place. Wherever header folding appears in a message (that is, a header field body containing a CRLF followed by any WSP), header unfolding (removal of the CRLF) is performed before any further lexical analysis is performed on that header field according to this standard. That is to say, any CRLF that appears in FWS is semantically "invisible."

A comment is normally used in a structured field body to provide some human readable informational text. Since a comment is allowed to contain FWS, folding is permitted within the comment. Also note that since quoted-pair is allowed in a comment, the parentheses and





backslash characters may appear in a comment so long as they appear as a quoted-pair. Semantically, the enclosing parentheses are not part of the comment; the comment is what is contained between the two parentheses. As stated earlier, the "\" in any quoted-pair and the CRLF in any FWS that appears within the comment are semantically "invisible" and therefore not part of the comment either.

Runs of FWS, comment or CFWS that occur between lexical tokens in a structured field header are semantically interpreted as a single space character.

### 3.2.4. Atom

Several productions in structured header field bodies are simply strings of certain basic characters. Such productions are called atoms.

Some of the structured header field bodies also allow the period character (".", ASCII value 46) within runs of atext. An additional "dot-atom" token is defined for those purposes.

```
atext           =       ALPHA / DIGIT / ; Any character except controls,
                        "!" / "#" /     ;  SP, and specials.
                        "$" / "%" /     ;  Used for atoms
                        "&" / "'" /
                        "*" / "+" /
                        "-" / "/" /
                        "=" / "?" /
                        "^" / "_" /
                        "`" / "{" /
                        "|" / "}" /
                        "~"

atom            =       [CFWS] 1*atext [CFWS]

dot-atom        =       [CFWS] dot-atom-text [CFWS]

dot-atom-text   =       1*atext *("." 1*atext)
```

Both atom and dot-atom are interpreted as a single unit, comprised of the string of characters that make it up. Semantically, the optional comments and FWS surrounding the rest of the characters are not part of the atom; the atom is only the run of atext characters in an atom, or the atext and "." characters in a dot-atom.

### 3.2.5. Quoted strings

Strings of characters that include characters other than those allowed in atoms may be represented in a quoted string format, where the characters are surrounded by quote (DQUOTE, ASCII value 34) characters.

```
qtext           =       NO-WS-CTL /     ; Non white space controls

                        %d33 /          ; The rest of the US-ASCII
                        %d35-91 /       ;  characters not including "\"
                        %d93-126        ;  or the quote character

qcontent        =       qtext / quoted-pair

quoted-string   =       [CFWS]
                        DQUOTE *([FWS] qcontent) [FWS] DQUOTE
                        [CFWS]
```

A quoted-string is treated as a unit. That is, quoted-string is identical to atom, semantically. Since a quoted-string is allowed to contain FWS, folding is permitted. Also note that since quoted-pair is allowed in a quoted-string, the quote and backslash characters may appear in a quoted-string so long as they appear as a quoted-pair.

Semantically, neither the optional CFWS outside of the quote characters nor the quote characters themselves are part of the quoted-string; the quoted-string is what is contained between the two quote characters. As stated earlier, the "\" in any quoted-pair and the CRLF in any FWS/CFWS that appears within the quoted-string are semantically "invisible" and therefore not part of the quoted-string either.

### 3.2.6. Miscellaneous tokens

Three additional tokens are defined, word and phrase for combinations of atoms and/or quoted-strings, and unstructured for use in unstructured header fields and in some places within structured header fields.

```
word            =       atom / quoted-string

phrase          =       1*word / obs-phrase
```





```
utext           =       NO-WS-CTL /     ; Non white space controls
                        %d33-126 /      ; The rest of US-ASCII
                        obs-utext

unstructured    =       *([FWS] utext) [FWS]
```

### 3.3. Date and Time Specification

Date and time occur in several header fields. This section specifies the syntax for a full date and time specification. Though folding white space is permitted throughout the date-time specification, it is RECOMMENDED that a single space be used in each place that FWS appears (whether it is required or optional); some older implementations may not interpret other occurrences of folding white space correctly.

```
date-time       =       [ day-of-week "," ] date FWS time [CFWS]

day-of-week     =       ([FWS] day-name) / obs-day-of-week

day-name        =       "Mon" / "Tue" / "Wed" / "Thu" /
                        "Fri" / "Sat" / "Sun"

date            =       day month year

year            =       4*DIGIT / obs-year

month           =       (FWS month-name FWS) / obs-month

month-name      =       "Jan" / "Feb" / "Mar" / "Apr" /
                        "May" / "Jun" / "Jul" / "Aug" /
                        "Sep" / "Oct" / "Nov" / "Dec"

day             =       ([FWS] 1*2DIGIT) / obs-day

time            =       time-of-day FWS zone

time-of-day     =       hour ":" minute [ ":" second ]

hour            =       2DIGIT / obs-hour

minute          =       2DIGIT / obs-minute

second          =       2DIGIT / obs-second

zone            =       (( "+" / "-" ) 4DIGIT) / obs-zone
```

The day is the numeric day of the month. The year is any numeric year 1900 or later.

The time-of-day specifies the number of hours, minutes, and optionally seconds since midnight of the date indicated.

The date and time-of-day SHOULD express local time.

The zone specifies the offset from Coordinated Universal Time (UTC, formerly referred to as "Greenwich Mean Time") that the date and time-of-day represent. The "+" or "-" indicates whether the time-of-day is ahead of (i.e., east of) or behind (i.e., west of) Universal Time. The first two digits indicate the number of hours difference from Universal Time, and the last two digits indicate the number of minutes difference from Universal Time. (Hence, +hhmm means +(hh * 60 + mm) minutes, and -hhmm means -(hh * 60 + mm) minutes). The form "+0000" SHOULD be used to indicate a time zone at Universal Time. Though "-0000" also indicates Universal Time, it is used to indicate that the time was generated on a system that may be in a local time zone other than Universal Time and therefore indicates that the date-time contains no information about the local time zone.

A date-time specification MUST be semantically valid. That is, the day-of-the-week (if included) MUST be the day implied by the date, the numeric day-of-month MUST be between 1 and the number of days allowed for the specified month (in the specified year), the time-of-day MUST be in the range 00:00:00 through 23:59:60 (the number of seconds allowing for a leap second; see [STD12]), and the zone MUST be within the range -9959 through +9959.

### 3.4. Address Specification

Addresses occur in several message header fields to indicate senders and recipients of messages. An address may either be an individual mailbox, or a group of mailboxes.

```
address         =       mailbox / group

mailbox         =       name-addr / addr-spec

name-addr       =       [display-name] angle-addr

angle-addr      =       [CFWS] "<" addr-spec ">" [CFWS] / obs-angle-addr

group           =       display-name ":" [mailbox-list / CFWS] ";"
                        [CFWS]
```





```
display-name    =       phrase

mailbox-list    =       (mailbox *("," mailbox)) / obs-mbox-list

address-list    =       (address *("," address)) / obs-addr-list
```

A mailbox receives mail. It is a conceptual entity which does not necessarily pertain to file storage. For example, some sites may choose to print mail on a printer and deliver the output to the addressee's desk. Normally, a mailbox is comprised of two parts: (1) an optional display name that indicates the name of the recipient (which could be a person or a system) that could be displayed to the user of a mail application, and (2) an addr-spec address enclosed in angle brackets ("<" and ">"). There is also an alternate simple form of a mailbox where the addr-spec address appears alone, without the recipient's name or the angle brackets. The Internet addr-spec address is described in section 3.4.1.

Note: Some legacy implementations used the simple form where the addr-spec appears without the angle brackets, but included the name of the recipient in parentheses as a comment following the addr-spec. Since the meaning of the information in a comment is unspecified, implementations SHOULD use the full name-addr form of the mailbox, instead of the legacy form, to specify the display name associated with a mailbox. Also, because some legacy implementations interpret the comment, comments generally SHOULD NOT be used in address fields to avoid confusing such implementations.

When it is desirable to treat several mailboxes as a single unit (i.e., in a distribution list), the group construct can be used. The group construct allows the sender to indicate a named group of recipients. This is done by giving a display name for the group, followed by a colon, followed by a comma separated list of any number of mailboxes (including zero and one), and ending with a semicolon. Because the list of mailboxes can be empty, using the group construct is also a simple way to communicate to recipients that the message was sent to one or more named sets of recipients, without actually providing the individual mailbox address for each of those recipients.

#### 3.4.1. Addr-spec specification

An addr-spec is a specific Internet identifier that contains a locally interpreted string followed by the at-sign character ("@", ASCII value 64) followed by an Internet domain. The locally interpreted string is either a quoted-string or a dot-atom. If the string can be represented as a dot-atom (that is, it contains no characters other than atext characters or "." surrounded by atext

characters), then the dot-atom form SHOULD be used and the quoted-string form SHOULD NOT be used. Comments and folding white space SHOULD NOT be used around the "@" in the addr-spec.

```
addr-spec       =       local-part "@" domain

local-part      =       dot-atom / quoted-string / obs-local-part

domain          =       dot-atom / domain-literal / obs-domain

domain-literal  =       [CFWS] "[" *([FWS] dcontent) [FWS] "]" [CFWS]

dcontent        =       dtext / quoted-pair

dtext           =       NO-WS-CTL /     ; Non white space controls

                        %d33-90 /       ; The rest of the US-ASCII
                        %d94-126        ;  characters not including "[",
                                        ;  "]", or "\"
```

The domain portion identifies the point to which the mail is delivered. In the dot-atom form, this is interpreted as an Internet domain name (either a host name or a mail exchanger name) as described in [STD3, STD13, STD14]. In the domain-literal form, the domain is interpreted as the literal Internet address of the particular host. In both cases, how addressing is used and how messages are transported to a particular host is covered in the mail transport document [RFC2821]. These mechanisms are outside of the scope of this document.

The local-part portion is a domain dependent string. In addresses, it is simply interpreted on the particular host as a name of a particular mailbox.

## 3.5 Overall message syntax

A message consists of header fields, optionally followed by a message body. Lines in a message MUST be a maximum of 998 characters excluding the CRLF, but it is RECOMMENDED that lines be limited to 78 characters excluding the CRLF. (See section 2.1.1 for explanation.) In a message body, though all of the characters listed in the text rule MAY be used, the use of US-ASCII control characters (values 1 through 8, 11, 12, and 14 through 31) is discouraged since their interpretation by receivers for display is not guaranteed.







```
message         =       (fields / obs-fields)
                        [CRLF body]

body            =       *(*998text CRLF) *998text
```

The header fields carry most of the semantic information and are defined in section 3.6. The body is simply a series of lines of text which are uninterpreted for the purposes of this standard.

## 3.6. Field definitions

The header fields of a message are defined here. All header fields have the same general syntactic structure: A field name, followed by a colon, followed by the field body. The specific syntax for each header field is defined in the subsequent sections.

Note: In the ABNF syntax for each field in subsequent sections, each field name is followed by the required colon. However, for brevity sometimes the colon is not referred to in the textual description of the syntax. It is, nonetheless, required.

It is important to note that the header fields are not guaranteed to be in a particular order. They may appear in any order, and they have been known to be reordered occasionally when transported over the Internet. However, for the purposes of this standard, header fields SHOULD NOT be reordered when a message is transported or transformed. More importantly, the trace header fields and resent header fields MUST NOT be reordered, and SHOULD be kept in blocks prepended to the message. See sections 3.6.6 and 3.6.7 for more information.

The only required header fields are the origination date field and the originator address field(s). All other header fields are syntactically optional. More information is contained in the table following this definition.

```
fields          =       *(trace
                          *(resent-date /
                            resent-from /
                            resent-sender /
                            resent-to /
                            resent-cc /
                            resent-bcc /
                            resent-msg-id))
                        *(orig-date /
                        from /
                        sender /
                        reply-to /
```

to /
cc /
bcc /
message-id /
in-reply-to /
references /
subject /
comments /
keywords /
optional-field)

The following table indicates limits on the number of times each field may occur in a message header as well as any special limitations on the use of those fields. An asterisk next to a value in the minimum or maximum column indicates that a special restriction appears in the Notes column.

| Field | Min number | Max number | Notes |
|---|---|---|---|
| trace | 0 | unlimited | Block prepended - see 3.6.7 |
| resent-date | 0* | unlimited* | One per block, required if other resent fields present - see 3.6.6 |
| resent-from | 0 | unlimited* | One per block - see 3.6.6 |
| resent-sender | 0* | unlimited* | One per block, MUST occur with multi-address resent-from - see 3.6.6 |
| resent-to | 0 | unlimited* | One per block - see 3.6.6 |
| resent-cc | 0 | unlimited* | One per block - see 3.6.6 |
| resent-bcc | 0 | unlimited* | One per block - see 3.6.6 |
| resent-msg-id | 0 | unlimited* | One per block - see 3.6.6 |
| orig-date | 1 | 1 | |
| from | 1 | 1 | See sender and 3.6.2 |





| | | | |
|---|---|---|---|
| sender | 0* | 1 | MUST occur with multi-address from - see 3.6.2 |
| reply-to | 0 | 1 | |
| to | 0 | 1 | |
| cc | 0 | 1 | |
| bcc | 0 | 1 | |
| message-id | 0* | 1 | SHOULD be present - see 3.6.4 |
| in-reply-to | 0* | 1 | SHOULD occur in some replies - see 3.6.4 |
| references | 0* | 1 | SHOULD occur in some replies - see 3.6.4 |
| subject | 0 | 1 | |
| comments | 0 | unlimited | |
| keywords | 0 | unlimited | |
| optional-field | 0 | unlimited | |

The exact interpretation of each field is described in subsequent sections.

### 3.6.1. The origination date field

The origination date field consists of the field name "Date" followed by a date-time specification.

orig-date = "Date:" date-time CRLF

The origination date specifies the date and time at which the creator of the message indicated that the message was complete and ready to enter the mail delivery system. For instance, this might be the time that a user pushes the "send" or "submit" button in an application program. In any case, it is specifically not intended to convey the time that the message is actually transported, but rather the time at which the human or other creator of the message has put the message into its final form, ready for transport. (For example, a portable computer user who is not connected to a network might queue a message

for delivery. The origination date is intended to contain the date and time that the user queued the message, not the time when the user connected to the network to send the message.)

### 3.6.2. Originator fields

The originator fields of a message consist of the from field, the sender field (when applicable), and optionally the reply-to field. The from field consists of the field name "From" and a comma-separated list of one or more mailbox specifications. If the from field contains more than one mailbox specification in the mailbox-list, then the sender field, containing the field name "Sender" and a single mailbox specification, MUST appear in the message. In either case, an optional reply-to field MAY also be included, which contains the field name "Reply-To" and a comma-separated list of one or more addresses.

```
from            =       "From:" mailbox-list CRLF

sender          =       "Sender:" mailbox CRLF

reply-to        =       "Reply-To:" address-list CRLF
```

The originator fields indicate the mailbox(es) of the source of the message. The "From:" field specifies the author(s) of the message, that is, the mailbox(es) of the person(s) or system(s) responsible for the writing of the message. The "Sender:" field specifies the mailbox of the agent responsible for the actual transmission of the message. For example, if a secretary were to send a message for another person, the mailbox of the secretary would appear in the "Sender:" field and the mailbox of the actual author would appear in the "From:" field. If the originator of the message can be indicated by a single mailbox and the author and transmitter are identical, the "Sender:" field SHOULD NOT be used. Otherwise, both fields SHOULD appear.

The originator fields also provide the information required when replying to a message. When the "Reply-To:" field is present, it indicates the mailbox(es) to which the author of the message suggests that replies be sent. In the absence of the "Reply-To:" field, replies SHOULD by default be sent to the mailbox(es) specified in the "From:" field unless otherwise specified by the person composing the reply.

In all cases, the "From:" field SHOULD NOT contain any mailbox that does not belong to the author(s) of the message. See also section 3.6.3 for more information on forming the destination addresses for a reply.







### 3.6.3. Destination address fields

The destination fields of a message consist of three possible fields, each of the same form: The field name, which is either "To", "Cc", or "Bcc", followed by a comma-separated list of one or more addresses (either mailbox or group syntax).

```
to              =       "To:" address-list CRLF

cc              =       "Cc:" address-list CRLF

bcc             =       "Bcc:" (address-list / [CFWS]) CRLF
```

The destination fields specify the recipients of the message. Each destination field may have one or more addresses, and each of the addresses indicate the intended recipients of the message. The only difference between the three fields is how each is used.

The "To:" field contains the address(es) of the primary recipient(s) of the message.

The "Cc:" field (where the "Cc" means "Carbon Copy" in the sense of making a copy on a typewriter using carbon paper) contains the addresses of others who are to receive the message, though the content of the message may not be directed at them.

The "Bcc:" field (where the "Bcc" means "Blind Carbon Copy") contains addresses of recipients of the message whose addresses are not to be revealed to other recipients of the message. There are three ways in which the "Bcc:" field is used. In the first case, when a message containing a "Bcc:" field is prepared to be sent, the "Bcc:" line is removed even though all of the recipients (including those specified in the "Bcc:" field) are sent a copy of the message. In the second case, recipients specified in the "To:" and "Cc:" lines each are sent a copy of the message with the "Bcc:" line removed as above, but the recipients on the "Bcc:" line get a separate copy of the message containing a "Bcc:" line. (When there are multiple recipient addresses in the "Bcc:" field, some implementations actually send a separate copy of the message to each recipient with a "Bcc:" containing only the address of that particular recipient.) Finally, since a "Bcc:" field may contain no addresses, a "Bcc:" field can be sent without any addresses indicating to the recipients that blind copies were sent to someone. Which method to use with "Bcc:" fields is implementation dependent, but refer to the "Security Considerations" section of this document for a discussion of each.

When a message is a reply to another message, the mailboxes of the authors of the original message (the mailboxes in the "From:" field) or mailboxes specified in the "Reply-To:" field (if it exists) MAY appear in the "To:" field of the reply since these would normally be the primary recipients of the reply. If a reply is sent to a message that has destination fields, it is often desirable to send a copy of the reply to all of the recipients of the message, in addition to the author. When such a reply is formed, addresses in the "To:" and "Cc:" fields of the original message MAY appear in the "Cc:" field of the reply, since these are normally secondary recipients of the reply. If a "Bcc:" field is present in the original message, addresses in that field MAY appear in the "Bcc:" field of the reply, but SHOULD NOT appear in the "To:" or "Cc:" fields.

Note: Some mail applications have automatic reply commands that include the destination addresses of the original message in the destination addresses of the reply. How those reply commands behave is implementation dependent and is beyond the scope of this document. In particular, whether or not to include the original destination addresses when the original message had a "Reply-To:" field is not addressed here.

### 3.6.4. Identification fields

Though optional, every message SHOULD have a "Message-ID:" field. Furthermore, reply messages SHOULD have "In-Reply-To:" and "References:" fields as appropriate, as described below.

The "Message-ID:" field contains a single unique message identifier. The "References:" and "In-Reply-To:" field each contain one or more unique message identifiers, optionally separated by CFWS.

The message identifier (msg-id) is similar in syntax to an angle-addr construct without the internal CFWS.

```
message-id      =       "Message-ID:" msg-id CRLF

in-reply-to     =       "In-Reply-To:" 1*msg-id CRLF

references      =       "References:" 1*msg-id CRLF

msg-id          =       [CFWS] "<" id-left "@" id-right ">" [CFWS]

id-left         =       dot-atom-text / no-fold-quote / obs-id-left

id-right        =       dot-atom-text / no-fold-literal / obs-id-right

no-fold-quote   =       DQUOTE *(qtext / quoted-pair) DQUOTE
```





```
no-fold-literal =       "[" *(dtext / quoted-pair) "]"
```

The "Message-ID:" field provides a unique message identifier that refers to a particular version of a particular message. The uniqueness of the message identifier is guaranteed by the host that generates it (see below). This message identifier is intended to be machine readable and not necessarily meaningful to humans. A message identifier pertains to exactly one instantiation of a particular message; subsequent revisions to the message each receive new message identifiers.

Note: There are many instances when messages are "changed", but those changes do not constitute a new instantiation of that message, and therefore the message would not get a new message identifier. For example, when messages are introduced into the transport system, they are often prepended with additional header fields such as trace fields (described in section 3.6.7) and resent fields (described in section 3.6.6). The addition of such header fields does not change the identity of the message and therefore the original "Message-ID:" field is retained. In all cases, it is the meaning that the sender of the message wishes to convey (i.e., whether this is the same message or a different message) that determines whether or not the "Message-ID:" field changes, not any particular syntactic difference that appears (or does not appear) in the message.

The "In-Reply-To:" and "References:" fields are used when creating a reply to a message. They hold the message identifier of the original message and the message identifiers of other messages (for example, in the case of a reply to a message which was itself a reply). The "In-Reply-To:" field may be used to identify the message (or messages) to which the new message is a reply, while the "References:" field may be used to identify a "thread" of conversation.

When creating a reply to a message, the "In-Reply-To:" and "References:" fields of the resultant message are constructed as follows:

The "In-Reply-To:" field will contain the contents of the "Message-ID:" field of the message to which this one is a reply (the "parent message"). If there is more than one parent message, then the "In-Reply-To:" field will contain the contents of all of the parents' "Message-ID:" fields. If there is no "Message-ID:" field in any of the parent messages, then the new message will have no "In-Reply-To:" field.

The "References:" field will contain the contents of the parent's "References:" field (if any) followed by the contents of the parent's "Message-ID:" field (if any). If the parent message does not contain a "References:" field but does have an "In-Reply-To:" field containing a single message identifier, then the "References:" field will contain the contents of the parent's "In-Reply-To:" field followed by the contents of the parent's "Message-ID:" field (if any). If the parent has none of the "References:", "In-Reply-To:", or "Message-ID:" fields, then the new message will have no "References:" field.

Note: Some implementations parse the "References:" field to display the "thread of the discussion". These implementations assume that each new message is a reply to a single parent and hence that they can walk backwards through the "References:" field to find the parent of each message listed there. Therefore, trying to form a "References:" field for a reply that has multiple parents is discouraged and how to do so is not defined in this document.

The message identifier (msg-id) itself MUST be a globally unique identifier for a message. The generator of the message identifier MUST guarantee that the msg-id is unique. There are several algorithms that can be used to accomplish this. Since the msg-id has a similar syntax to angle-addr (identical except that comments and folding white space are not allowed), a good method is to put the domain name (or a domain literal IP address) of the host on which the message identifier was created on the right hand side of the "@", and put a combination of the current absolute date and time along with some other currently unique (perhaps sequential) identifier available on the system (for example, a process id number) on the left hand side. Using a date on the left hand side and a domain name or domain literal on the right hand side makes it possible to guarantee uniqueness since no two hosts use the same domain name or IP address at the same time. Though other algorithms will work, it is RECOMMENDED that the right hand side contain some domain identifier (either of the host itself or otherwise) such that the generator of the message identifier can guarantee the uniqueness of the left hand side within the scope of that domain.

Semantically, the angle bracket characters are not part of the msg-id; the msg-id is what is contained between the two angle bracket characters.







### 3.6.5. Informational fields

The informational fields are all optional. The "Keywords:" field contains a comma-separated list of one or more words or quoted-strings. The "Subject:" and "Comments:" fields are unstructured fields as defined in section 2.2.1, and therefore may contain text or folding white space.

```
subject         =       "Subject:" unstructured CRLF

comments        =       "Comments:" unstructured CRLF

keywords        =       "Keywords:" phrase *("," phrase) CRLF
```

These three fields are intended to have only human-readable content with information about the message. The "Subject:" field is the most common and contains a short string identifying the topic of the message. When used in a reply, the field body MAY start with the string "Re: " (from the Latin "res", in the matter of) followed by the contents of the "Subject:" field body of the original message. If this is done, only one instance of the literal string "Re: " ought to be used since use of other strings or more than one instance can lead to undesirable consequences. The "Comments:" field contains any additional comments on the text of the body of the message. The "Keywords:" field contains a comma-separated list of important words and phrases that might be useful for the recipient.

### 3.6.6. Resent fields

Resent fields SHOULD be added to any message that is reintroduced by a user into the transport system. A separate set of resent fields SHOULD be added each time this is done. All of the resent fields corresponding to a particular resending of the message SHOULD be together. Each new set of resent fields is prepended to the message; that is, the most recent set of resent fields appear earlier in the message. No other fields in the message are changed when resent fields are added.

Each of the resent fields corresponds to a particular field elsewhere in the syntax. For instance, the "Resent-Date:" field corresponds to the "Date:" field and the "Resent-To:" field corresponds to the "To:" field. In each case, the syntax for the field body is identical to the syntax given previously for the corresponding field.

When resent fields are used, the "Resent-From:" and "Resent-Date:" fields MUST be sent. The "Resent-Message-ID:" field SHOULD be sent. "Resent-Sender:" SHOULD NOT be used if "Resent-Sender:" would be identical to "Resent-From:".

```
resent-date     =       "Resent-Date:" date-time CRLF

resent-from     =       "Resent-From:" mailbox-list CRLF

resent-sender   =       "Resent-Sender:" mailbox CRLF

resent-to       =       "Resent-To:" address-list CRLF

resent-cc       =       "Resent-Cc:" address-list CRLF

resent-bcc      =       "Resent-Bcc:" (address-list / [CFWS]) CRLF

resent-msg-id   =       "Resent-Message-ID:" msg-id CRLF
```

Resent fields are used to identify a message as having been reintroduced into the transport system by a user. The purpose of using resent fields is to have the message appear to the final recipient as if it were sent directly by the original sender, with all of the original fields remaining the same. Each set of resent fields correspond to a particular resending event. That is, if a message is resent multiple times, each set of resent fields gives identifying information for each individual time. Resent fields are strictly informational. They MUST NOT be used in the normal processing of replies or other such automatic actions on messages.

Note: Reintroducing a message into the transport system and using resent fields is a different operation from "forwarding". "Forwarding" has two meanings: One sense of forwarding is that a mail reading program can be told by a user to forward a copy of a message to another person, making the forwarded message the body of the new message. A forwarded message in this sense does not appear to have come from the original sender, but is an entirely new message from the forwarder of the message. On the other hand, forwarding is also used to mean when a mail transport program gets a message and forwards it on to a different destination for final delivery. Resent header fields are not intended for use with either type of forwarding.

The resent originator fields indicate the mailbox of the person(s) or system(s) that resent the message. As with the regular originator fields, there are two forms: a simple "Resent-From:" form which contains the mailbox of the individual doing the resending, and the more complex form, when one individual (identified in the "Resent-Sender:" field) resends a message on behalf of one or more others (identified in the "Resent-From:" field).

Note: When replying to a resent message, replies behave just as they would with any other message, using the original "From:",







"Reply-To:", "Message-ID:", and other fields. The resent fields are only informational and MUST NOT be used in the normal processing of replies.

The "Resent-Date:" indicates the date and time at which the resent message is dispatched by the resender of the message. Like the "Date:" field, it is not the date and time that the message was actually transported.

The "Resent-To:", "Resent-Cc:", and "Resent-Bcc:" fields function identically to the "To:", "Cc:", and "Bcc:" fields respectively, except that they indicate the recipients of the resent message, not the recipients of the original message.

The "Resent-Message-ID:" field provides a unique identifier for the resent message.

### 3.6.7. Trace fields

The trace fields are a group of header fields consisting of an optional "Return-Path:" field, and one or more "Received:" fields. The "Return-Path:" header field contains a pair of angle brackets that enclose an optional addr-spec. The "Received:" field contains a (possibly empty) list of name/value pairs followed by a semicolon and a date-time specification. The first item of the name/value pair is defined by item-name, and the second item is either an addr-spec, an atom, a domain, or a msg-id. Further restrictions may be applied to the syntax of the trace fields by standards that provide for their use, such as [RFC2821].

```
trace           =       [return]
                        1*received

return          =       "Return-Path:" path CRLF

path            =       ([CFWS] "<" ([CFWS] / addr-spec) ">" [CFWS]) /
                        obs-path

received        =       "Received:" name-val-list ";" date-time CRLF

name-val-list   =       [CFWS] [name-val-pair *(CFWS name-val-pair)]

name-val-pair   =       item-name CFWS item-value

item-name       =       ALPHA *(["-"] (ALPHA / DIGIT))

item-value      =       1*angle-addr / addr-spec /
                         atom / domain / msg-id
```

A full discussion of the Internet mail use of trace fields is contained in [RFC2821]. For the purposes of this standard, the trace fields are strictly informational, and any formal interpretation of them is outside of the scope of this document.

### 3.6.8. Optional fields

Fields may appear in messages that are otherwise unspecified in this standard. They MUST conform to the syntax of an optional-field. This is a field name, made up of the printable US-ASCII characters except SP and colon, followed by a colon, followed by any text which conforms to unstructured.

The field names of any optional-field MUST NOT be identical to any field name specified elsewhere in this standard.

```
optional-field  =       field-name ":" unstructured CRLF

field-name      =       1*ftext

ftext           =       %d33-57 /               ; Any character except
                        %d59-126                ;  controls, SP, and
                                                ;  ":".
```

For the purposes of this standard, any optional field is uninterpreted.

## 4. Obsolete Syntax

Earlier versions of this standard allowed for different (usually more liberal) syntax than is allowed in this version. Also, there have been syntactic elements used in messages on the Internet whose interpretation have never been documented. Though some of these syntactic forms MUST NOT be generated according to the grammar in section 3, they MUST be accepted and parsed by a conformant receiver. This section documents many of these syntactic elements. Taking the grammar in section 3 and adding the definitions presented in this section will result in the grammar to use for interpretation of messages.

Note: This section identifies syntactic forms that any implementation MUST reasonably interpret. However, there are certainly Internet messages which do not conform to even the additional syntax given in this section. The fact that a particular form does not appear in any section of this document is not justification for computer programs to crash or for malformed data to be irretrievably lost by any implementation. To repeat an example, though this document requires lines in messages to be no longer than 998 characters, silently





discarding the 999th and subsequent characters in a line without warning would still be bad behavior for an implementation. It is up to the implementation to deal with messages robustly.

One important difference between the obsolete (interpreting) and the current (generating) syntax is that in structured header field bodies (i.e., between the colon and the CRLF of any structured header field), white space characters, including folding white space, and comments can be freely inserted between any syntactic tokens. This allows many complex forms that have proven difficult for some implementations to parse.

Another key difference between the obsolete and the current syntax is that the rule in section 3.2.3 regarding lines composed entirely of white space in comments and folding white space does not apply. See the discussion of folding white space in section 4.2 below.

Finally, certain characters that were formerly allowed in messages appear in this section. The NUL character (ASCII value 0) was once allowed, but is no longer for compatibility reasons. CR and LF were allowed to appear in messages other than as CRLF; this use is also shown here.

Other differences in syntax and semantics are noted in the following sections.

### 4.1. Miscellaneous obsolete tokens

These syntactic elements are used elsewhere in the obsolete syntax or in the main syntax. The obs-char and obs-qp elements each add ASCII value 0. Bare CR and bare LF are added to obs-text and obs-utext. The period character is added to obs-phrase. The obs-phrase-list provides for "empty" elements in a comma-separated list of phrases.

Note: The "period" (or "full stop") character (".") in obs-phrase is not a form that was allowed in earlier versions of this or any other standard. Period (nor any other character from specials) was not allowed in phrase because it introduced a parsing difficulty distinguishing between phrases and portions of an addr-spec (see section 4.4). It appears here because the period character is currently used in many messages in the display-name portion of addresses, especially for initials in names, and therefore must be interpreted properly. In the future, period may appear in the regular syntax of phrase.

```
obs-qp          =       "\" (%d0-127)

obs-text        =       *LF *CR *(obs-char *LF *CR)
```

```
obs-char        =       %d0-9 / %d11 /          ; %d0-127 except CR and
                        %d12 / %d14-127         ;  LF

obs-utext       =       obs-text

obs-phrase      =       word *(word / "." / CFWS)

obs-phrase-list =       phrase / 1*([phrase] [CFWS] "," [CFWS]) [phrase]
```

Bare CR and bare LF appear in messages with two different meanings. In many cases, bare CR or bare LF are used improperly instead of CRLF to indicate line separators. In other cases, bare CR and bare LF are used simply as ASCII control characters with their traditional ASCII meanings.

### 4.2. Obsolete folding white space

In the obsolete syntax, any amount of folding white space MAY be inserted where the obs-FWS rule is allowed. This creates the possibility of having two consecutive "folds" in a line, and therefore the possibility that a line which makes up a folded header field could be composed entirely of white space.

```
obs-FWS         =       1*WSP *(CRLF 1*WSP)
```

### 4.3. Obsolete Date and Time

The syntax for the obsolete date format allows a 2 digit year in the date field and allows for a list of alphabetic time zone specifications that were used in earlier versions of this standard. It also permits comments and folding white space between many of the tokens.

```
obs-day-of-week =       [CFWS] day-name [CFWS]

obs-year        =       [CFWS] 2*DIGIT [CFWS]

obs-month       =       CFWS month-name CFWS

obs-day         =       [CFWS] 1*2DIGIT [CFWS]

obs-hour        =       [CFWS] 2DIGIT [CFWS]

obs-minute      =       [CFWS] 2DIGIT [CFWS]

obs-second      =       [CFWS] 2DIGIT [CFWS]

obs-zone        =       "UT" / "GMT" /          ; Universal Time
```





```
                                ; North American UT
                                ; offsets
                "EST" / "EDT" / ; Eastern:  - 5/ - 4
                "CST" / "CDT" / ; Central:  - 6/ - 5
                "MST" / "MDT" / ; Mountain: - 7/ - 6
                "PST" / "PDT" / ; Pacific:  - 8/ - 7

                %d65-73 /       ; Military zones - "A"
                %d75-90 /       ; through "I" and "K"
                %d97-105 /      ; through "Z", both
                %d107-122       ; upper and lower case
```

Where a two or three digit year occurs in a date, the year is to be interpreted as follows: If a two digit year is encountered whose value is between 00 and 49, the year is interpreted by adding 2000, ending up with a value between 2000 and 2049. If a two digit year is encountered with a value between 50 and 99, or any three digit year is encountered, the year is interpreted by adding 1900.

In the obsolete time zone, "UT" and "GMT" are indications of "Universal Time" and "Greenwich Mean Time" respectively and are both semantically identical to "+0000".

The remaining three character zones are the US time zones. The first letter, "E", "C", "M", or "P" stands for "Eastern", "Central", "Mountain" and "Pacific". The second letter is either "S" for "Standard" time, or "D" for "Daylight" (or summer) time. Their interpretations are as follows:

EDT is semantically equivalent to -0400
EST is semantically equivalent to -0500
CDT is semantically equivalent to -0500
CST is semantically equivalent to -0600
MDT is semantically equivalent to -0600
MST is semantically equivalent to -0700
PDT is semantically equivalent to -0700
PST is semantically equivalent to -0800

The 1 character military time zones were defined in a non-standard way in [RFC822] and are therefore unpredictable in their meaning. The original definitions of the military zones "A" through "I" are equivalent to "+0100" through "+0900" respectively; "K", "L", and "M" are equivalent to "+1000", "+1100", and "+1200" respectively; "N" through "Y" are equivalent to "-0100" through "-1200" respectively; and "Z" is equivalent to "+0000". However, because of the error in [RFC822], they SHOULD all be considered equivalent to "-0000" unless there is out-of-band information confirming their meaning.

Other multi-character (usually between 3 and 5) alphabetic time zones have been used in Internet messages. Any such time zone whose meaning is not known SHOULD be considered equivalent to "-0000" unless there is out-of-band information confirming their meaning.

### 4.4. Obsolete Addressing

There are three primary differences in addressing. First, mailbox addresses were allowed to have a route portion before the addr-spec when enclosed in "<" and ">". The route is simply a comma-separated list of domain names, each preceded by "@", and the list terminated by a colon. Second, CFWS were allowed between the period-separated elements of local-part and domain (i.e., dot-atom was not used). In addition, local-part is allowed to contain quoted-string in addition to just atom. Finally, mailbox-list and address-list were allowed to have "null" members. That is, there could be two or more commas in such a list with nothing in between them.

```
obs-angle-addr  =       [CFWS] "<" [obs-route] addr-spec ">" [CFWS]

obs-route       =       [CFWS] obs-domain-list ":" [CFWS]

obs-domain-list =       "@" domain *(*(CFWS / "," ) [CFWS] "@" domain)

obs-local-part  =       word *("." word)

obs-domain      =       atom *("." atom)

obs-mbox-list   =       1*([mailbox] [CFWS] "," [CFWS]) [mailbox]

obs-addr-list   =       1*([address] [CFWS] "," [CFWS]) [address]
```

When interpreting addresses, the route portion SHOULD be ignored.

### 4.5. Obsolete header fields

Syntactically, the primary difference in the obsolete field syntax is that it allows multiple occurrences of any of the fields and they may occur in any order. Also, any amount of white space is allowed before the ":" at the end of the field name.

```
obs-fields      =       *(obs-return /
                        obs-received /
                        obs-orig-date /
                        obs-from /
                        obs-sender /
                        obs-reply-to /
                        obs-to /
                        obs-cc /
                        obs-bcc /
                        obs-message-id /
                        obs-in-reply-to /
                        obs-references /
                        obs-subject /
                        obs-comments /
                        obs-keywords /
                        obs-resent-date /
                        obs-resent-from /
                        obs-resent-send /
                        obs-resent-rply /
                        obs-resent-to /
                        obs-resent-cc /
                        obs-resent-bcc /
                        obs-resent-mid /
                        obs-optional)
```

Except for destination address fields (described in section 4.5.3), the interpretation of multiple occurrences of fields is unspecified. Also, the interpretation of trace fields and resent fields which do not occur in blocks prepended to the message is unspecified as well. Unless otherwise noted in the following sections, interpretation of other fields is identical to the interpretation of their non-obsolete counterparts in section 3.

#### 4.5.1. Obsolete origination date field

```
obs-orig-date   =       "Date" *WSP ":" date-time CRLF
```

#### 4.5.2. Obsolete originator fields

```
obs-from        =       "From" *WSP ":" mailbox-list CRLF

obs-sender      =       "Sender" *WSP ":" mailbox CRLF

obs-reply-to    =       "Reply-To" *WSP ":" mailbox-list CRLF
```

#### 4.5.3. Obsolete destination address fields

```
obs-to          =       "To" *WSP ":" address-list CRLF

obs-cc          =       "Cc" *WSP ":" address-list CRLF

obs-bcc         =       "Bcc" *WSP ":" (address-list / [CFWS]) CRLF
```

When multiple occurrences of destination address fields occur in a message, they SHOULD be treated as if the address-list in the first occurrence of the field is combined with the address lists of the subsequent occurrences by adding a comma and concatenating.

### 4.5.4. Obsolete identification fields

The obsolete "In-Reply-To:" and "References:" fields differ from the current syntax in that they allow phrase (words or quoted strings) to appear. The obsolete forms of the left and right sides of msg-id allow interspersed CFWS, making them syntactically identical to local-part and domain respectively.

```
obs-message-id  =       "Message-ID" *WSP ":" msg-id CRLF

obs-in-reply-to =       "In-Reply-To" *WSP ":" *(phrase / msg-id) CRLF

obs-references  =       "References" *WSP ":" *(phrase / msg-id) CRLF

obs-id-left     =       local-part

obs-id-right    =       domain
```

For purposes of interpretation, the phrases in the "In-Reply-To:" and "References:" fields are ignored.

Semantically, none of the optional CFWS surrounding the local-part and the domain are part of the obs-id-left and obs-id-right respectively.

### 4.5.5. Obsolete informational fields

```
obs-subject     =       "Subject" *WSP ":" unstructured CRLF

obs-comments    =       "Comments" *WSP ":" unstructured CRLF

obs-keywords    =       "Keywords" *WSP ":" obs-phrase-list CRLF
```

### 4.5.6. Obsolete resent fields

The obsolete syntax adds a "Resent-Reply-To:" field, which consists of the field name, the optional comments and folding white space, the colon, and a comma separated list of addresses.

```
obs-resent-from =       "Resent-From" *WSP ":" mailbox-list CRLF

obs-resent-send =       "Resent-Sender" *WSP ":" mailbox CRLF
```





```
obs-resent-date =       "Resent-Date" *WSP ":" date-time CRLF

obs-resent-to   =       "Resent-To" *WSP ":" address-list CRLF

obs-resent-cc   =       "Resent-Cc" *WSP ":" address-list CRLF

obs-resent-bcc  =       "Resent-Bcc" *WSP ":"
                         (address-list / [CFWS]) CRLF

obs-resent-mid  =       "Resent-Message-ID" *WSP ":" msg-id CRLF

obs-resent-rply =       "Resent-Reply-To" *WSP ":" address-list CRLF
```

As with other resent fields, the "Resent-Reply-To:" field is to be treated as trace information only.

### 4.5.7. Obsolete trace fields

The obs-return and obs-received are again given here as template definitions, just as return and received are in section 3. Their full syntax is given in [RFC2821].

```
obs-return      =       "Return-Path" *WSP ":" path CRLF

obs-received    =       "Received" *WSP ":" name-val-list CRLF

obs-path        =       obs-angle-addr
```

### 4.5.8. Obsolete optional fields

```
obs-optional    =       field-name *WSP ":" unstructured CRLF
```

## 5. Security Considerations

Care needs to be taken when displaying messages on a terminal or terminal emulator. Powerful terminals may act on escape sequences and other combinations of ASCII control characters with a variety of consequences. They can remap the keyboard or permit other modifications to the terminal which could lead to denial of service or even damaged data. They can trigger (sometimes programmable) answerback messages which can allow a message to cause commands to be issued on the recipient's behalf. They can also effect the operation of terminal attached devices such as printers. Message viewers may wish to strip potentially dangerous terminal escape sequences from the message prior to display. However, other escape sequences appear in messages for useful purposes (cf. [RFC2045, RFC2046, RFC2047, RFC2048, RFC2049, ISO2022]) and therefore should not be stripped indiscriminately.

Transmission of non-text objects in messages raises additional security issues. These issues are discussed in [RFC2045, RFC2046, RFC2047, RFC2048, RFC2049].

Many implementations use the "Bcc:" (blind carbon copy) field described in section 3.6.3 to facilitate sending messages to recipients without revealing the addresses of one or more of the addressees to the other recipients. Mishandling this use of "Bcc:" has implications for confidential information that might be revealed, which could eventually lead to security problems through knowledge of even the existence of a particular mail address. For example, if using the first method described in section 3.6.3, where the "Bcc:" line is removed from the message, blind recipients have no explicit indication that they have been sent a blind copy, except insofar as their address does not appear in the message header. Because of this, one of the blind addressees could potentially send a reply to all of the shown recipients and accidentally reveal that the message went to the blind recipient. When the second method from section 3.6.3 is used, the blind recipient's address appears in the "Bcc:" field of a separate copy of the message. If the "Bcc:" field sent contains all of the blind addressees, all of the "Bcc:" recipients will be seen by each "Bcc:" recipient. Even if a separate message is sent to each "Bcc:" recipient with only the individual's address, implementations still need to be careful to process replies to the message as per section 3.6.3 so as not to accidentally reveal the blind recipient to other recipients.

## 6. Bibliography

[ASCII] American National Standards Institute (ANSI), Coded Character Set - 7-Bit American National Standard Code for Information Interchange, ANSI X3.4, 1986.

[ISO2022] International Organization for Standardization (ISO), Information processing - ISO 7-bit and 8-bit coded character sets - Code extension techniques, Third edition - 1986-05-01, ISO 2022, 1986.

[RFC822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", RFC 822, August 1982.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.





[RFC2047] Moore, K., "Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[RFC2048] Freed, N., Klensin, J. and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Format of Internet Message Bodies", RFC 2048, November 1996.

[RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D., Editor, and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2821] Klensin, J., Editor, "Simple Mail Transfer Protocol", RFC 2821, March 2001.

[STD3] Braden, R., "Host Requirements", STD 3, RFC 1122 and RFC 1123, October 1989.

[STD12] Mills, D., "Network Time Protocol", STD 12, RFC 1119, September 1989.

[STD13] Mockapetris, P., "Domain Name System", STD 13, RFC 1034 and RFC 1035, November 1987.

[STD14] Partridge, C., "Mail Routing and the Domain System", STD 14, RFC 974, January 1986.

## 7. Editor's Address

Peter W. Resnick
QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
USA

Phone: +1 858 651 4478
Fax: +1 858 651 1102
EMail: presnick@qualcomm.com

## 8. Acknowledgements

Many people contributed to this document. They included folks who participated in the Detailed Revision and Update of Messaging Standards (DRUMS) Working Group of the Internet Engineering Task Force (IETF), the chair of DRUMS, the Area Directors of the IETF, and people who simply sent their comments in via e-mail. The editor is deeply indebted to them all and thanks them sincerely. The below list includes everyone who sent e-mail concerning this document. Hopefully, everyone who contributed is named here:

Matti Aarnio
Tanaka Akira
Russ Allbery
Eric Allman
Harald Tveit Alvestrand
Ran Atkinson
Jos Backus
Bruce Balden
Dave Barr
Alan Barrett
John Beck
J. Robert von Behren
Jos den Bekker
D. J. Bernstein
James Berriman
Norbert Bollow
Raj Bose
Antony Bowesman
Scott Bradner
Randy Bush
Tom Byrer
Bruce Campbell
Larry Campbell
W. J. Carpenter
Michael Chapman
Richard Clayton
Maurizio Codogno
Jim Conklin
R. Kelley Cook
Steve Coya
Mark Crispin
Dave Crocker
Matt Curtin
Michael D'Errico
Cyrus Daboo
Jutta Degener
Mark Delany
Barry Finkel
Erik Forsberg
Chuck Foster
Paul Fox
Klaus M. Frank
Ned Freed
Jochen Friedrich
Randall C. Gellens
Sukvinder Singh Gill
Tim Goodwin
Philip Guenther
Tony Hansen
John Hawkinson
Philip Hazel
Kai Henningsen
Robert Herriot
Paul Hethmon
Jim Hill
Paul E. Hoffman
Steve Hole
Kari Hurtta
Marco S. Hyman
Ofer Inbar
Olle Jarnefors
Kevin Johnson
Sudish Joseph
Maynard Kang
Prabhat Keni
John C. Klensin
Graham Klyne
Brad Knowles
Shuhei Kobayashi
Peter Koch
Dan Kohn
Christian Kuhtz
Anand Kumria
Steen Larsen
Larry Masinter
Denis McKeon
William P McQuillan
Alexey Melnikov
Perry E. Metzger
Steven Miller
Keith Moore
John Gardiner Myers
Chris Newman
John W. Noerenberg
Eric Norman
Mike O'Dell
Larry Osterman
Paul Overell
Jacob Palme
Michael A. Patton
Uzi Paz
Michael A. Quinlan
Eric S. Raymond
Sam Roberts
Hugh Sasse
Bart Schaefer
Tom Scola
Wolfgang Segmuller
Nick Shelness
John Stanley
Einar Stefferud
Jeff Stephenson
Bernard Stern
Peter Sylvester
Mark Symons
Eric Thomas
Lee Thompson
Karel De Vriendt
Matthew Wall
Rolf Weber
Brent B. Welch
Steve Dorner
Harold A. Driscoll
Michael Elkins
Robert Elz
Johnny Eriksson
Erik E. Fair
Roger Fajman
Patrik Faltstrom
Claus Andre Farber
Eliot Lear
Barry Leiba
Jay Levitt
Lars-Johan Liman
Charles Lindsey
Pete Loshin
Simon Lyall
Bill Manning
John Martin
Dan Wing
Jack De Winter
Gregory J. Woodhouse
Greg A. Woods
Kazu Yamamoto
Alain Zahm
Jamie Zawinski
Timothy S. Zurcher

## Appendix A. Example messages

This section presents a selection of messages. These are intended to assist in the implementation of this standard, but should not be taken as normative; that is to say, although the examples in this section were carefully reviewed, if there happens to be a conflict between these examples and the syntax described in sections 3 and 4 of this document, the syntax in those sections is to be taken as correct.

Messages are delimited in this section between lines of "----". The "----" lines are not part of the message itself.

### A.1. Addressing examples

The following are examples of messages that might be sent between two individuals.

#### A.1.1. A message from one person to another with simple addressing

This could be called a canonical message. It has a single author, John Doe, a single recipient, Mary Smith, a subject, the date, a message identifier, and a textual message in the body.

```
----
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```





If John's secretary Michael actually sent the message, though John was the author and replies to this message should go back to him, the sender field would be used:

```
----
From: John Doe <jdoe@machine.example>
Sender: Michael Jones <mjones@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```

#### A.1.2. Different types of mailboxes

This message includes multiple addresses in the destination fields and also uses several different forms of addresses.

```
----
From: "Joe Q. Public" <john.q.public@example.com>
To: Mary Smith <mary@x.test>, jdoe@example.org, Who? <one@y.test>
Cc: <boss@nil.test>, "Giant; \"Big\" Box" <sysservices@example.net>
Date: Tue, 1 Jul 2003 10:52:37 +0200
Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.
----
```

Note that the display names for Joe Q. Public and Giant; "Big" Box needed to be enclosed in double-quotes because the former contains the period and the latter contains both semicolon and double-quote characters (the double-quote characters appearing as quoted-pair construct). Conversely, the display name for Who? could appear without them because the question mark is legal in an atom. Notice also that jdoe@example.org and boss@nil.test have no display names associated with them at all, and jdoe@example.org uses the simpler address form without the angle brackets.

A.1.3. Group addresses

```
----
From: Pete <pete@silly.example>
To: A Group:Chris Jones <c@a.test>,joe@where.test,John <jdoe@one.test>;
Cc: Undisclosed recipients:;
Date: Thu, 13 Feb 1969 23:32:54 -0330
Message-ID: <testabcd.1234@silly.example>

Testing.
----
```

In this message, the "To:" field has a single group recipient named A Group which contains 3 addresses, and a "Cc:" field with an empty group recipient named Undisclosed recipients.

A.2. Reply messages

The following is a series of three messages that make up a conversation thread between John and Mary. John firsts sends a message to Mary, Mary then replies to John's message, and then John replies to Mary's reply message.

Note especially the "Message-ID:", "References:", and "In-Reply-To:" fields in each message.

```
----
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```





When sending replies, the Subject field is often retained, though prepended with "Re: " as described in section 3.6.5.

```
----
From: Mary Smith <mary@example.net>
To: John Doe <jdoe@machine.example>
Reply-To: "Mary Smith: Personal Account" <smith@home.example>
Subject: Re: Saying Hello
Date: Fri, 21 Nov 1997 10:01:10 -0600
Message-ID: <3456@example.net>
In-Reply-To: <1234@local.machine.example>
References: <1234@local.machine.example>

This is a reply to your hello.
----
```

Note the "Reply-To:" field in the above message. When John replies to Mary's message above, the reply should go to the address in the "Reply-To:" field instead of the address in the "From:" field.

```
----
To: "Mary Smith: Personal Account" <smith@home.example>
From: John Doe <jdoe@machine.example>
Subject: Re: Saying Hello
Date: Fri, 21 Nov 1997 11:00:00 -0600
Message-ID: <abcd.1234@local.machine.tld>
In-Reply-To: <3456@example.net>
References: <1234@local.machine.example> <3456@example.net>

This is a reply to your reply.
----
```

A.3. Resent messages

Start with the message that has been used as an example several times:

```
----
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```

Say that Mary, upon receiving this message, wishes to send a copy of the message to Jane such that (a) the message would appear to have come straight from John; (b) if Jane replies to the message, the reply should go back to John; and (c) all of the original information, like the date the message was originally sent to Mary, the message identifier, and the original addressee, is preserved. In this case, resent fields are prepended to the message:

```
----
Resent-From: Mary Smith <mary@example.net>
Resent-To: Jane Brown <j-brown@other.example>
Resent-Date: Mon, 24 Nov 1997 14:22:01 -0800
Resent-Message-ID: <78910@example.net>
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```

If Jane, in turn, wished to resend this message to another person, she would prepend her own set of resent header fields to the above and send that.





## A.4. Messages with trace fields

As messages are sent through the transport system as described in [RFC2821], trace fields are prepended to the message. The following is an example of what those trace fields might look like. Note that there is some folding white space in the first one since these lines can be long.

```
----
Received: from x.y.test
   by example.net
   via TCP
   with ESMTP
   id ABC12345
   for <mary@example.net>;  21 Nov 1997 10:05:43 -0600
Received: from machine.example by x.y.test; 21 Nov 1997 10:01:22 -0600
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```

A.5. White space, comments, and other oddities

White space, including folding white space, and comments can be inserted between many of the tokens of fields. Taking the example from A.1.3, white space and comments can be inserted into all of the fields.

```
----
From: Pete(A wonderful \) chap) <pete(his account)@silly.test(his host)>
To:A Group(Some people)
     :Chris Jones <c@(Chris's host.)public.example>,
         joe@example.org,
  John <jdoe@one.test> (my dear friend); (the end of the group)
Cc:(Empty list)(start)Undisclosed recipients  :(nobody(that I know))  ;
Date: Thu,
      13
        Feb
          1969
      23:32
               -0330 (Newfoundland Time)
Message-ID:              <testabcd.1234@silly.test>

Testing.
----
```

The above example is aesthetically displeasing, but perfectly legal. Note particularly (1) the comments in the "From:" field (including one that has a ")" character appearing as part of a quoted-pair); (2) the white space absent after the ":" in the "To:" field as well as the comment and folding white space after the group name, the special character (".") in the comment in Chris Jones's address, and the folding white space before and after "joe@example.org,"; (3) the multiple and nested comments in the "Cc:" field as well as the comment immediately following the ":" after "Cc"; (4) the folding white space (but no comments except at the end) and the missing seconds in the time of the date field; and (5) the white space before (but not within) the identifier in the "Message-ID:" field.

A.6. Obsoleted forms

The following are examples of obsolete (that is, the "MUST NOT generate") syntactic elements described in section 4 of this document.





A.6.1. Obsolete addressing

Note in the below example the lack of quotes around Joe Q. Public, the route that appears in the address for Mary Smith, the two commas that appear in the "To:" field, and the spaces that appear around the "." in the jdoe address.

```
----
From: Joe Q. Public <john.q.public@example.com>
To: Mary Smith <@machine.tld:mary@example.net>, , jdoe@test   . example
Date: Tue, 1 Jul 2003 10:52:37 +0200
Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.
----
```

A.6.2. Obsolete dates

The following message uses an obsolete date format, including a non-numeric time zone and a two digit year. Note that although the day-of-week is missing, that is not specific to the obsolete syntax; it is optional in the current syntax as well.

```
----
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: 21 Nov 97 09:55:06 GMT
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
----
```

A.6.3. Obsolete white space and comments

White space and comments can appear between many more elements than in the current syntax. Also, folding lines that are made up entirely of white space are legal.

```
----
From  : John Doe <jdoe@machine(comment).  example>
To    : Mary Smith
__
          <mary@example.net>
Subject     : Saying Hello
Date  : Fri, 21 Nov 1997 09(comment):   55  :  06 -0600
Message-ID  : <1234   @   local(blah)  .machine .example>

This is a message just to say hello.
So, "Hello".
----
```

Note especially the second line of the "To:" field. It starts with two space characters. (Note that "__" represent blank spaces.) Therefore, it is considered part of the folding as described in section 4.2. Also, the comments and white space throughout addresses, dates, and message identifiers are all part of the obsolete syntax.

## Appendix B. Differences from earlier standards

This appendix contains a list of changes that have been made in the Internet Message Format from earlier standards, specifically [RFC822] and [STD3]. Items marked with an asterisk (*) below are items which appear in section 4 of this document and therefore can no longer be generated.

1. Period allowed in obsolete form of phrase.
2. ABNF moved out of document to [RFC2234].
3. Four or more digits allowed for year.
4. Header field ordering (and lack thereof) made explicit.
5. Encrypted header field removed.
6. Received syntax loosened to allow any token/value pair.
7. Specifically allow and give meaning to "-0000" time zone.
8. Folding white space is not allowed between every token.
9. Requirement for destinations removed.
10. Forwarding and resending redefined.
11. Extension header fields no longer specifically called out.
12. ASCII 0 (null) removed.*
13. Folding continuation lines cannot contain only white space.*
14. Free insertion of comments not allowed in date.*
15. Non-numeric time zones not allowed.*
16. Two digit years not allowed.*
17. Three digit years interpreted, but not allowed for generation.
18. Routes in addresses not allowed.*
19. CFWS within local-parts and domains not allowed.*
20. Empty members of address lists not allowed.*





21. Folding white space between field name and colon not allowed.*
22. Comments between field name and colon not allowed.
23. Tightened syntax of in-reply-to and references.*
24. CFWS within msg-id not allowed.*
25. Tightened semantics of resent fields as informational only.
26. Resent-Reply-To not allowed.*
27. No multiple occurrences of fields (except resent and received).*
28. Free CR and LF not allowed.*
29. Routes in return path not allowed.*
30. Line length limits specified.
31. Bcc more clearly specified.

## Appendix C. Notices

### Intellectual Property

Network Working Group
Request for Comments: 2197
Obsoletes: 1854
Category: Standards Track

N. Freed
Innosoft

September 1997

# SMTP Service Extension for Command Pipelining

## Status of this Memo



## 1. Abstract

This memo defines an extension to the SMTP service whereby a server can indicate the extent of its ability to accept multiple commands in a single TCP send operation. Using a single TCP send operation for multiple commands can improve SMTP performance significantly.

The present document is an updated version of RFC 1854 [3]. Only textual and editorial changes have been made; the protocol has not changed in any way.

## 2. Introduction

Although SMTP is widely and robustly deployed, certain extensions may nevertheless prove useful. In particular, many parts of the Internet make use of high latency network links. SMTP's intrinsic one command-one response structure is significantly penalized by high latency links, often to the point where the factors contributing to overall connection time are dominated by the time spent waiting for responses to individual commands (turnaround time).

In the best of all worlds it would be possible to simply deploy SMTP client software that makes use of command pipelining: batching up multiple commands into single TCP send operations. Unfortunately, the original SMTP specification [1] did not explicitly state that SMTP servers must support this. As a result a non-trivial number of Internet SMTP servers cannot adequately handle command pipelining. Flaws known to exist in deployed servers include:





(1) Connection handoff and buffer flushes in the middle of the SMTP dialogue. Creation of server processes for incoming SMTP connections is a useful, obvious, and harmless implementation technique. However, some SMTP servers defer process forking and connection handoff until some intermediate point in the SMTP dialogue. When this is done material read from the TCP connection and kept in process buffers can be lost.

(2) Flushing the TCP input buffer when an SMTP command fails. SMTP commands often fail but there is no reason to flush the TCP input buffer when this happens. Nevertheless, some SMTP servers do this.

(3) Improper processing and promulgation of SMTP command failures. For example, some SMTP servers will refuse to accept a DATA command if the last RCPT TO command fails, paying no attention to the success or failure of prior RCPT TO command results. Other servers will accept a DATA command even when all previous RCPT TO commands have failed. Although it is possible to accommodate this sort of behavior in a client that employs command pipelining, it does complicate the construction of the client unnecessarily.

This memo uses the mechanism described in [2] to define an extension to the SMTP service whereby an SMTP server can declare that it is capable of handling pipelined commands. The SMTP client can then check for this declaration and use pipelining only when the server declares itself capable of handling it.

### 2.1. Requirements notation

This document occasionally uses terms that appear in capital letters. When the terms "MUST", "SHOULD", "MUST NOT", "SHOULD NOT", and "MAY" appear capitalized, they are being used to indicate particular requirements of this specification. A discussion of the meanings of these terms appears in RFC 2119 [4].

## 3. Framework for the Command Pipelining Extension

The Command Pipelining extension is defined as follows:

(1) the name of the SMTP service extension is Pipelining;

(2) the EHLO keyword value associated with the extension is PIPELINING;

(3) no parameter is used with the PIPELINING EHLO keyword;

(4) no additional parameters are added to either the MAIL FROM or RCPT TO commands.

(5) no additional SMTP verbs are defined by this extension; and,

(6) the next section specifies how support for the extension affects the behavior of a server and client SMTP.

## 4. The Pipelining Service Extension

When a client SMTP wishes to employ command pipelining, it first issues the EHLO command to the server SMTP. If the server SMTP responds with code 250 to the EHLO command, and the response includes the EHLO keyword value PIPELINING, then the server SMTP has indicated that it can accommodate SMTP command pipelining.

### 4.1. Client use of pipelining

Once the client SMTP has confirmed that support exists for the pipelining extension, the client SMTP may then elect to transmit groups of SMTP commands in batches without waiting for a response to each individual command. In particular, the commands RSET, MAIL FROM, SEND FROM, SOML FROM, SAML FROM, and RCPT TO can all appear anywhere in a pipelined command group. The EHLO, DATA, VRFY, EXPN, TURN, QUIT, and NOOP commands can only appear as the last command in a group since their success or failure produces a change of state which the client SMTP must accommodate. (NOOP is included in this group so it can be used as a synchronization point.)

Additional commands added by other SMTP extensions may only appear as the last command in a group unless otherwise specified by the extensions that define the commands.

The actual transfer of message content is explicitly allowed to be the first "command" in a group. That is, a RSET/MAIL FROM sequence used to initiate a new message transaction can be placed in the same group as the final transfer of the headers and body of the previous message.

Client SMTP implementations that employ pipelining MUST check ALL statuses associated with each command in a group. For example, if none of the RCPT TO recipient addresses were accepted the client must





then check the response to the DATA command -- the client cannot assume that the DATA command will be rejected just because none of the RCPT TO commands worked. If the DATA command was properly rejected the client SMTP can just issue RSET, but if the DATA command was accepted the client SMTP should send a single dot.

Command statuses MUST be coordinated with responses by counting each separate response and correlating that count with the number of commands known to have been issued. Multiline responses MUST be supported. Matching on the basis of either the error code value or associated text is expressly forbidden.

Client SMTP implementations MAY elect to operate in a nonblocking fashion, processing server responses immediately upon receipt, even if there is still data pending transmission from the client's previous TCP send operation. If nonblocking operation is not supported, however, client SMTP implementations MUST also check the TCP window size and make sure that each group of commands fits entirely within the window. The window size is usually, but not always, 4K octets. Failure to perform this check can lead to deadlock conditions.

Clients MUST NOT confuse responses to multiple commands with multiline responses. Each command requires one or more lines of response, the last line not containing a dash between the response code and the response string.

### 4.2. Server support of pipelining

A server SMTP implementation that offers the pipelining extension:

(1) MUST NOT flush or otherwise lose the contents of the TCP input buffer under any circumstances whatsoever.

(2) SHOULD issue a positive response to the DATA command if and only if one or more valid RCPT TO addresses have been previously received.

(3) MUST NOT, after issuing a positive response to a DATA command with no valid recipients and subsequently receiving an empty message, send any message whatsoever to anybody.

(4) SHOULD elect to store responses to grouped RSET, MAIL FROM, SEND FROM, SOML FROM, SAML FROM, and RCPT TO commands in an internal buffer so they can sent as a unit.

(5) MUST NOT buffer responses to EHLO, DATA, VRFY, EXPN, TURN, QUIT, and NOOP.

(6) MUST NOT buffer responses to unrecognized commands.

(7) MUST send all pending responses immediately whenever the local TCP input buffer is emptied.

(8) MUST NOT make assumptions about commands that are yet to be received.

(9) SHOULD issue response text that indicates, either implicitly or explicitly, what command the response matches.

The overriding intent of these server requirements is to make it as easy as possible for servers to conform to these pipelining extensions.

## 5. Examples

Consider the following SMTP dialogue that does not use pipelining:

```
S: <wait for open connection>
C: <open connection to server>
S: 220 innosoft.com SMTP service ready
C: HELO dbc.mtview.ca.us
S: 250 innosoft.com
C: MAIL FROM:<mrose@dbc.mtview.ca.us>
S: 250 sender <mrose@dbc.mtview.ca.us> OK
C: RCPT TO:<ned@innosoft.com>
S: 250 recipient <ned@innosoft.com> OK
C: RCPT TO:<dan@innosoft.com>
S: 250 recipient <dan@innosoft.com> OK
C: RCPT TO:<kvc@innosoft.com>
S: 250 recipient <kvc@innosoft.com> OK
C: DATA
S: 354 enter mail, end with line containing only "."
 ...
C: .
S: 250 message sent
C: QUIT
S: 221 goodbye
```





The client waits for a server response a total of 9 times in this simple example. But if pipelining is employed the following dialogue is possible:

```
S: <wait for open connection>
C: <open connection to server>
S: 220 innosoft.com SMTP service ready
C: EHLO dbc.mtview.ca.us
S: 250-innosoft.com
S: 250 PIPELINING
C: MAIL FROM:<mrose@dbc.mtview.ca.us>
C: RCPT TO:<ned@innosoft.com>
C: RCPT TO:<dan@innosoft.com>
C: RCPT TO:<kvc@innosoft.com>
C: DATA
S: 250 sender <mrose@dbc.mtview.ca.us> OK
S: 250 recipient <ned@innosoft.com> OK
S: 250 recipient <dan@innosoft.com> OK
S: 250 recipient <kvc@innosoft.com> OK
S: 354 enter mail, end with line containing only "."
 ...
C: .
C: QUIT
S: 250 message sent
S: 221 goodbye
```

The total number of turnarounds has been reduced from 9 to 4.

The next example illustrates one possible form of behavior when pipelining is used and all recipients are rejected:

```
S: <wait for open connection>
C: <open connection to server>
S: 220 innosoft.com SMTP service ready
C: EHLO dbc.mtview.ca.us
S: 250-innosoft.com
S: 250 PIPELINING
C: MAIL FROM:<mrose@dbc.mtview.ca.us>
C: RCPT TO:<nsb@thumper.bellcore.com>
C: RCPT TO:<galvin@tis.com>
C: DATA
S: 250 sender <mrose@dbc.mtview.ca.us> OK
S: 550 remote mail to <nsb@thumper.bellore.com> not allowed
S: 550 remote mail to <galvin@tis.com> not allowed
S: 554 no valid recipients given
C: QUIT
S: 221 goodbye
```

The client SMTP waits for the server 4 times here as well. If the server SMTP does not check for at least one valid recipient prior to accepting the DATA command, the following dialogue would result:

```
S: <wait for open connection>
C: <open connection to server>
S: 220 innosoft.com SMTP service ready
C: EHLO dbc.mtview.ca.us
S: 250-innosoft.com
S: 250 PIPELINING
C: MAIL FROM:<mrose@dbc.mtview.ca.us>
C: RCPT TO:<nsb@thumper.bellcore.com>
C: RCPT TO:<galvin@tis.com>
C: DATA
S: 250 sender <mrose@dbc.mtview.ca.us> OK
S: 550 remote mail to <nsb@thumper.bellore.com> not allowed
S: 550 remote mail to <galvin@tis.com> not allowed
S: 354 enter mail, end with line containing only "."
C: .
C: QUIT
S: 554 no valid recipients
S: 221 goodbye
```

## 6. Security Considerations

This document does not discuss security issues and is not believed to raise any security issues not endemic in electronic mail and present in fully conforming implementations of [1].

## 7. Acknowledgements

This document is based on the SMTP service extension model presented in RFC 1425. Marshall Rose's description of SMTP command pipelining in his book "The Internet Message" also served as a source of inspiration for this extension.

## 8. References

[1] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, August 1982.

[2] Klensin, J., Freed, N., Rose, M., Stefferud, E., and D. Crocker, "SMTP Service Extensions", RFC 1869, November 1995.

[3] Freed, N., "SMTP Service Extension for Command Pipelining", RFC 1854, October 1995.





[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

## 9. Author's Address

Ned Freed
Innosoft International, Inc.
1050 Lakes Drive
West Covina, CA 91790
USA

Phone: +1 626 919 3600
Fax: +1 626 919 3614
EMail: ned.freed@innosoft.com

This document is a product of work done by the Internet Engineering Task Force Working Group on Messaging Extensions, Alan Cargille, chair.

Network Working Group
Request for Comments: 2045
Obsoletes: 1521, 1522, 1590
Category: Standards Track

N. Freed
Innosoft
N. Borenstein
First Virtual
November 1996

# Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies

## Status of this Memo



## Abstract

STD 11, RFC 822, defines a message representation protocol specifying considerable detail about US-ASCII message headers, and leaves the message content, or message body, as flat US-ASCII text. This set of documents, collectively called the Multipurpose Internet Mail Extensions, or MIME, redefines the format of messages to allow for

(1) textual message bodies in character sets other than US-ASCII,

(2) an extensible set of different formats for non-textual message bodies,

(3) multi-part message bodies, and

(4) textual header information in character sets other than US-ASCII.

These documents are based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said so little about message bodies, these documents are largely orthogonal to (rather than a revision of) RFC 822.

This initial document specifies the various headers used to describe the structure of MIME messages. The second document, RFC 2046, defines the general structure of the MIME media typing system and defines an initial set of media types. The third document, RFC 2047, describes extensions to RFC 822 to allow non-US-ASCII text data in





Internet mail header fields. The fourth document, RFC 2048, specifies various IANA registration procedures for MIME-related facilities. The fifth and final document, RFC 2049, describes MIME conformance criteria as well as providing some illustrative examples of MIME message formats, acknowledgements, and the bibliography.

These documents are revisions of RFCs 1521, 1522, and 1590, which themselves were revisions of RFCs 1341 and 1342. An appendix in RFC 2049 describes differences and changes from previous versions.

## Table of Contents

## 1. Introduction

Since its publication in 1982, RFC 822 has defined the standard format of textual mail messages on the Internet. Its success has been such that the RFC 822 format has been adopted, wholly or partially, well beyond the confines of the Internet and the Internet SMTP transport defined by RFC 821. As the format has seen wider use, a number of limitations have proven increasingly restrictive for the user community.

RFC 822 was intended to specify a format for text messages. As such, non-text messages, such as multimedia messages that might include audio or images, are simply not mentioned. Even in the case of text, however, RFC 822 is inadequate for the needs of mail users whose languages require the use of character sets richer than US-ASCII. Since RFC 822 does not specify mechanisms for mail containing audio, video, Asian language text, or even text in most European languages, additional specifications are needed.

One of the notable limitations of RFC 821/822 based mail systems is the fact that they limit the contents of electronic mail messages to relatively short lines (e.g. 1000 characters or less [RFC-821]) of 7bit US-ASCII. This forces users to convert any non-textual data that they may wish to send into seven-bit bytes representable as printable US-ASCII characters before invoking a local mail UA (User Agent, a program with which human users send and receive mail). Examples of such encodings currently used in the Internet include pure hexadecimal, uuencode, the 3-in-4 base 64 scheme specified in RFC 1421, the Andrew Toolkit Representation [ATK], and many others.

The limitations of RFC 822 mail become even more apparent as gateways are designed to allow for the exchange of mail messages between RFC 822 hosts and X.400 hosts. X.400 [X400] specifies mechanisms for the inclusion of non-textual material within electronic mail messages. The current standards for the mapping of X.400 messages to RFC 822 messages specify either that X.400 non-textual material must be converted to (not encoded in) IA5Text format, or that they must be discarded, notifying the RFC 822 user that discarding has occurred. This is clearly undesirable, as information that a user may wish to receive is lost. Even though a user agent may not have the capability of dealing with the non-textual material, the user might have some mechanism external to the UA that can extract useful information from the material. Moreover, it does not allow for the fact that the message may eventually be gatewayed back into an X.400 message handling system (i.e., the X.400 message is "tunneled" through Internet mail), where the non-textual information would definitely become useful again.





This document describes several mechanisms that combine to solve most of these problems without introducing any serious incompatibilities with the existing world of RFC 822 mail. In particular, it describes:

(1) A MIME-Version header field, which uses a version number to declare a message to be conformant with MIME and allows mail processing agents to distinguish between such messages and those generated by older or non-conformant software, which are presumed to lack such a field.

(2) A Content-Type header field, generalized from RFC 1049, which can be used to specify the media type and subtype of data in the body of a message and to fully specify the native representation (canonical form) of such data.

(3) A Content-Transfer-Encoding header field, which can be used to specify both the encoding transformation that was applied to the body and the domain of the result. Encoding transformations other than the identity transformation are usually applied to data in order to allow it to pass through mail transport mechanisms which may have data or character set limitations.

(4) Two additional header fields that can be used to further describe the data in a body, the Content-ID and Content-Description header fields.

All of the header fields defined in this document are subject to the general syntactic rules for header fields specified in RFC 822. In particular, all of these header fields except for Content-Disposition can include RFC 822 comments, which have no semantic content and should be ignored during MIME processing.

Finally, to specify and promote interoperability, RFC 2049 provides a basic applicability statement for a subset of the above mechanisms that defines a minimal level of "conformance" with this document.

HISTORICAL NOTE: Several of the mechanisms described in this set of documents may seem somewhat strange or even baroque at first reading. It is important to note that compatibility with existing standards AND robustness across existing practice were two of the highest priorities of the working group that developed this set of documents. In particular, compatibility was always favored over elegance.

Please refer to the current edition of the "Internet Official Protocol Standards" for the standardization state and status of this protocol. RFC 822 and STD 3, RFC 1123 also provide essential background for MIME since no conforming implementation of MIME can violate them. In addition, several other informational RFC documents will be of interest to the MIME implementor, in particular RFC 1344, RFC 1345, and RFC 1524.

## 2. Definitions, Conventions, and Generic BNF Grammar

Although the mechanisms specified in this set of documents are all described in prose, most are also described formally in the augmented BNF notation of RFC 822. Implementors will need to be familiar with this notation in order to understand this set of documents, and are referred to RFC 822 for a complete explanation of the augmented BNF notation.

Some of the augmented BNF in this set of documents makes named references to syntax rules defined in RFC 822. A complete formal grammar, then, is obtained by combining the collected grammar appendices in each document in this set with the BNF of RFC 822 plus the modifications to RFC 822 defined in RFC 1123 (which specifically changes the syntax for `return', `date' and `mailbox').

All numeric and octet values are given in decimal notation in this set of documents. All media type values, subtype values, and parameter names as defined are case-insensitive. However, parameter values are case-sensitive unless otherwise specified for the specific parameter.

FORMATTING NOTE: Notes, such at this one, provide additional nonessential information which may be skipped by the reader without missing anything essential. The primary purpose of these non-essential notes is to convey information about the rationale of this set of documents, or to place these documents in the proper historical or evolutionary context. Such information may in particular be skipped by those who are focused entirely on building a conformant implementation, but may be of use to those who wish to understand why certain design choices were made.

### 2.1. CRLF

The term CRLF, in this set of documents, refers to the sequence of octets corresponding to the two US-ASCII characters CR (decimal value 13) and LF (decimal value 10) which, taken together, in this order, denote a line break in RFC 822 mail.





### 2.2. Character Set

The term "character set" is used in MIME to refer to a method of converting a sequence of octets into a sequence of characters. Note that unconditional and unambiguous conversion in the other direction is not required, in that not all characters may be representable by a given character set and a character set may provide more than one sequence of octets to represent a particular sequence of characters.

This definition is intended to allow various kinds of character encodings, from simple single-table mappings such as US-ASCII to complex table switching methods such as those that use ISO 2022's techniques, to be used as character sets. However, the definition associated with a MIME character set name must fully specify the mapping to be performed. In particular, use of external profiling information to determine the exact mapping is not permitted.

NOTE: The term "character set" was originally to describe such straightforward schemes as US-ASCII and ISO-8859-1 which have a simple one-to-one mapping from single octets to single characters. Multi-octet coded character sets and switching techniques make the situation more complex. For example, some communities use the term "character encoding" for what MIME calls a "character set", while using the phrase "coded character set" to denote an abstract mapping from integers (not octets) to characters.

### 2.3. Message

The term "message", when not further qualified, means either a (complete or "top-level") RFC 822 message being transferred on a network, or a message encapsulated in a body of type "message/rfc822" or "message/partial".

### 2.4. Entity

The term "entity", refers specifically to the MIME-defined header fields and contents of either a message or one of the parts in the body of a multipart entity. The specification of such entities is the essence of MIME. Since the contents of an entity are often called the "body", it makes sense to speak about the body of an entity. Any sort of field may be present in the header of an entity, but only those fields whose names begin with "content-" actually have any MIME-related meaning. Note that this does NOT imply thay they have no meaning at all -- an entity that is also a message has non-MIME header fields whose meanings are defined by RFC 822.

### 2.5. Body Part

The term "body part" refers to an entity inside of a multipart entity.

### 2.6. Body

The term "body", when not further qualified, means the body of an entity, that is, the body of either a message or of a body part.

NOTE: The previous four definitions are clearly circular. This is unavoidable, since the overall structure of a MIME message is indeed recursive.

### 2.7. 7bit Data

"7bit data" refers to data that is all represented as relatively short lines with 998 octets or less between CRLF line separation sequences [RFC-821]. No octets with decimal values greater than 127 are allowed and neither are NULs (octets with decimal value 0). CR (decimal value 13) and LF (decimal value 10) octets only occur as part of CRLF line separation sequences.

### 2.8. 8bit Data

"8bit data" refers to data that is all represented as relatively short lines with 998 octets or less between CRLF line separation sequences [RFC-821]), but octets with decimal values greater than 127 may be used. As with "7bit data" CR and LF octets only occur as part of CRLF line separation sequences and no NULs are allowed.

### 2.9. Binary Data

"Binary data" refers to data where any sequence of octets whatsoever is allowed.

### 2.10. Lines

"Lines" are defined as sequences of octets separated by a CRLF sequences. This is consistent with both RFC 821 and RFC 822. "Lines" only refers to a unit of data in a message, which may or may not correspond to something that is actually displayed by a user agent.







## 3. MIME Header Fields

MIME defines a number of new RFC 822 header fields that are used to describe the content of a MIME entity. These header fields occur in at least two contexts:

(1) As part of a regular RFC 822 message header.

(2) In a MIME body part header within a multipart construct.

The formal definition of these header fields is as follows:

```
entity-headers := [ content CRLF ]
                  [ encoding CRLF ]
                  [ id CRLF ]
                  [ description CRLF ]
                  *( MIME-extension-field CRLF )

MIME-message-headers := entity-headers
                        fields
                        version CRLF
                        ; The ordering of the header
                        ; fields implied by this BNF
                        ; definition should be ignored.

MIME-part-headers := entity-headers
                     [ fields ]
                     ; Any field not beginning with
                     ; "content-" can have no defined
                     ; meaning and may be ignored.
                     ; The ordering of the header
                     ; fields implied by this BNF
                     ; definition should be ignored.
```

The syntax of the various specific MIME header fields will be described in the following sections.

## 4. MIME-Version Header Field

Since RFC 822 was published in 1982, there has really been only one format standard for Internet messages, and there has been little perceived need to declare the format standard in use. This document is an independent specification that complements RFC 822. Although the extensions in this document have been defined in such a way as to be compatible with RFC 822, there are still circumstances in which it might be desirable for a mail-processing agent to know whether a message was composed with the new standard in mind.

Therefore, this document defines a new header field, "MIME-Version", which is to be used to declare the version of the Internet message body format standard in use.

Messages composed in accordance with this document MUST include such a header field, with the following verbatim text:

```
MIME-Version: 1.0
```

The presence of this header field is an assertion that the message has been composed in compliance with this document.

Since it is possible that a future document might extend the message format standard again, a formal BNF is given for the content of the MIME-Version field:

```
version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT
```

Thus, future format specifiers, which might replace or extend "1.0", are constrained to be two integer fields, separated by a period. If a message is received with a MIME-version value other than "1.0", it cannot be assumed to conform with this document.

Note that the MIME-Version header field is required at the top level of a message. It is not required for each body part of a multipart entity. It is required for the embedded headers of a body of type "message/rfc822" or "message/partial" if and only if the embedded message is itself claimed to be MIME-conformant.

It is not possible to fully specify how a mail reader that conforms with MIME as defined in this document should treat a message that might arrive in the future with some value of MIME-Version other than "1.0".

It is also worth noting that version control for specific media types is not accomplished using the MIME-Version mechanism. In particular, some formats (such as application/postscript) have version numbering conventions that are internal to the media format. Where such conventions exist, MIME does nothing to supersede them. Where no such conventions exist, a MIME media type might use a "version" parameter in the content-type field if necessary.







NOTE TO IMPLEMENTORS: When checking MIME-Version values any RFC 822 comment strings that are present must be ignored. In particular, the following four MIME-Version fields are equivalent:

```
MIME-Version: 1.0

MIME-Version: 1.0 (produced by MetaSend Vx.x)

MIME-Version: (produced by MetaSend Vx.x) 1.0

MIME-Version: 1.(produced by MetaSend Vx.x)0
```

In the absence of a MIME-Version field, a receiving mail user agent (whether conforming to MIME requirements or not) may optionally choose to interpret the body of the message according to local conventions. Many such conventions are currently in use and it should be noted that in practice non-MIME messages can contain just about anything.

It is impossible to be certain that a non-MIME mail message is actually plain text in the US-ASCII character set since it might well be a message that, using some set of nonstandard local conventions that predate MIME, includes text in another character set or non-textual data presented in a manner that cannot be automatically recognized (e.g., a uuencoded compressed UNIX tar file).

## 5. Content-Type Header Field

The purpose of the Content-Type field is to describe the data contained in the body fully enough that the receiving user agent can pick an appropriate agent or mechanism to present the data to the user, or otherwise deal with the data in an appropriate manner. The value in this field is called a media type.

HISTORICAL NOTE: The Content-Type header field was first defined in RFC 1049. RFC 1049 used a simpler and less powerful syntax, but one that is largely compatible with the mechanism given here.

The Content-Type header field specifies the nature of the data in the body of an entity by giving media type and subtype identifiers, and by providing auxiliary information that may be required for certain media types. After the media type and subtype names, the remainder of the header field is simply a set of parameters, specified in an attribute=value notation. The ordering of parameters is not significant.

In general, the top-level media type is used to declare the general type of data, while the subtype specifies a specific format for that type of data. Thus, a media type of "image/xyz" is enough to tell a user agent that the data is an image, even if the user agent has no knowledge of the specific image format "xyz". Such information can be used, for example, to decide whether or not to show a user the raw data from an unrecognized subtype -- such an action might be reasonable for unrecognized subtypes of text, but not for unrecognized subtypes of image or audio. For this reason, registered subtypes of text, image, audio, and video should not contain embedded information that is really of a different type. Such compound formats should be represented using the "multipart" or "application" types.

Parameters are modifiers of the media subtype, and as such do not fundamentally affect the nature of the content. The set of meaningful parameters depends on the media type and subtype. Most parameters are associated with a single specific subtype. However, a given top-level media type may define parameters which are applicable to any subtype of that type. Parameters may be required by their defining content type or subtype or they may be optional. MIME implementations must ignore any parameters whose names they do not recognize.

For example, the "charset" parameter is applicable to any subtype of "text", while the "boundary" parameter is required for any subtype of the "multipart" media type.

There are NO globally-meaningful parameters that apply to all media types. Truly global mechanisms are best addressed, in the MIME model, by the definition of additional Content-* header fields.

An initial set of seven top-level media types is defined in RFC 2046. Five of these are discrete types whose content is essentially opaque as far as MIME processing is concerned. The remaining two are composite types whose contents require additional handling by MIME processors.

This set of top-level media types is intended to be substantially complete. It is expected that additions to the larger set of supported types can generally be accomplished by the creation of new subtypes of these initial types. In the future, more top-level types may be defined only by a standards-track extension to this standard. If another top-level type is to be used for any reason, it must be given a name starting with "X-" to indicate its non-standard status and to avoid a potential conflict with a future official name.







## 5.1. Syntax of the Content-Type Header Field

In the Augmented BNF notation of RFC 822, a Content-Type header field value is defined as follows:

```
content := "Content-Type" ":" type "/" subtype
           *(";" parameter)
           ; Matching of media type and subtype
           ; is ALWAYS case-insensitive.

type := discrete-type / composite-type

discrete-type := "text" / "image" / "audio" / "video" /
                 "application" / extension-token

composite-type := "message" / "multipart" / extension-token

extension-token := ietf-token / x-token

ietf-token := <An extension token defined by a
               standards-track RFC and registered
               with IANA.>

x-token := <The two characters "X-" or "x-" followed, with
            no intervening white space, by any token>

subtype := extension-token / iana-token

iana-token := <A publicly-defined extension token. Tokens
               of this form must be registered with IANA
               as specified in RFC 2048.>

parameter := attribute "=" value

attribute := token
             ; Matching of attributes
             ; is ALWAYS case-insensitive.

value := token / quoted-string

token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
           or tspecials>

tspecials :=  "(" / ")" / "<" / ">" / "@" /
              "," / ";" / ":" / "\" / <">
              "/" / "[" / "]" / "?" / "="
              ; Must be in quoted-string,
              ; to use within parameter values
```

Note that the definition of "tspecials" is the same as the RFC 822 definition of "specials" with the addition of the three characters "/", "?", and "=", and the removal of ".".

Note also that a subtype specification is MANDATORY -- it may not be omitted from a Content-Type header field. As such, there are no default subtypes.

The type, subtype, and parameter names are not case sensitive. For example, TEXT, Text, and TeXt are all equivalent top-level media types. Parameter values are normally case sensitive, but sometimes are interpreted in a case-insensitive fashion, depending on the intended use. (For example, multipart boundaries are case-sensitive, but the "access-type" parameter for message/External-body is not case-sensitive.)

Note that the value of a quoted string parameter does not include the quotes. That is, the quotation marks in a quoted-string are not a part of the value of the parameter, but are merely used to delimit that parameter value. In addition, comments are allowed in accordance with RFC 822 rules for structured header fields. Thus the following two forms

```
Content-type: text/plain; charset=us-ascii (Plain text)

Content-type: text/plain; charset="us-ascii"
```

are completely equivalent.

Beyond this syntax, the only syntactic constraint on the definition of subtype names is the desire that their uses must not conflict. That is, it would be undesirable to have two different communities using "Content-Type: application/foobar" to mean two different things. The process of defining new media subtypes, then, is not intended to be a mechanism for imposing restrictions, but simply a mechanism for publicizing their definition and usage. There are, therefore, two acceptable mechanisms for defining new media subtypes:

(1) Private values (starting with "X-") may be defined bilaterally between two cooperating agents without outside registration or standardization. Such values cannot be registered or standardized.

(2) New standard values should be registered with IANA as described in RFC 2048.

The second document in this set, RFC 2046, defines the initial set of media types for MIME.







### 5.2. Content-Type Defaults

Default RFC 822 messages without a MIME Content-Type header are taken by this protocol to be plain text in the US-ASCII character set, which can be explicitly specified as:

```
Content-type: text/plain; charset=us-ascii
```

This default is assumed if no Content-Type header field is specified. It is also recommend that this default be assumed when a syntactically invalid Content-Type header field is encountered. In the presence of a MIME-Version header field and the absence of any Content-Type header field, a receiving User Agent can also assume that plain US-ASCII text was the sender's intent. Plain US-ASCII text may still be assumed in the absence of a MIME-Version or the presence of an syntactically invalid Content-Type header field, but the sender's intent might have been otherwise.

## 6. Content-Transfer-Encoding Header Field

Many media types which could be usefully transported via email are represented, in their "natural" format, as 8bit character or binary data. Such data cannot be transmitted over some transfer protocols. For example, RFC 821 (SMTP) restricts mail messages to 7bit US-ASCII data with lines no longer than 1000 characters including any trailing CRLF line separator.

It is necessary, therefore, to define a standard mechanism for encoding such data into a 7bit short line format. Proper labelling of unencoded material in less restrictive formats for direct use over less restrictive transports is also desireable. This document specifies that such encodings will be indicated by a new "Content-Transfer-Encoding" header field. This field has not been defined by any previous standard.

### 6.1. Content-Transfer-Encoding Syntax

The Content-Transfer-Encoding field's value is a single token specifying the type of encoding, as enumerated below. Formally:

```
encoding := "Content-Transfer-Encoding" ":" mechanism

mechanism := "7bit" / "8bit" / "binary" /
             "quoted-printable" / "base64" /
             ietf-token / x-token
```

These values are not case sensitive -- Base64 and BASE64 and bAsE64 are all equivalent. An encoding type of 7BIT requires that the body

is already in a 7bit mail-ready representation. This is the default value -- that is, "Content-Transfer-Encoding: 7BIT" is assumed if the Content-Transfer-Encoding header field is not present.

### 6.2. Content-Transfer-Encodings Semantics

This single Content-Transfer-Encoding token actually provides two pieces of information. It specifies what sort of encoding transformation the body was subjected to and hence what decoding operation must be used to restore it to its original form, and it specifies what the domain of the result is.

The transformation part of any Content-Transfer-Encodings specifies, either explicitly or implicitly, a single, well-defined decoding algorithm, which for any sequence of encoded octets either transforms it to the original sequence of octets which was encoded, or shows that it is illegal as an encoded sequence. Content-Transfer-Encodings transformations never depend on any additional external profile information for proper operation. Note that while decoders must produce a single, well-defined output for a valid encoding no such restrictions exist for encoders: Encoding a given sequence of octets to different, equivalent encoded sequences is perfectly legal.

Three transformations are currently defined: identity, the "quoted-printable" encoding, and the "base64" encoding. The domains are "binary", "8bit" and "7bit".

The Content-Transfer-Encoding values "7bit", "8bit", and "binary" all mean that the identity (i.e. NO) encoding transformation has been performed. As such, they serve simply as indicators of the domain of the body data, and provide useful information about the sort of encoding that might be needed for transmission in a given transport system. The terms "7bit data", "8bit data", and "binary data" are all defined in Section 2.

The quoted-printable and base64 encodings transform their input from an arbitrary domain into material in the "7bit" range, thus making it safe to carry over restricted transports. The specific definition of the transformations are given below.

The proper Content-Transfer-Encoding label must always be used. Labelling unencoded data containing 8bit characters as "7bit" is not allowed, nor is labelling unencoded non-line-oriented data as anything other than "binary" allowed.

Unlike media subtypes, a proliferation of Content-Transfer-Encoding values is both undesirable and unnecessary. However, establishing only a single transformation into the "7bit" domain does not seem







possible. There is a tradeoff between the desire for a compact and efficient encoding of largely- binary data and the desire for a somewhat readable encoding of data that is mostly, but not entirely, 7bit. For this reason, at least two encoding mechanisms are necessary: a more or less readable encoding (quoted-printable) and a "dense" or "uniform" encoding (base64).

Mail transport for unencoded 8bit data is defined in RFC 1652. As of the initial publication of this document, there are no standardized Internet mail transports for which it is legitimate to include unencoded binary data in mail bodies. Thus there are no circumstances in which the "binary" Content-Transfer-Encoding is actually valid in Internet mail. However, in the event that binary mail transport becomes a reality in Internet mail, or when MIME is used in conjunction with any other binary-capable mail transport mechanism, binary bodies must be labelled as such using this mechanism.

NOTE: The five values defined for the Content-Transfer-Encoding field imply nothing about the media type other than the algorithm by which it was encoded or the transport system requirements if unencoded.

### 6.3. New Content-Transfer-Encodings

Implementors may, if necessary, define private Content-Transfer-Encoding values, but must use an x-token, which is a name prefixed by "X-", to indicate its non-standard status, e.g., "Content-Transfer-Encoding: x-my-new-encoding". Additional standardized Content-Transfer-Encoding values must be specified by a standards-track RFC. The requirements such specifications must meet are given in RFC 2048. As such, all content-transfer-encoding namespace except that beginning with "X-" is explicitly reserved to the IETF for future use.

Unlike media types and subtypes, the creation of new Content-Transfer-Encoding values is STRONGLY discouraged, as it seems likely to hinder interoperability with little potential benefit

### 6.4. Interpretation and Use

If a Content-Transfer-Encoding header field appears as part of a message header, it applies to the entire body of that message. If a Content-Transfer-Encoding header field appears as part of an entity's headers, it applies only to the body of that entity. If an entity is of type "multipart" the Content-Transfer-Encoding is not permitted to have any value other than "7bit", "8bit" or "binary". Even more severe restrictions apply to some subtypes of the "message" type.

It should be noted that most media types are defined in terms of octets rather than bits, so that the mechanisms described here are mechanisms for encoding arbitrary octet streams, not bit streams. If a bit stream is to be encoded via one of these mechanisms, it must first be converted to an 8bit byte stream using the network standard bit order ("big-endian"), in which the earlier bits in a stream become the higher-order bits in a 8bit byte. A bit stream not ending at an 8bit boundary must be padded with zeroes. RFC 2046 provides a mechanism for noting the addition of such padding in the case of the application/octet-stream media type, which has a "padding" parameter.

The encoding mechanisms defined here explicitly encode all data in US-ASCII. Thus, for example, suppose an entity has header fields such as:

```
Content-Type: text/plain; charset=ISO-8859-1
Content-transfer-encoding: base64
```

This must be interpreted to mean that the body is a base64 US-ASCII encoding of data that was originally in ISO-8859-1, and will be in that character set again after decoding.

Certain Content-Transfer-Encoding values may only be used on certain media types. In particular, it is EXPRESSLY FORBIDDEN to use any encodings other than "7bit", "8bit", or "binary" with any composite media type, i.e. one that recursively includes other Content-Type fields. Currently the only composite media types are "multipart" and "message". All encodings that are desired for bodies of type multipart or message must be done at the innermost level, by encoding the actual body that needs to be encoded.

It should also be noted that, by definition, if a composite entity has a transfer-encoding value such as "7bit", but one of the enclosed entities has a less restrictive value such as "8bit", then either the outer "7bit" labelling is in error, because 8bit data are included, or the inner "8bit" labelling placed an unnecessarily high demand on the transport system because the actual included data were actually 7bit-safe.

NOTE ON ENCODING RESTRICTIONS: Though the prohibition against using content-transfer-encodings on composite body data may seem overly restrictive, it is necessary to prevent nested encodings, in which data are passed through an encoding algorithm multiple times, and must be decoded multiple times in order to be properly viewed. Nested encodings add considerable complexity to user agents: Aside from the obvious efficiency problems with such multiple encodings, they can obscure the basic structure of a message. In particular, they can imply that several decoding operations are necessary simply







to find out what types of bodies a message contains. Banning nested encodings may complicate the job of certain mail gateways, but this seems less of a problem than the effect of nested encodings on user agents.

Any entity with an unrecognized Content-Transfer-Encoding must be treated as if it has a Content-Type of "application/octet-stream", regardless of what the Content-Type header field actually says.

NOTE ON THE RELATIONSHIP BETWEEN CONTENT-TYPE AND CONTENT-TRANSFER-ENCODING: It may seem that the Content-Transfer-Encoding could be inferred from the characteristics of the media that is to be encoded, or, at the very least, that certain Content-Transfer-Encodings could be mandated for use with specific media types. There are several reasons why this is not the case. First, given the varying types of transports used for mail, some encodings may be appropriate for some combinations of media types and transports but not for others. (For example, in an 8bit transport, no encoding would be required for text in certain character sets, while such encodings are clearly required for 7bit SMTP.)

Second, certain media types may require different types of transfer encoding under different circumstances. For example, many PostScript bodies might consist entirely of short lines of 7bit data and hence require no encoding at all. Other PostScript bodies (especially those using Level 2 PostScript's binary encoding mechanism) may only be reasonably represented using a binary transport encoding. Finally, since the Content-Type field is intended to be an open-ended specification mechanism, strict specification of an association between media types and encodings effectively couples the specification of an application protocol with a specific lower-level transport. This is not desirable since the developers of a media type should not have to be aware of all the transports in use and what their limitations are.

### 6.5. Translating Encodings

The quoted-printable and base64 encodings are designed so that conversion between them is possible. The only issue that arises in such a conversion is the handling of hard line breaks in quoted-printable encoding output. When converting from quoted-printable to base64 a hard line break in the quoted-printable form represents a CRLF sequence in the canonical form of the data. It must therefore be converted to a corresponding encoded CRLF in the base64 form of the data. Similarly, a CRLF sequence in the canonical form of the data obtained after base64 decoding must be converted to a quoted-printable hard line break, but ONLY when converting text data.

### 6.6. Canonical Encoding Model

There was some confusion, in the previous versions of this RFC, regarding the model for when email data was to be converted to canonical form and encoded, and in particular how this process would affect the treatment of CRLFs, given that the representation of newlines varies greatly from system to system, and the relationship between content-transfer-encodings and character sets. A canonical model for encoding is presented in RFC 2049 for this reason.

### 6.7. Quoted-Printable Content-Transfer-Encoding

The Quoted-Printable encoding is intended to represent data that largely consists of octets that correspond to printable characters in the US-ASCII character set. It encodes the data in such a way that the resulting octets are unlikely to be modified by mail transport. If the data being encoded are mostly US-ASCII text, the encoded form of the data remains largely recognizable by humans. A body which is entirely US-ASCII may also be encoded in Quoted-Printable to ensure the integrity of the data should the message pass through a character-translating, and/or line-wrapping gateway.

In this encoding, octets are to be represented as determined by the following rules:

(1) (General 8bit representation) Any octet, except a CR or LF that is part of a CRLF line break of the canonical (standard) form of the data being encoded, may be represented by an "=" followed by a two digit hexadecimal representation of the octet's value. The digits of the hexadecimal alphabet, for this purpose, are "0123456789ABCDEF". Uppercase letters must be used; lowercase letters are not allowed. Thus, for example, the decimal value 12 (US-ASCII form feed) can be represented by "=0C", and the decimal value 61 (US-ASCII EQUAL SIGN) can be represented by "=3D". This rule must be followed except when the following rules allow an alternative encoding.

(2) (Literal representation) Octets with decimal values of 33 through 60 inclusive, and 62 through 126, inclusive, MAY be represented as the US-ASCII characters which correspond to those octets (EXCLAMATION POINT through LESS THAN, and GREATER THAN through TILDE, respectively).

(3) (White Space) Octets with values of 9 and 32 MAY be represented as US-ASCII TAB (HT) and SPACE characters, respectively, but MUST NOT be so represented at the end of an encoded line. Any TAB (HT) or SPACE characters on an encoded line MUST thus be followed on that line by a printable character. In particular, an "=" at the end of an encoded line, indicating a soft line break (see rule #5) may follow one or more TAB (HT) or SPACE characters. It follows that an octet with decimal value 9 or 32 appearing at the end of an encoded line must be represented according to Rule #1. This rule is necessary because some MTAs (Message Transport Agents, programs which transport messages from one user to another, or perform a portion of such transfers) are known to pad lines of text with SPACEs, and others are known to remove "white space" characters from the end of a line. Therefore, when decoding a Quoted-Printable body, any trailing white space on a line must be deleted, as it will necessarily have been added by intermediate transport agents.

(4) (Line Breaks) A line break in a text body, represented as a CRLF sequence in the text canonical form, must be represented by a (RFC 822) line break, which is also a CRLF sequence, in the Quoted-Printable encoding. Since the canonical representation of media types other than text do not generally include the representation of line breaks as CRLF sequences, no hard line breaks (i.e. line breaks that are intended to be meaningful and to be displayed to the user) can occur in the quoted-printable encoding of such types. Sequences like "=0D", "=0A", "=0A=0D" and "=0D=0A" will routinely appear in non-text data represented in quoted-printable, of course.

Note that many implementations may elect to encode the local representation of various content types directly rather than converting to canonical form first, encoding, and then converting back to local representation. In particular, this may apply to plain text material on systems that use newline conventions other than a CRLF terminator sequence. Such an implementation optimization is permissible, but only when the combined canonicalization-encoding step is equivalent to performing the three steps separately.

(5) (Soft Line Breaks) The Quoted-Printable encoding REQUIRES that encoded lines be no more than 76 characters long. If longer lines are to be encoded with the Quoted-Printable encoding, "soft" line breaks

must be used. An equal sign as the last character on a encoded line indicates such a non-significant ("soft") line break in the encoded text.

Thus if the "raw" form of the line is a single unencoded line that says:

```
Now's the time for all folk to come to the aid of their country.
```

This can be represented, in the Quoted-Printable encoding, as:

```
Now's the time =
for all folk to come=
 to the aid of their country.
```

This provides a mechanism with which long lines are encoded in such a way as to be restored by the user agent. The 76 character limit does not count the trailing CRLF, but counts all other characters, including any equal signs.

Since the hyphen character ("-") may be represented as itself in the Quoted-Printable encoding, care must be taken, when encapsulating a quoted-printable encoded body inside one or more multipart entities, to ensure that the boundary delimiter does not appear anywhere in the encoded body. (A good strategy is to choose a boundary that includes a character sequence such as "=_" which can never appear in a quoted-printable body. See the definition of multipart messages in RFC 2046.)

NOTE: The quoted-printable encoding represents something of a compromise between readability and reliability in transport. Bodies encoded with the quoted-printable encoding will work reliably over most mail gateways, but may not work perfectly over a few gateways, notably those involving translation into EBCDIC. A higher level of confidence is offered by the base64 Content-Transfer-Encoding. A way to get reasonably reliable transport through EBCDIC gateways is to also quote the US-ASCII characters

```
!"#$@[\]^`{|}~
```

according to rule #1.

Because quoted-printable data is generally assumed to be line-oriented, it is to be expected that the representation of the breaks between the lines of quoted-printable data may be altered in transport, in the same manner that plain text mail has always been altered in Internet mail when passing between systems with differing newline conventions. If such alterations are likely to constitute a







corruption of the data, it is probably more sensible to use the base64 encoding rather than the quoted-printable encoding.

NOTE: Several kinds of substrings cannot be generated according to the encoding rules for the quoted-printable content-transfer-encoding, and hence are formally illegal if they appear in the output of a quoted-printable encoder. This note enumerates these cases and suggests ways to handle such illegal substrings if any are encountered in quoted-printable data that is to be decoded.

(1) An "=" followed by two hexadecimal digits, one or both of which are lowercase letters in "abcdef", is formally illegal. A robust implementation might choose to recognize them as the corresponding uppercase letters.

(2) An "=" followed by a character that is neither a hexadecimal digit (including "abcdef") nor the CR character of a CRLF pair is illegal. This case can be the result of US-ASCII text having been included in a quoted-printable part of a message without itself having been subjected to quoted-printable encoding. A reasonable approach by a robust implementation might be to include the "=" character and the following character in the decoded data without any transformation and, if possible, indicate to the user that proper decoding was not possible at this point in the data.

(3) An "=" cannot be the ultimate or penultimate character in an encoded object. This could be handled as in case (2) above.

(4) Control characters other than TAB, or CR and LF as parts of CRLF pairs, must not appear. The same is true for octets with decimal values greater than 126. If found in incoming quoted-printable data by a decoder, a robust implementation might exclude them from the decoded data and warn the user that illegal characters were discovered.

(5) Encoded lines must not be longer than 76 characters, not counting the trailing CRLF. If longer lines are found in incoming, encoded data, a robust implementation might nevertheless decode the lines, and might report the erroneous encoding to the user.

WARNING TO IMPLEMENTORS: If binary data is encoded in quoted-printable, care must be taken to encode CR and LF characters as "=0D" and "=0A", respectively. In particular, a CRLF sequence in binary data should be encoded as "=0D=0A". Otherwise, if CRLF were represented as a hard line break, it might be incorrectly decoded on platforms with different line break conventions.

For formalists, the syntax of quoted-printable data is described by the following grammar:

```
quoted-printable := qp-line *(CRLF qp-line)

qp-line := *(qp-segment transport-padding CRLF)
           qp-part transport-padding

qp-part := qp-section
           ; Maximum length of 76 characters

qp-segment := qp-section *(SPACE / TAB) "="
              ; Maximum length of 76 characters

qp-section := [*(ptext / SPACE / TAB) ptext]

ptext := hex-octet / safe-char

safe-char := <any octet with decimal value of 33 through
             60 inclusive, and 62 through 126>
             ; Characters not listed as "mail-safe" in
             ; RFC 2049 are also not recommended.

hex-octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
             ; Octet must be used for characters > 127, =,
             ; SPACEs or TABs at the ends of lines, and is
             ; recommended for any character not listed in
             ; RFC 2049 as "mail-safe".

transport-padding := *LWSP-char
                     ; Composers MUST NOT generate
                     ; non-zero length transport
                     ; padding, but receivers MUST
                     ; be able to handle padding
                     ; added by message transports.
```

IMPORTANT: The addition of LWSP between the elements shown in this BNF is NOT allowed since this BNF does not specify a structured header field.







## 6.8. Base64 Content-Transfer-Encoding

The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. The encoding and decoding algorithms are simple, but the encoded data are consistently only about 33 percent larger than the unencoded data. This encoding is virtually identical to the one used in Privacy Enhanced Mail (PEM) applications, as defined in RFC 1421.

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.)

NOTE: This subset has the important property that it is represented identically in all versions of ISO 646, including US-ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. Other popular encodings, such as the encoding used by the uuencode utility, Macintosh binhex 4.0 [RFC-1741], and the base85 encoding specified as part of Level 2 PostScript, do not share these properties, and thus do not fulfill the portability requirements a binary transport encoding for mail must meet.

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first 8bit byte, and the eighth bit will be the low-order bit in the first 8bit byte, and so on.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 1, below, are selected so as to be universally representable, and the set excludes characters with particular significance to SMTP (e.g., ".", CR, LF) and to the multipart boundary delimiters defined in RFC 2046 (e.g., "-").

Table 1: The Base64 Alphabet

| Value | Encoding | Value | Encoding | Value | Encoding | Value | Encoding |
|---|---|---|---|---|---|---|---|
| 0 | A | 17 | R | 34 | i | 51 | z |
| 1 | B | 18 | S | 35 | j | 52 | 0 |
| 2 | C | 19 | T | 36 | k | 53 | 1 |
| 3 | D | 20 | U | 37 | l | 54 | 2 |
| 4 | E | 21 | V | 38 | m | 55 | 3 |
| 5 | F | 22 | W | 39 | n | 56 | 4 |
| 6 | G | 23 | X | 40 | o | 57 | 5 |
| 7 | H | 24 | Y | 41 | p | 58 | 6 |
| 8 | I | 25 | Z | 42 | q | 59 | 7 |
| 9 | J | 26 | a | 43 | r | 60 | 8 |
| 10 | K | 27 | b | 44 | s | 61 | 9 |
| 11 | L | 28 | c | 45 | t | 62 | + |
| 12 | M | 29 | d | 46 | u | 63 | / |
| 13 | N | 30 | e | 47 | v | | |
| 14 | O | 31 | f | 48 | w | (pad) | = |
| 15 | P | 32 | g | 49 | x | | |
| 16 | Q | 33 | h | 50 | y | | |

The encoded output stream must be represented in lines of no more than 76 characters each. All line breaks or other characters not found in Table 1 must be ignored by decoding software. In base64 data, characters other than those in Table 1, line breaks, and other white space probably indicate a transmission error, about which a warning message or even a message rejection might be appropriate under some circumstances.

Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the "=" character. Since all base64 input is an integral number of octets, only the following cases can arise: (1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding, (2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or (3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

Because it is used only for padding at the end of the data, the occurrence of any "=" characters may be taken as evidence that the end of the data has been reached (without truncation in transit). No





such assurance is possible, however, when the number of octets transmitted was a multiple of three and no "=" characters are present.

Any characters outside of the base64 alphabet are to be ignored in base64-encoded data.

Care must be taken to use the proper octets for line breaks if base64 encoding is applied directly to text material that has not been converted to canonical form. In particular, text line breaks must be converted into CRLF sequences prior to base64 encoding. The important thing to note is that this may be done directly by the encoder rather than in a prior canonicalization step in some implementations.

NOTE: There is no need to worry about quoting potential boundary delimiters within base64-encoded bodies within multipart entities because no hyphen characters are used in the base64 encoding.

## 7. Content-ID Header Field

In constructing a high-level user agent, it may be desirable to allow one body to make reference to another. Accordingly, bodies may be labelled using the "Content-ID" header field, which is syntactically identical to the "Message-ID" header field:

```
id := "Content-ID" ":" msg-id
```

Like the Message-ID values, Content-ID values must be generated to be world-unique.

The Content-ID value may be used for uniquely identifying MIME entities in several contexts, particularly for caching data referenced by the message/external-body mechanism. Although the Content-ID header is generally optional, its use is MANDATORY in implementations which generate data of the optional MIME media type "message/external-body". That is, each message/external-body entity must have a Content-ID field to permit caching of such data.

It is also worth noting that the Content-ID value has special semantics in the case of the multipart/alternative media type. This is explained in the section of RFC 2046 dealing with multipart/alternative.

## 8. Content-Description Header Field

The ability to associate some descriptive information with a given body is often desirable. For example, it may be useful to mark an "image" body as "a picture of the Space Shuttle Endeavor." Such text may be placed in the Content-Description header field. This header field is always optional.

```
description := "Content-Description" ":" *text
```

The description is presumed to be given in the US-ASCII character set, although the mechanism specified in RFC 2047 may be used for non-US-ASCII Content-Description values.

## 9. Additional MIME Header Fields

Future documents may elect to define additional MIME header fields for various purposes. Any new header field that further describes the content of a message should begin with the string "Content-" to allow such fields which appear in a message header to be distinguished from ordinary RFC 822 message header fields.

```
MIME-extension-field := <Any RFC 822 header field which
                         begins with the string
                         "Content-">
```

## 10. Summary

Using the MIME-Version, Content-Type, and Content-Transfer-Encoding header fields, it is possible to include, in a standardized way, arbitrary types of data with RFC 822 conformant mail messages. No restrictions imposed by either RFC 821 or RFC 822 are violated, and care has been taken to avoid problems caused by additional restrictions imposed by the characteristics of some Internet mail transport mechanisms (see RFC 2049).

The next document in this set, RFC 2046, specifies the initial set of media types that can be labelled and transported using these headers.

## 11. Security Considerations

Security issues are discussed in the second document in this set, RFC 2046.







## 12. Authors' Addresses

For more information, the authors of this document are best contacted via Internet mail:

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790
USA

Phone: +1 818 919 3600
Fax: +1 818 919 3614
EMail: ned@innosoft.com

Nathaniel S. Borenstein
First Virtual Holdings
25 Washington Avenue
Morristown, NJ 07960
USA

Phone: +1 201 540 8967
Fax: +1 201 993 3032
EMail: nsb@nsb.fv.com

MIME is a result of the work of the Internet Engineering Task Force Working Group on RFC 822 Extensions. The chairman of that group, Greg Vaudreuil, may be reached at:

Gregory M. Vaudreuil
Octel Network Services
17080 Dallas Parkway
Dallas, TX 75248-1905
USA

EMail: Greg.Vaudreuil@Octel.Com

## Appendix A -- Collected Grammar

This appendix contains the complete BNF grammar for all the syntax specified by this document.

By itself, however, this grammar is incomplete. It refers by name to several syntax rules that are defined by RFC 822. Rather than reproduce those definitions here, and risk unintentional differences between the two, this document simply refers the reader to RFC 822 for the remaining definitions. Wherever a term is undefined, it refers to the RFC 822 definition.

```
attribute := token
             ; Matching of attributes
             ; is ALWAYS case-insensitive.

composite-type := "message" / "multipart" / extension-token

content := "Content-Type" ":" type "/" subtype
           *(";" parameter)
           ; Matching of media type and subtype
           ; is ALWAYS case-insensitive.

description := "Content-Description" ":" *text

discrete-type := "text" / "image" / "audio" / "video" /
                 "application" / extension-token

encoding := "Content-Transfer-Encoding" ":" mechanism

entity-headers := [ content CRLF ]
                  [ encoding CRLF ]
                  [ id CRLF ]
                  [ description CRLF ]
                  *( MIME-extension-field CRLF )

extension-token := ietf-token / x-token

hex-octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
             ; Octet must be used for characters > 127, =,
             ; SPACEs or TABs at the ends of lines, and is
             ; recommended for any character not listed in
             ; RFC 2049 as "mail-safe".

iana-token := <A publicly-defined extension token. Tokens
               of this form must be registered with IANA
               as specified in RFC 2048.>
```





```
ietf-token := <An extension token defined by a
               standards-track RFC and registered
               with IANA.>

id := "Content-ID" ":" msg-id

mechanism := "7bit" / "8bit" / "binary" /
             "quoted-printable" / "base64" /
             ietf-token / x-token

MIME-extension-field := <Any RFC 822 header field which
                         begins with the string
                         "Content-">

MIME-message-headers := entity-headers
                        fields
                        version CRLF
                        ; The ordering of the header
                        ; fields implied by this BNF
                        ; definition should be ignored.

MIME-part-headers := entity-headers
                     [fields]
                     ; Any field not beginning with
                     ; "content-" can have no defined
                     ; meaning and may be ignored.
                     ; The ordering of the header
                     ; fields implied by this BNF
                     ; definition should be ignored.

parameter := attribute "=" value

ptext := hex-octet / safe-char

qp-line := *(qp-segment transport-padding CRLF)
           qp-part transport-padding

qp-part := qp-section
           ; Maximum length of 76 characters

qp-section := [*(ptext / SPACE / TAB) ptext]

qp-segment := qp-section *(SPACE / TAB) "="
              ; Maximum length of 76 characters

quoted-printable := qp-line *(CRLF qp-line)
```

```
safe-char := <any octet with decimal value of 33 through
             60 inclusive, and 62 through 126>
             ; Characters not listed as "mail-safe" in
             ; RFC 2049 are also not recommended.

subtype := extension-token / iana-token

token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
           or tspecials>

transport-padding := *LWSP-char
                     ; Composers MUST NOT generate
                     ; non-zero length transport
                     ; padding, but receivers MUST
                     ; be able to handle padding
                     ; added by message transports.

tspecials :=  "(" / ")" / "<" / ">" / "@" /
              "," / ";" / ":" / "\" / <">
              "/" / "[" / "]" / "?" / "="
              ; Must be in quoted-string,
              ; to use within parameter values

type := discrete-type / composite-type

value := token / quoted-string

version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT

x-token := <The two characters "X-" or "x-" followed, with
            no  intervening white space, by any token>
```

Network Working Group
Request for Comments: 2046
Obsoletes: 1521, 1522, 1590
Category: Standards Track

N. Freed
Innosoft
N. Borenstein
First Virtual

November 1996

# Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types

## Status of this Memo



## Abstract

STD 11, RFC 822 defines a message representation protocol specifying considerable detail about US-ASCII message headers, but which leaves the message content, or message body, as flat US-ASCII text. This set of documents, collectively called the Multipurpose Internet Mail Extensions, or MIME, redefines the format of messages to allow for

(1) textual message bodies in character sets other than US-ASCII,

(2) an extensible set of different formats for non-textual message bodies,

(3) multi-part message bodies, and

(4) textual header information in character sets other than US-ASCII.

These documents are based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said so little about message bodies, these documents are largely orthogonal to (rather than a revision of) RFC 822.

The initial document in this set, RFC 2045, specifies the various headers used to describe the structure of MIME messages. This second document defines the general structure of the MIME media typing system and defines an initial set of media types. The third document, RFC 2047, describes extensions to RFC 822 to allow non-US-ASCII text





data in Internet mail header fields. The fourth document, RFC 2048, specifies various IANA registration procedures for MIME-related facilities. The fifth and final document, RFC 2049, describes MIME conformance criteria as well as providing some illustrative examples of MIME message formats, acknowledgements, and the bibliography.

These documents are revisions of RFCs 1521 and 1522, which themselves were revisions of RFCs 1341 and 1342. An appendix in RFC 2049 describes differences and changes from previous versions.

## Table of Contents

## 1. Introduction

The first document in this set, RFC 2045, defines a number of header fields, including Content-Type. The Content-Type field is used to specify the nature of the data in the body of a MIME entity, by giving media type and subtype identifiers, and by providing auxiliary information that may be required for certain media types. After the type and subtype names, the remainder of the header field is simply a set of parameters, specified in an attribute/value notation. The ordering of parameters is not significant.

In general, the top-level media type is used to declare the general type of data, while the subtype specifies a specific format for that type of data. Thus, a media type of "image/xyz" is enough to tell a user agent that the data is an image, even if the user agent has no knowledge of the specific image format "xyz". Such information can be used, for example, to decide whether or not to show a user the raw data from an unrecognized subtype -- such an action might be reasonable for unrecognized subtypes of "text", but not for unrecognized subtypes of "image" or "audio". For this reason, registered subtypes of "text", "image", "audio", and "video" should not contain embedded information that is really of a different type. Such compound formats should be represented using the "multipart" or "application" types.

Parameters are modifiers of the media subtype, and as such do not fundamentally affect the nature of the content. The set of meaningful parameters depends on the media type and subtype. Most parameters are associated with a single specific subtype. However, a given top-level media type may define parameters which are applicable to any subtype of that type. Parameters may be required by their defining media type or subtype or they may be optional. MIME implementations must also ignore any parameters whose names they do not recognize.

MIME's Content-Type header field and media type mechanism has been carefully designed to be extensible, and it is expected that the set of media type/subtype pairs and their associated parameters will grow significantly over time. Several other MIME facilities, such as transfer encodings and "message/external-body" access types, are likely to have new values defined over time. In order to ensure that the set of such values is developed in an orderly, well-specified, and public manner, MIME sets up a registration process which uses the Internet Assigned Numbers Authority (IANA) as a central registry for MIME's various areas of extensibility. The registration process for these areas is described in a companion document, RFC 2048.







The initial seven standard top-level media type are defined and described in the remainder of this document.

## 2. Definition of a Top-Level Media Type

The definition of a top-level media type consists of:

(1) a name and a description of the type, including criteria for whether a particular type would qualify under that type,

(2) the names and definitions of parameters, if any, which are defined for all subtypes of that type (including whether such parameters are required or optional),

(3) how a user agent and/or gateway should handle unknown subtypes of this type,

(4) general considerations on gatewaying entities of this top-level type, if any, and

(5) any restrictions on content-transfer-encodings for entities of this top-level type.

## 3. Overview Of The Initial Top-Level Media Types

The five discrete top-level media types are:

(1) text -- textual information. The subtype "plain" in particular indicates plain text containing no formatting commands or directives of any sort. Plain text is intended to be displayed "as-is". No special software is required to get the full meaning of the text, aside from support for the indicated character set. Other subtypes are to be used for enriched text in forms where application software may enhance the appearance of the text, but such software must not be required in order to get the general idea of the content. Possible subtypes of "text" thus include any word processor format that can be read without resorting to software that understands the format. In particular, formats that employ embeddded binary formatting information are not considered directly readable. A very simple and portable subtype, "richtext", was defined in RFC 1341, with a further revision in RFC 1896 under the name "enriched".

(2) image -- image data. "Image" requires a display device (such as a graphical display, a graphics printer, or a FAX machine) to view the information. An initial subtype is defined for the widely-used image format JPEG. . subtypes are defined for two widely-used image formats, jpeg and gif.

(3) audio -- audio data. "Audio" requires an audio output device (such as a speaker or a telephone) to "display" the contents. An initial subtype "basic" is defined in this document.

(4) video -- video data. "Video" requires the capability to display moving images, typically including specialized hardware and software. An initial subtype "mpeg" is defined in this document.

(5) application -- some other kind of data, typically either uninterpreted binary data or information to be processed by an application. The subtype "octet-stream" is to be used in the case of uninterpreted binary data, in which case the simplest recommended action is to offer to write the information into a file for the user. The "PostScript" subtype is also defined for the transport of PostScript material. Other expected uses for "application" include spreadsheets, data for mail-based scheduling systems, and languages for "active" (computational) messaging, and word processing formats that are not directly readable. Note that security considerations may exist for some types of application data, most notably "application/PostScript" and any form of active messaging. These issues are discussed later in this document.

The two composite top-level media types are:

(1) multipart -- data consisting of multiple entities of independent data types. Four subtypes are initially defined, including the basic "mixed" subtype specifying a generic mixed set of parts, "alternative" for representing the same data in multiple formats, "parallel" for parts intended to be viewed simultaneously, and "digest" for multipart entities in which each part has a default type of "message/rfc822".







(2) message -- an encapsulated message. A body of media type "message" is itself all or a portion of some kind of message object. Such objects may or may not in turn contain other entities. The "rfc822" subtype is used when the encapsulated content is itself an RFC 822 message. The "partial" subtype is defined for partial RFC 822 messages, to permit the fragmented transmission of bodies that are thought to be too large to be passed through transport facilities in one piece. Another subtype, "external-body", is defined for specifying large bodies by reference to an external data source.

It should be noted that the list of media type values given here may be augmented in time, via the mechanisms described above, and that the set of subtypes is expected to grow substantially.

## 4. Discrete Media Type Values

Five of the seven initial media type values refer to discrete bodies. The content of these types must be handled by non-MIME mechanisms; they are opaque to MIME processors.

### 4.1. Text Media Type

The "text" media type is intended for sending material which is principally textual in form. A "charset" parameter may be used to indicate the character set of the body text for "text" subtypes, notably including the subtype "text/plain", which is a generic subtype for plain text. Plain text does not provide for or allow formatting commands, font attribute specifications, processing instructions, interpretation directives, or content markup. Plain text is seen simply as a linear sequence of characters, possibly interrupted by line breaks or page breaks. Plain text may allow the stacking of several characters in the same position in the text. Plain text in scripts like Arabic and Hebrew may also include facilitites that allow the arbitrary mixing of text segments with opposite writing directions.

Beyond plain text, there are many formats for representing what might be known as "rich text". An interesting characteristic of many such representations is that they are to some extent readable even without the software that interprets them. It is useful, then, to distinguish them, at the highest level, from such unreadable data as images, audio, or text represented in an unreadable form. In the absence of appropriate interpretation software, it is reasonable to show subtypes of "text" to the user, while it is not reasonable to do so with most nontextual data. Such formatted textual data should be represented using subtypes of "text".

### 4.1.1. Representation of Line Breaks

The canonical form of any MIME "text" subtype MUST always represent a line break as a CRLF sequence. Similarly, any occurrence of CRLF in MIME "text" MUST represent a line break. Use of CR and LF outside of line break sequences is also forbidden.

This rule applies regardless of format or character set or sets involved.

NOTE: The proper interpretation of line breaks when a body is displayed depends on the media type. In particular, while it is appropriate to treat a line break as a transition to a new line when displaying a "text/plain" body, this treatment is actually incorrect for other subtypes of "text" like "text/enriched" [RFC-1896]. Similarly, whether or not line breaks should be added during display operations is also a function of the media type. It should not be necessary to add any line breaks to display "text/plain" correctly, whereas proper display of "text/enriched" requires the appropriate addition of line breaks.

NOTE: Some protocols defines a maximum line length. E.g. SMTP [RFC-821] allows a maximum of 998 octets before the next CRLF sequence. To be transported by such protocols, data which includes too long segments without CRLF sequences must be encoded with a suitable content-transfer-encoding.

### 4.1.2. Charset Parameter

A critical parameter that may be specified in the Content-Type field for "text/plain" data is the character set. This is specified with a "charset" parameter, as in:

```
Content-type: text/plain; charset=iso-8859-1
```

Unlike some other parameter values, the values of the charset parameter are NOT case sensitive. The default character set, which must be assumed in the absence of a charset parameter, is US-ASCII.

The specification for any future subtypes of "text" must specify whether or not they will also utilize a "charset" parameter, and may possibly restrict its values as well. For other subtypes of "text" than "text/plain", the semantics of the "charset" parameter should be defined to be identical to those specified here for "text/plain", i.e., the body consists entirely of characters in the given charset. In particular, definers of future "text" subtypes should pay close attention to the implications of multioctet character sets for their subtype definitions.







The charset parameter for subtypes of "text" gives a name of a character set, as "character set" is defined in RFC 2045. The rules regarding line breaks detailed in the previous section must also be observed -- a character set whose definition does not conform to these rules cannot be used in a MIME "text" subtype.

An initial list of predefined character set names can be found at the end of this section. Additional character sets may be registered with IANA.

Other media types than subtypes of "text" might choose to employ the charset parameter as defined here, but with the CRLF/line break restriction removed. Therefore, all character sets that conform to the general definition of "character set" in RFC 2045 can be registered for MIME use.

Note that if the specified character set includes 8-bit characters and such characters are used in the body, a Content-Transfer-Encoding header field and a corresponding encoding on the data are required in order to transmit the body via some mail transfer protocols, such as SMTP [RFC-821].

The default character set, US-ASCII, has been the subject of some confusion and ambiguity in the past. Not only were there some ambiguities in the definition, there have been wide variations in practice. In order to eliminate such ambiguity and variations in the future, it is strongly recommended that new user agents explicitly specify a character set as a media type parameter in the Content-Type header field. "US-ASCII" does not indicate an arbitrary 7-bit character set, but specifies that all octets in the body must be interpreted as characters according to the US-ASCII character set. National and application-oriented versions of ISO 646 [ISO-646] are usually NOT identical to US-ASCII, and in that case their use in Internet mail is explicitly discouraged. The omission of the ISO 646 character set from this document is deliberate in this regard. The character set name of "US-ASCII" explicitly refers to the character set defined in ANSI X3.4-1986 [US- ASCII]. The new international reference version (IRV) of the 1991 edition of ISO 646 is identical to US-ASCII. The character set name "ASCII" is reserved and must not be used for any purpose.

NOTE: RFC 821 explicitly specifies "ASCII", and references an earlier version of the American Standard. Insofar as one of the purposes of specifying a media type and character set is to permit the receiver to unambiguously determine how the sender intended the coded message to be interpreted, assuming anything other than "strict ASCII" as the default would risk unintentional and incompatible changes to the semantics of messages now being transmitted. This also implies that

messages containing characters coded according to other versions of ISO 646 than US-ASCII and the 1991 IRV, or using code-switching procedures (e.g., those of ISO 2022), as well as 8bit or multiple octet character encodings MUST use an appropriate character set specification to be consistent with MIME.

The complete US-ASCII character set is listed in ANSI X3.4- 1986. Note that the control characters including DEL (0-31, 127) have no defined meaning in apart from the combination CRLF (US-ASCII values 13 and 10) indicating a new line. Two of the characters have de facto meanings in wide use: FF (12) often means "start subsequent text on the beginning of a new page"; and TAB or HT (9) often (though not always) means "move the cursor to the next available column after the current position where the column number is a multiple of 8 (counting the first column as column 0)." Aside from these conventions, any use of the control characters or DEL in a body must either occur

(1) because a subtype of text other than "plain" specifically assigns some additional meaning, or

(2) within the context of a private agreement between the sender and recipient. Such private agreements are discouraged and should be replaced by the other capabilities of this document.

NOTE: An enormous proliferation of character sets exist beyond US-ASCII. A large number of partially or totally overlapping character sets is NOT a good thing. A SINGLE character set that can be used universally for representing all of the world's languages in Internet mail would be preferrable. Unfortunately, existing practice in several communities seems to point to the continued use of multiple character sets in the near future. A small number of standard character sets are, therefore, defined for Internet use in this document.

The defined charset values are:

(1) US-ASCII -- as defined in ANSI X3.4-1986 [US-ASCII].

(2) ISO-8859-X -- where "X" is to be replaced, as necessary, for the parts of ISO-8859 [ISO-8859]. Note that the ISO 646 character sets have deliberately been omitted in favor of their 8859 replacements, which are the designated character sets for Internet mail. As of the publication of this document, the legitimate values for "X" are the digits 1 through 10.







Characters in the range 128-159 has no assigned meaning in ISO-8859-X. Characters with values below 128 in ISO-8859-X have the same assigned meaning as they do in US-ASCII.

Part 6 of ISO 8859 (Latin/Arabic alphabet) and part 8 (Latin/Hebrew alphabet) includes both characters for which the normal writing direction is right to left and characters for which it is left to right, but do not define a canonical ordering method for representing bi-directional text. The charset values "ISO-8859-6" and "ISO-8859-8", however, specify that the visual method is used [RFC-1556].

All of these character sets are used as pure 7bit or 8bit sets without any shift or escape functions. The meaning of shift and escape sequences in these character sets is not defined.

The character sets specified above are the ones that were relatively uncontroversial during the drafting of MIME. This document does not endorse the use of any particular character set other than US-ASCII, and recognizes that the future evolution of world character sets remains unclear.

Note that the character set used, if anything other than US- ASCII, must always be explicitly specified in the Content-Type field.

No character set name other than those defined above may be used in Internet mail without the publication of a formal specification and its registration with IANA, or by private agreement, in which case the character set name must begin with "X-".

Implementors are discouraged from defining new character sets unless absolutely necessary.

The "charset" parameter has been defined primarily for the purpose of textual data, and is described in this section for that reason. However, it is conceivable that non-textual data might also wish to specify a charset value for some purpose, in which case the same syntax and values should be used.

In general, composition software should always use the "lowest common denominator" character set possible. For example, if a body contains only US-ASCII characters, it SHOULD be marked as being in the US-ASCII character set, not ISO-8859-1, which, like all the ISO-8859 family of character sets, is a superset of US-ASCII. More generally, if a widely-used character set is a subset of another character set, and a body contains only characters in the widely-used subset, it should be labelled as being in that subset. This will increase the chances that the recipient will be able to view the resulting entity correctly.

#### 4.1.3. Plain Subtype

The simplest and most important subtype of "text" is "plain". This indicates plain text that does not contain any formatting commands or directives. Plain text is intended to be displayed "as-is", that is, no interpretation of embedded formatting commands, font attribute specifications, processing instructions, interpretation directives, or content markup should be necessary for proper display. The default media type of "text/plain; charset=us-ascii" for Internet mail describes existing Internet practice. That is, it is the type of body defined by RFC 822.

No other "text" subtype is defined by this document.

#### 4.1.4. Unrecognized Subtypes

Unrecognized subtypes of "text" should be treated as subtype "plain" as long as the MIME implementation knows how to handle the charset. Unrecognized subtypes which also specify an unrecognized charset should be treated as "application/octet- stream".

### 4.2. Image Media Type

A media type of "image" indicates that the body contains an image. The subtype names the specific image format. These names are not case sensitive. An initial subtype is "jpeg" for the JPEG format using JFIF encoding [JPEG].

The list of "image" subtypes given here is neither exclusive nor exhaustive, and is expected to grow as more types are registered with IANA, as described in RFC 2048.

Unrecognized subtypes of "image" should at a miniumum be treated as "application/octet-stream". Implementations may optionally elect to pass subtypes of "image" that they do not specifically recognize to a secure and robust general-purpose image viewing application, if such an application is available.

NOTE: Using of a generic-purpose image viewing application this way inherits the security problems of the most dangerous type supported by the application.

### 4.3. Audio Media Type

A media type of "audio" indicates that the body contains audio data. Although there is not yet a consensus on an "ideal" audio format for use with computers, there is a pressing need for a format capable of providing interoperable behavior.







The initial subtype of "basic" is specified to meet this requirement by providing an absolutely minimal lowest common denominator audio format. It is expected that richer formats for higher quality and/or lower bandwidth audio will be defined by a later document.

The content of the "audio/basic" subtype is single channel audio encoded using 8bit ISDN mu-law [PCM] at a sample rate of 8000 Hz.

Unrecognized subtypes of "audio" should at a miniumum be treated as "application/octet-stream". Implementations may optionally elect to pass subtypes of "audio" that they do not specifically recognize to a robust general-purpose audio playing application, if such an application is available.

### 4.4. Video Media Type

A media type of "video" indicates that the body contains a time-varying-picture image, possibly with color and coordinated sound. The term 'video' is used in its most generic sense, rather than with reference to any particular technology or format, and is not meant to preclude subtypes such as animated drawings encoded compactly. The subtype "mpeg" refers to video coded according to the MPEG standard [MPEG].

Note that although in general this document strongly discourages the mixing of multiple media in a single body, it is recognized that many so-called video formats include a representation for synchronized audio, and this is explicitly permitted for subtypes of "video".

Unrecognized subtypes of "video" should at a minumum be treated as "application/octet-stream". Implementations may optionally elect to pass subtypes of "video" that they do not specifically recognize to a robust general-purpose video display application, if such an application is available.

### 4.5. Application Media Type

The "application" media type is to be used for discrete data which do not fit in any of the other categories, and particularly for data to be processed by some type of application program. This is information which must be processed by an application before it is viewable or usable by a user. Expected uses for the "application" media type include file transfer, spreadsheets, data for mail-based scheduling systems, and languages for "active" (computational) material. (The latter, in particular, can pose security problems which must be understood by implementors, and are considered in detail in the discussion of the "application/PostScript" media type.)

For example, a meeting scheduler might define a standard representation for information about proposed meeting dates. An intelligent user agent would use this information to conduct a dialog with the user, and might then send additional material based on that dialog. More generally, there have been several "active" messaging languages developed in which programs in a suitably specialized language are transported to a remote location and automatically run in the recipient's environment.

Such applications may be defined as subtypes of the "application" media type. This document defines two subtypes:

octet-stream, and PostScript.

The subtype of "application" will often be either the name or include part of the name of the application for which the data are intended. This does not mean, however, that any application program name may be used freely as a subtype of "application".

### 4.5.1. Octet-Stream Subtype

The "octet-stream" subtype is used to indicate that a body contains arbitrary binary data. The set of currently defined parameters is:

(1) TYPE -- the general type or category of binary data. This is intended as information for the human recipient rather than for any automatic processing.

(2) PADDING -- the number of bits of padding that were appended to the bit-stream comprising the actual contents to produce the enclosed 8bit byte-oriented data. This is useful for enclosing a bit-stream in a body when the total number of bits is not a multiple of 8.

Both of these parameters are optional.

An additional parameter, "CONVERSIONS", was defined in RFC 1341 but has since been removed. RFC 1341 also defined the use of a "NAME" parameter which gave a suggested file name to be used if the data were to be written to a file. This has been deprecated in anticipation of a separate Content-Disposition header field, to be defined in a subsequent RFC.

The recommended action for an implementation that receives an "application/octet-stream" entity is to simply offer to put the data in a file, with any Content-Transfer-Encoding undone, or perhaps to use it as input to a user-specified process.







To reduce the danger of transmitting rogue programs, it is strongly recommended that implementations NOT implement a path-search mechanism whereby an arbitrary program named in the Content-Type parameter (e.g., an "interpreter=" parameter) is found and executed using the message body as input.

### 4.5.2. PostScript Subtype

A media type of "application/postscript" indicates a PostScript program. Currently two variants of the PostScript language are allowed; the original level 1 variant is described in [POSTSCRIPT] and the more recent level 2 variant is described in [POSTSCRIPT2].

PostScript is a registered trademark of Adobe Systems, Inc. Use of the MIME media type "application/postscript" implies recognition of that trademark and all the rights it entails.

The PostScript language definition provides facilities for internal labelling of the specific language features a given program uses. This labelling, called the PostScript document structuring conventions, or DSC, is very general and provides substantially more information than just the language level. The use of document structuring conventions, while not required, is strongly recommended as an aid to interoperability. Documents which lack proper structuring conventions cannot be tested to see whether or not they will work in a given environment. As such, some systems may assume the worst and refuse to process unstructured documents.

The execution of general-purpose PostScript interpreters entails serious security risks, and implementors are discouraged from simply sending PostScript bodies to "off- the-shelf" interpreters. While it is usually safe to send PostScript to a printer, where the potential for harm is greatly constrained by typical printer environments, implementors should consider all of the following before they add interactive display of PostScript bodies to their MIME readers.

The remainder of this section outlines some, though probably not all, of the possible problems with the transport of PostScript entities.

(1) Dangerous operations in the PostScript language include, but may not be limited to, the PostScript operators "deletefile", "renamefile", "filenameforall", and "file". "File" is only dangerous when applied to something other than standard input or output. Implementations may also define additional nonstandard file operators; these may also pose a threat to security. "Filenameforall", the wildcard file search operator, may appear at first glance to be harmless.

Note, however, that this operator has the potential to reveal information about what files the recipient has access to, and this information may itself be sensitive. Message senders should avoid the use of potentially dangerous file operators, since these operators are quite likely to be unavailable in secure PostScript implementations. Message receiving and displaying software should either completely disable all potentially dangerous file operators or take special care not to delegate any special authority to their operation. These operators should be viewed as being done by an outside agency when interpreting PostScript documents. Such disabling and/or checking should be done completely outside of the reach of the PostScript language itself; care should be taken to insure that no method exists for re-enabling full-function versions of these operators.

(2) The PostScript language provides facilities for exiting the normal interpreter, or server, loop. Changes made in this "outer" environment are customarily retained across documents, and may in some cases be retained semipermanently in nonvolatile memory. The operators associated with exiting the interpreter loop have the potential to interfere with subsequent document processing. As such, their unrestrained use constitutes a threat of service denial. PostScript operators that exit the interpreter loop include, but may not be limited to, the exitserver and startjob operators. Message sending software should not generate PostScript that depends on exiting the interpreter loop to operate, since the ability to exit will probably be unavailable in secure PostScript implementations. Message receiving and displaying software should completely disable the ability to make retained changes to the PostScript environment by eliminating or disabling the "startjob" and "exitserver" operations. If these operations cannot be eliminated or completely disabled the password associated with them should at least be set to a hard-to-guess value.

(3) PostScript provides operators for setting system-wide and device-specific parameters. These parameter settings may be retained across jobs and may potentially pose a threat to the correct operation of the interpreter. The PostScript operators that set system and device parameters include, but may not be







limited to, the "setsystemparams" and "setdevparams" operators. Message sending software should not generate PostScript that depends on the setting of system or device parameters to operate correctly. The ability to set these parameters will probably be unavailable in secure PostScript implementations. Message receiving and displaying software should disable the ability to change system and device parameters. If these operators cannot be completely disabled the password associated with them should at least be set to a hard-to-guess value.

(4) Some PostScript implementations provide nonstandard facilities for the direct loading and execution of machine code. Such facilities are quite obviously open to substantial abuse. Message sending software should not make use of such features. Besides being totally hardware-specific, they are also likely to be unavailable in secure implementations of PostScript. Message receiving and displaying software should not allow such operators to be used if they exist.

(5) PostScript is an extensible language, and many, if not most, implementations of it provide a number of their own extensions. This document does not deal with such extensions explicitly since they constitute an unknown factor. Message sending software should not make use of nonstandard extensions; they are likely to be missing from some implementations. Message receiving and displaying software should make sure that any nonstandard PostScript operators are secure and don't present any kind of threat.

(6) It is possible to write PostScript that consumes huge amounts of various system resources. It is also possible to write PostScript programs that loop indefinitely. Both types of programs have the potential to cause damage if sent to unsuspecting recipients. Message-sending software should avoid the construction and dissemination of such programs, which is antisocial. Message receiving and displaying software should provide appropriate mechanisms to abort processing after a reasonable amount of time has elapsed. In addition, PostScript interpreters should be limited to the consumption of only a reasonable amount of any given system resource.

(7) It is possible to include raw binary information inside PostScript in various forms. This is not recommended for use in Internet mail, both because it is not supported by all PostScript interpreters and because it significantly complicates the use of a MIME Content-Transfer-Encoding. (Without such binary, PostScript may typically be viewed as line-oriented data. The treatment of CRLF sequences becomes extremely problematic if binary and line-oriented data are mixed in a single Postscript data stream.)

(8) Finally, bugs may exist in some PostScript interpreters which could possibly be exploited to gain unauthorized access to a recipient's system. Apart from noting this possibility, there is no specific action to take to prevent this, apart from the timely correction of such bugs if any are found.

#### 4.5.3. Other Application Subtypes

It is expected that many other subtypes of "application" will be defined in the future. MIME implementations must at a minimum treat any unrecognized subtypes as being equivalent to "application/octet-stream".

## 5. Composite Media Type Values

The remaining two of the seven initial Content-Type values refer to composite entities. Composite entities are handled using MIME mechanisms -- a MIME processor typically handles the body directly.

### 5.1. Multipart Media Type

In the case of multipart entities, in which one or more different sets of data are combined in a single body, a "multipart" media type field must appear in the entity's header. The body must then contain one or more body parts, each preceded by a boundary delimiter line, and the last one followed by a closing boundary delimiter line. After its boundary delimiter line, each body part then consists of a header area, a blank line, and a body area. Thus a body part is similar to an RFC 822 message in syntax, but different in meaning.

A body part is an entity and hence is NOT to be interpreted as actually being an RFC 822 message. To begin with, NO header fields are actually required in body parts. A body part that starts with a blank line, therefore, is allowed and is a body part for which all default values are to be assumed. In such a case, the absence of a Content-Type header usually indicates that the corresponding body has





a content-type of "text/plain; charset=US-ASCII".

The only header fields that have defined meaning for body parts are those the names of which begin with "Content-". All other header fields may be ignored in body parts. Although they should generally be retained if at all possible, they may be discarded by gateways if necessary. Such other fields are permitted to appear in body parts but must not be depended on. "X-" fields may be created for experimental or private purposes, with the recognition that the information they contain may be lost at some gateways.

NOTE: The distinction between an RFC 822 message and a body part is subtle, but important. A gateway between Internet and X.400 mail, for example, must be able to tell the difference between a body part that contains an image and a body part that contains an encapsulated message, the body of which is a JPEG image. In order to represent the latter, the body part must have "Content-Type: message/rfc822", and its body (after the blank line) must be the encapsulated message, with its own "Content-Type: image/jpeg" header field. The use of similar syntax facilitates the conversion of messages to body parts, and vice versa, but the distinction between the two must be understood by implementors. (For the special case in which parts actually are messages, a "digest" subtype is also defined.)

As stated previously, each body part is preceded by a boundary delimiter line that contains the boundary delimiter. The boundary delimiter MUST NOT appear inside any of the encapsulated parts, on a line by itself or as the prefix of any line. This implies that it is crucial that the composing agent be able to choose and specify a unique boundary parameter value that does not contain the boundary parameter value of an enclosing multipart as a prefix.

All present and future subtypes of the "multipart" type must use an identical syntax. Subtypes may differ in their semantics, and may impose additional restrictions on syntax, but must conform to the required syntax for the "multipart" type. This requirement ensures that all conformant user agents will at least be able to recognize and separate the parts of any multipart entity, even those of an unrecognized subtype.

As stated in the definition of the Content-Transfer-Encoding field [RFC 2045], no encoding other than "7bit", "8bit", or "binary" is permitted for entities of type "multipart". The "multipart" boundary delimiters and header fields are always represented as 7bit US-ASCII in any case (though the header fields may encode non-US-ASCII header text as per RFC 2047) and data within the body parts can be encoded on a part-by-part basis, with Content-Transfer-Encoding fields for each appropriate body part.

### 5.1.1. Common Syntax

This section defines a common syntax for subtypes of "multipart". All subtypes of "multipart" must use this syntax. A simple example of a multipart message also appears in this section. An example of a more complex multipart message is given in RFC 2049.

The Content-Type field for multipart entities requires one parameter, "boundary". The boundary delimiter line is then defined as a line consisting entirely of two hyphen characters ("-", decimal value 45) followed by the boundary parameter value from the Content-Type header field, optional linear whitespace, and a terminating CRLF.

NOTE: The hyphens are for rough compatibility with the earlier RFC 934 method of message encapsulation, and for ease of searching for the boundaries in some implementations. However, it should be noted that multipart messages are NOT completely compatible with RFC 934 encapsulations; in particular, they do not obey RFC 934 quoting conventions for embedded lines that begin with hyphens. This mechanism was chosen over the RFC 934 mechanism because the latter causes lines to grow with each level of quoting. The combination of this growth with the fact that SMTP implementations sometimes wrap long lines made the RFC 934 mechanism unsuitable for use in the event that deeply-nested multipart structuring is ever desired.

WARNING TO IMPLEMENTORS: The grammar for parameters on the Content-type field is such that it is often necessary to enclose the boundary parameter values in quotes on the Content-type line. This is not always necessary, but never hurts. Implementors should be sure to study the grammar carefully in order to avoid producing invalid Content-type fields. Thus, a typical "multipart" Content-Type header field might look like this:

```
Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
```

But the following is not valid:

```
Content-Type: multipart/mixed; boundary=gc0pJq0M:08jU534c0p
```

(because of the colon) and must instead be represented as

```
Content-Type: multipart/mixed; boundary="gc0pJq0M:08jU534c0p"
```

This Content-Type value indicates that the content consists of one or more parts, each with a structure that is syntactically identical to an RFC 822 message, except that the header area is allowed to be completely empty, and that the parts are each preceded by the line





```
--gc0pJq0M:08jU534c0p
```

The boundary delimiter MUST occur at the beginning of a line, i.e., following a CRLF, and the initial CRLF is considered to be attached to the boundary delimiter line rather than part of the preceding part. The boundary may be followed by zero or more characters of linear whitespace. It is then terminated by either another CRLF and the header fields for the next part, or by two CRLFs, in which case there are no header fields for the next part. If no Content-Type field is present it is assumed to be "message/rfc822" in a "multipart/digest" and "text/plain" otherwise.

NOTE: The CRLF preceding the boundary delimiter line is conceptually attached to the boundary so that it is possible to have a part that does not end with a CRLF (line break). Body parts that must be considered to end with line breaks, therefore, must have two CRLFs preceding the boundary delimiter line, the first of which is part of the preceding body part, and the second of which is part of the encapsulation boundary.

Boundary delimiters must not appear within the encapsulated material, and must be no longer than 70 characters, not counting the two leading hyphens.

The boundary delimiter line following the last body part is a distinguished delimiter that indicates that no further body parts will follow. Such a delimiter line is identical to the previous delimiter lines, with the addition of two more hyphens after the boundary parameter value.

```
--gc0pJq0M:08jU534c0p--
```

NOTE TO IMPLEMENTORS: Boundary string comparisons must compare the boundary value with the beginning of each candidate line. An exact match of the entire candidate line is not required; it is sufficient that the boundary appear in its entirety following the CRLF.

There appears to be room for additional information prior to the first boundary delimiter line and following the final boundary delimiter line. These areas should generally be left blank, and implementations must ignore anything that appears before the first boundary delimiter line or after the last one.

NOTE: These "preamble" and "epilogue" areas are generally not used because of the lack of proper typing of these parts and the lack of clear semantics for handling these areas at gateways, particularly X.400 gateways. However, rather than leaving the preamble area blank, many MIME implementations have found this to be a convenient

place to insert an explanatory note for recipients who read the message with pre-MIME software, since such notes will be ignored by MIME-compliant software.

NOTE: Because boundary delimiters must not appear in the body parts being encapsulated, a user agent must exercise care to choose a unique boundary parameter value. The boundary parameter value in the example above could have been the result of an algorithm designed to produce boundary delimiters with a very low probability of already existing in the data to be encapsulated without having to prescan the data. Alternate algorithms might result in more "readable" boundary delimiters for a recipient with an old user agent, but would require more attention to the possibility that the boundary delimiter might appear at the beginning of some line in the encapsulated part. The simplest boundary delimiter line possible is something like "---", with a closing boundary delimiter line of "-----".

As a very simple example, the following multipart message has two parts, both of them plain text, one of them explicitly typed and one of them implicitly typed:

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Date: Sun, 21 Mar 1993 23:56:48 -0800 (PST)
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple boundary"

This is the preamble.  It is to be ignored, though it
is a handy place for composition agents to include an
explanatory note to non-MIME conformant readers.

--simple boundary

This is implicitly typed plain US-ASCII text.
It does NOT end with a linebreak.
--simple boundary
Content-type: text/plain; charset=us-ascii

This is explicitly typed plain US-ASCII text.
It DOES end with a linebreak.

--simple boundary--

This is the epilogue.  It is also to be ignored.
```





The use of a media type of "multipart" in a body part within another "multipart" entity is explicitly allowed. In such cases, for obvious reasons, care must be taken to ensure that each nested "multipart" entity uses a different boundary delimiter. See RFC 2049 for an example of nested "multipart" entities.

The use of the "multipart" media type with only a single body part may be useful in certain contexts, and is explicitly permitted.

NOTE: Experience has shown that a "multipart" media type with a single body part is useful for sending non-text media types. It has the advantage of providing the preamble as a place to include decoding instructions. In addition, a number of SMTP gateways move or remove the MIME headers, and a clever MIME decoder can take a good guess at multipart boundaries even in the absence of the Content-Type header and thereby successfully decode the message.

The only mandatory global parameter for the "multipart" media type is the boundary parameter, which consists of 1 to 70 characters from a set of characters known to be very robust through mail gateways, and NOT ending with white space. (If a boundary delimiter line appears to end with white space, the white space must be presumed to have been added by a gateway, and must be deleted.) It is formally specified by the following BNF:

```
boundary := 0*69<bchars> bcharsnospace

bchars := bcharsnospace / " "

bcharsnospace := DIGIT / ALPHA / "'" / "(" / ")" /
                 "+" / "_" / "," / "-" / "." /
                 "/" / ":" / "=" / "?"
```

Overall, the body of a "multipart" entity may be specified as follows:

```
dash-boundary := "--" boundary
                 ; boundary taken from the value of
                 ; boundary parameter of the
                 ; Content-Type field.

multipart-body := [preamble CRLF]
                  dash-boundary transport-padding CRLF
                  body-part *encapsulation
                  close-delimiter transport-padding
                  [CRLF epilogue]
```

```
transport-padding := *LWSP-char
                     ; Composers MUST NOT generate
                     ; non-zero length transport
                     ; padding, but receivers MUST
                     ; be able to handle padding
                     ; added by message transports.

encapsulation := delimiter transport-padding
                 CRLF body-part

delimiter := CRLF dash-boundary

close-delimiter := delimiter "--"

preamble := discard-text

epilogue := discard-text

discard-text := *(*text CRLF) *text
                ; May be ignored or discarded.

body-part := MIME-part-headers [CRLF *OCTET]
             ; Lines in a body-part must not start
             ; with the specified dash-boundary and
             ; the delimiter must not appear anywhere
             ; in the body part.  Note that the
             ; semantics of a body-part differ from
             ; the semantics of a message, as
             ; described in the text.

OCTET := <any 0-255 octet value>
```

IMPORTANT: The free insertion of linear-white-space and RFC 822 comments between the elements shown in this BNF is NOT allowed since this BNF does not specify a structured header field.

NOTE: In certain transport enclaves, RFC 822 restrictions such as the one that limits bodies to printable US-ASCII characters may not be in force. (That is, the transport domains may exist that resemble standard Internet mail transport as specified in RFC 821 and assumed by RFC 822, but without certain restrictions.) The relaxation of these restrictions should be construed as locally extending the definition of bodies, for example to include octets outside of the US-ASCII range, as long as these extensions are supported by the transport and adequately documented in the Content- Transfer-Encoding header field. However, in no event are headers (either message headers or body part headers) allowed to contain anything other than US-ASCII characters.







NOTE: Conspicuously missing from the "multipart" type is a notion of structured, related body parts. It is recommended that those wishing to provide more structured or integrated multipart messaging facilities should define subtypes of multipart that are syntactically identical but define relationships between the various parts. For example, subtypes of multipart could be defined that include a distinguished part which in turn is used to specify the relationships between the other parts, probably referring to them by their Content-ID field. Old implementations will not recognize the new subtype if this approach is used, but will treat it as multipart/mixed and will thus be able to show the user the parts that are recognized.

### 5.1.2. Handling Nested Messages and Multiparts

The "message/rfc822" subtype defined in a subsequent section of this document has no terminating condition other than running out of data. Similarly, an improperly truncated "multipart" entity may not have any terminating boundary marker, and can turn up operationally due to mail system malfunctions.

It is essential that such entities be handled correctly when they are themselves imbedded inside of another "multipart" structure. MIME implementations are therefore required to recognize outer level boundary markers at ANY level of inner nesting. It is not sufficient to only check for the next expected marker or other terminating condition.

### 5.1.3. Mixed Subtype

The "mixed" subtype of "multipart" is intended for use when the body parts are independent and need to be bundled in a particular order. Any "multipart" subtypes that an implementation does not recognize must be treated as being of subtype "mixed".

### 5.1.4. Alternative Subtype

The "multipart/alternative" type is syntactically identical to "multipart/mixed", but the semantics are different. In particular, each of the body parts is an "alternative" version of the same information.

Systems should recognize that the content of the various parts are interchangeable. Systems should choose the "best" type based on the local environment and references, in some cases even through user interaction. As with "multipart/mixed", the order of body parts is significant. In this case, the alternatives appear in an order of increasing faithfulness to the original content. In general, the

best choice is the LAST part of a type supported by the recipient system's local environment.

"Multipart/alternative" may be used, for example, to send a message in a fancy text format in such a way that it can easily be displayed anywhere:

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Date: Mon, 22 Mar 1993 09:41:09 -0800 (PST)
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary=boundary42

--boundary42
Content-Type: text/plain; charset=us-ascii

  ... plain text version of message goes here ...

--boundary42
Content-Type: text/enriched

  ... RFC 1896 text/enriched version of same message
      goes here ...

--boundary42
Content-Type: application/x-whatever

  ... fanciest version of same message goes here ...

--boundary42--
```

In this example, users whose mail systems understood the "application/x-whatever" format would see only the fancy version, while other users would see only the enriched or plain text version, depending on the capabilities of their system.

In general, user agents that compose "multipart/alternative" entities must place the body parts in increasing order of preference, that is, with the preferred format last. For fancy text, the sending user agent should put the plainest format first and the richest format last. Receiving user agents should pick and display the last format they are capable of displaying. In the case where one of the alternatives is itself of type "multipart" and contains unrecognized sub-parts, the user agent may choose either to show that alternative, an earlier alternative, or both.







NOTE: From an implementor's perspective, it might seem more sensible to reverse this ordering, and have the plainest alternative last. However, placing the plainest alternative first is the friendliest possible option when "multipart/alternative" entities are viewed using a non-MIME-conformant viewer. While this approach does impose some burden on conformant MIME viewers, interoperability with older mail readers was deemed to be more important in this case.

It may be the case that some user agents, if they can recognize more than one of the formats, will prefer to offer the user the choice of which format to view. This makes sense, for example, if a message includes both a nicely- formatted image version and an easily-edited text version. What is most critical, however, is that the user not automatically be shown multiple versions of the same data. Either the user should be shown the last recognized version or should be given the choice.

THE SEMANTICS OF CONTENT-ID IN MULTIPART/ALTERNATIVE: Each part of a "multipart/alternative" entity represents the same data, but the mappings between the two are not necessarily without information loss. For example, information is lost when translating ODA to PostScript or plain text. It is recommended that each part should have a different Content-ID value in the case where the information content of the two parts is not identical. And when the information content is identical -- for example, where several parts of type "message/external-body" specify alternate ways to access the identical data -- the same Content-ID field value should be used, to optimize any caching mechanisms that might be present on the recipient's end. However, the Content-ID values used by the parts should NOT be the same Content-ID value that describes the "multipart/alternative" as a whole, if there is any such Content-ID field. That is, one Content-ID value will refer to the "multipart/alternative" entity, while one or more other Content-ID values will refer to the parts inside it.

### 5.1.5. Digest Subtype

This document defines a "digest" subtype of the "multipart" Content-Type. This type is syntactically identical to "multipart/mixed", but the semantics are different. In particular, in a digest, the default Content-Type value for a body part is changed from "text/plain" to "message/rfc822". This is done to allow a more readable digest format that is largely compatible (except for the quoting convention) with RFC 934.

Note: Though it is possible to specify a Content-Type value for a body part in a digest which is other than "message/rfc822", such as a "text/plain" part containing a description of the material in the

digest, actually doing so is undesireble. The "multipart/digest" Content-Type is intended to be used to send collections of messages. If a "text/plain" part is needed, it should be included as a seperate part of a "multipart/mixed" message.

A digest in this format might, then, look something like this:

```
From: Moderator-Address
To: Recipient-List
Date: Mon, 22 Mar 1994 13:34:51 +0000
Subject: Internet Digest, volume 42
MIME-Version: 1.0
Content-Type: multipart/mixed;
              boundary="---- main boundary ----"

------ main boundary ----

  ...Introductory text or table of contents...

------ main boundary ----
Content-Type: multipart/digest;
              boundary="---- next message ----"

------ next message ----

From: someone-else
Date: Fri, 26 Mar 1993 11:13:32 +0200
Subject: my opinion

  ...body goes here ...

------ next message ----

From: someone-else-again
Date: Fri, 26 Mar 1993 10:07:13 -0500
Subject: my different opinion

  ... another body goes here ...

------ next message ------

------ main boundary ------
```

#### 5.1.6. Parallel Subtype

This document defines a "parallel" subtype of the "multipart" Content-Type. This type is syntactically identical to "multipart/mixed", but the semantics are different. In particular, in a parallel entity, the order of body parts is not significant.

A common presentation of this type is to display all of the parts simultaneously on hardware and software that are capable of doing so. However, composing agents should be aware that many mail readers will lack this capability and will show the parts serially in any event.

#### 5.1.7. Other Multipart Subtypes

Other "multipart" subtypes are expected in the future. MIME implementations must in general treat unrecognized subtypes of "multipart" as being equivalent to "multipart/mixed".

### 5.2. Message Media Type

It is frequently desirable, in sending mail, to encapsulate another mail message. A special media type, "message", is defined to facilitate this. In particular, the "rfc822" subtype of "message" is used to encapsulate RFC 822 messages.

NOTE: It has been suggested that subtypes of "message" might be defined for forwarded or rejected messages. However, forwarded and rejected messages can be handled as multipart messages in which the first part contains any control or descriptive information, and a second part, of type "message/rfc822", is the forwarded or rejected message. Composing rejection and forwarding messages in this manner will preserve the type information on the original message and allow it to be correctly presented to the recipient, and hence is strongly encouraged.

Subtypes of "message" often impose restrictions on what encodings are allowed. These restrictions are described in conjunction with each specific subtype.

Mail gateways, relays, and other mail handling agents are commonly known to alter the top-level header of an RFC 822 message. In particular, they frequently add, remove, or reorder header fields. These operations are explicitly forbidden for the encapsulated headers embedded in the bodies of messages of type "message."

#### 5.2.1. RFC822 Subtype

A media type of "message/rfc822" indicates that the body contains an encapsulated message, with the syntax of an RFC 822 message. However, unlike top-level RFC 822 messages, the restriction that each "message/rfc822" body must include a "From", "Date", and at least one destination header is removed and replaced with the requirement that at least one of "From", "Subject", or "Date" must be present.

It should be noted that, despite the use of the numbers "822", a "message/rfc822" entity isn't restricted to material in strict conformance to RFC822, nor are the semantics of "message/rfc822" objects restricted to the semantics defined in RFC822. More specifically, a "message/rfc822" message could well be a News article or a MIME message.

No encoding other than "7bit", "8bit", or "binary" is permitted for the body of a "message/rfc822" entity. The message header fields are always US-ASCII in any case, and data within the body can still be encoded, in which case the Content-Transfer-Encoding header field in the encapsulated message will reflect this. Non-US-ASCII text in the headers of an encapsulated message can be specified using the mechanisms described in RFC 2047.

### 5.2.2. Partial Subtype

The "partial" subtype is defined to allow large entities to be delivered as several separate pieces of mail and automatically reassembled by a receiving user agent. (The concept is similar to IP fragmentation and reassembly in the basic Internet Protocols.) This mechanism can be used when intermediate transport agents limit the size of individual messages that can be sent. The media type "message/partial" thus indicates that the body contains a fragment of a larger entity.

Because data of type "message" may never be encoded in base64 or quoted-printable, a problem might arise if "message/partial" entities are constructed in an environment that supports binary or 8bit transport. The problem is that the binary data would be split into multiple "message/partial" messages, each of them requiring binary transport. If such messages were encountered at a gateway into a 7bit transport environment, there would be no way to properly encode them for the 7bit world, aside from waiting for all of the fragments, reassembling the inner message, and then encoding the reassembled data in base64 or quoted-printable. Since it is possible that different fragments might go through different gateways, even this is not an acceptable solution. For this reason, it is specified that entities of type "message/partial" must always have a content-transfer-encoding of 7bit (the default). In particular, even in environments that support binary or 8bit transport, the use of a content- transfer-encoding of "8bit" or "binary" is explicitly prohibited for MIME entities of type "message/partial". This in turn implies that the inner message must not use "8bit" or "binary" encoding.







Because some message transfer agents may choose to automatically fragment large messages, and because such agents may use very different fragmentation thresholds, it is possible that the pieces of a partial message, upon reassembly, may prove themselves to comprise a partial message. This is explicitly permitted.

Three parameters must be specified in the Content-Type field of type "message/partial": The first, "id", is a unique identifier, as close to a world-unique identifier as possible, to be used to match the fragments together. (In general, the identifier is essentially a message-id; if placed in double quotes, it can be ANY message-id, in accordance with the BNF for "parameter" given in RFC 2045.) The second, "number", an integer, is the fragment number, which indicates where this fragment fits into the sequence of fragments. The third, "total", another integer, is the total number of fragments. This third subfield is required on the final fragment, and is optional (though encouraged) on the earlier fragments. Note also that these parameters may be given in any order.

Thus, the second piece of a 3-piece message may have either of the following header fields:

```
Content-Type: Message/Partial; number=2; total=3;
              id="oc=jpbe0M2Yt4s@thumper.bellcore.com"

Content-Type: Message/Partial;
              id="oc=jpbe0M2Yt4s@thumper.bellcore.com";
              number=2
```

But the third piece MUST specify the total number of fragments:

```
Content-Type: Message/Partial; number=3; total=3;
              id="oc=jpbe0M2Yt4s@thumper.bellcore.com"
```

Note that fragment numbering begins with 1, not 0.

When the fragments of an entity broken up in this manner are put together, the result is always a complete MIME entity, which may have its own Content-Type header field, and thus may contain any other data type.

#### 5.2.2.1. Message Fragmentation and Reassembly

The semantics of a reassembled partial message must be those of the "inner" message, rather than of a message containing the inner message. This makes it possible, for example, to send a large audio message as several partial messages, and still have it appear to the recipient as a simple audio message rather than as an encapsulated

message containing an audio message. That is, the encapsulation of the message is considered to be "transparent".

When generating and reassembling the pieces of a "message/partial" message, the headers of the encapsulated message must be merged with the headers of the enclosing entities. In this process the following rules must be observed:

(1) Fragmentation agents must split messages at line boundaries only. This restriction is imposed because splits at points other than the ends of lines in turn depends on message transports being able to preserve the semantics of messages that don't end with a CRLF sequence. Many transports are incapable of preserving such semantics.

(2) All of the header fields from the initial enclosing message, except those that start with "Content-" and the specific header fields "Subject", "Message-ID", "Encrypted", and "MIME-Version", must be copied, in order, to the new message.

(3) The header fields in the enclosed message which start with "Content-", plus the "Subject", "Message-ID", "Encrypted", and "MIME-Version" fields, must be appended, in order, to the header fields of the new message. Any header fields in the enclosed message which do not start with "Content-" (except for the "Subject", "Message-ID", "Encrypted", and "MIME-Version" fields) will be ignored and dropped.

(4) All of the header fields from the second and any subsequent enclosing messages are discarded by the reassembly process.

5.2.2.2. Fragmentation and Reassembly Example

If an audio message is broken into two pieces, the first piece might look something like this:

```
X-Weird-Header-1: Foo
From: Bill@host.com
To: joe@otherhost.com
Date: Fri, 26 Mar 1993 12:59:38 -0500 (EST)
Subject: Audio mail (part 1 of 2)
Message-ID: <id1@host.com>
MIME-Version: 1.0
Content-type: message/partial; id="ABC@host.com";
              number=1; total=2

X-Weird-Header-1: Bar
X-Weird-Header-2: Hello
Message-ID: <anotherid@foo.com>
Subject: Audio mail
MIME-Version: 1.0
Content-type: audio/basic
Content-transfer-encoding: base64

  ... first half of encoded audio data goes here ...
```

and the second half might look something like this:

```
From: Bill@host.com
To: joe@otherhost.com
Date: Fri, 26 Mar 1993 12:59:38 -0500 (EST)
Subject: Audio mail (part 2 of 2)
MIME-Version: 1.0
Message-ID: <id2@host.com>
Content-type: message/partial;
              id="ABC@host.com"; number=2; total=2

  ... second half of encoded audio data goes here ...
```

Then, when the fragmented message is reassembled, the resulting message to be displayed to the user should look something like this:

```
X-Weird-Header-1: Foo
From: Bill@host.com
To: joe@otherhost.com
Date: Fri, 26 Mar 1993 12:59:38 -0500 (EST)
Subject: Audio mail
Message-ID: <anotherid@foo.com>
MIME-Version: 1.0
Content-type: audio/basic
Content-transfer-encoding: base64

  ... first half of encoded audio data goes here ...
  ... second half of encoded audio data goes here ...
```

The inclusion of a "References" field in the headers of the second and subsequent pieces of a fragmented message that references the Message-Id on the previous piece may be of benefit to mail readers that understand and track references. However, the generation of such "References" fields is entirely optional.

Finally, it should be noted that the "Encrypted" header field has been made obsolete by Privacy Enhanced Messaging (PEM) [RFC-1421, RFC-1422, RFC-1423, RFC-1424], but the rules above are nevertheless believed to describe the correct way to treat it if it is encountered in the context of conversion to and from "message/partial" fragments.

### 5.2.3. External-Body Subtype

The external-body subtype indicates that the actual body data are not included, but merely referenced. In this case, the parameters describe a mechanism for accessing the external data.

When a MIME entity is of type "message/external-body", it consists of a header, two consecutive CRLFs, and the message header for the encapsulated message. If another pair of consecutive CRLFs appears, this of course ends the message header for the encapsulated message. However, since the encapsulated message's body is itself external, it does NOT appear in the area that follows. For example, consider the following message:

```
Content-type: message/external-body;
              access-type=local-file;
              name="/u/nsb/Me.jpeg"

Content-type: image/jpeg
Content-ID: <id42@guppylake.bellcore.com>
Content-Transfer-Encoding: binary

THIS IS NOT REALLY THE BODY!
```

The area at the end, which might be called the "phantom body", is ignored for most external-body messages. However, it may be used to contain auxiliary information for some such messages, as indeed it is when the access-type is "mail- server". The only access-type defined in this document that uses the phantom body is "mail-server", but other access-types may be defined in the future in other specifications that use this area.

The encapsulated headers in ALL "message/external-body" entities MUST include a Content-ID header field to give a unique identifier by which to reference the data. This identifier may be used for caching mechanisms, and for recognizing the receipt of the data when the access-type is "mail-server".

Note that, as specified here, the tokens that describe external-body data, such as file names and mail server commands, are required to be in the US-ASCII character set.







If this proves problematic in practice, a new mechanism may be required as a future extension to MIME, either as newly defined access-types for "message/external-body" or by some other mechanism.

As with "message/partial", MIME entities of type "message/external-body" MUST have a content-transfer-encoding of 7bit (the default). In particular, even in environments that support binary or 8bit transport, the use of a content- transfer-encoding of "8bit" or "binary" is explicitly prohibited for entities of type "message/external-body".

#### 5.2.3.1. General External-Body Parameters

The parameters that may be used with any "message/external- body" are:

(1) ACCESS-TYPE -- A word indicating the supported access mechanism by which the file or data may be obtained. This word is not case sensitive. Values include, but are not limited to, "FTP", "ANON-FTP", "TFTP", "LOCAL-FILE", and "MAIL-SERVER". Future values, except for experimental values beginning with "X-", must be registered with IANA, as described in RFC 2048. This parameter is unconditionally mandatory and MUST be present on EVERY "message/external-body".

(2) EXPIRATION -- The date (in the RFC 822 "date-time" syntax, as extended by RFC 1123 to permit 4 digits in the year field) after which the existence of the external data is not guaranteed. This parameter may be used with ANY access-type and is ALWAYS optional.

(3) SIZE -- The size (in octets) of the data. The intent of this parameter is to help the recipient decide whether or not to expend the necessary resources to retrieve the external data. Note that this describes the size of the data in its canonical form, that is, before any Content-Transfer-Encoding has been applied or after the data have been decoded. This parameter may be used with ANY access-type and is ALWAYS optional.

(4) PERMISSION -- A case-insensitive field that indicates whether or not it is expected that clients might also attempt to overwrite the data. By default, or if permission is "read", the assumption is that they are not, and that if the data is retrieved once, it is never needed again. If PERMISSION is "read-write",

this assumption is invalid, and any local copy must be considered no more than a cache. "Read" and "Read-write" are the only defined values of permission. This parameter may be used with ANY access-type and is ALWAYS optional.

The precise semantics of the access-types defined here are described in the sections that follow.

#### 5.2.3.2. The 'ftp' and 'tftp' Access-Types

An access-type of FTP or TFTP indicates that the message body is accessible as a file using the FTP [RFC-959] or TFTP [RFC- 783] protocols, respectively. For these access-types, the following additional parameters are mandatory:

(1) NAME -- The name of the file that contains the actual body data.

(2) SITE -- A machine from which the file may be obtained, using the given protocol. This must be a fully qualified domain name, not a nickname.

(3) Before any data are retrieved, using FTP, the user will generally need to be asked to provide a login id and a password for the machine named by the site parameter. For security reasons, such an id and password are not specified as content-type parameters, but must be obtained from the user.

In addition, the following parameters are optional:

(1) DIRECTORY -- A directory from which the data named by NAME should be retrieved.

(2) MODE -- A case-insensitive string indicating the mode to be used when retrieving the information. The valid values for access-type "TFTP" are "NETASCII", "OCTET", and "MAIL", as specified by the TFTP protocol [RFC-783]. The valid values for access-type "FTP" are "ASCII", "EBCDIC", "IMAGE", and "LOCALn" where "n" is a decimal integer, typically 8. These correspond to the representation types "A" "E" "I" and "L n" as specified by the FTP protocol [RFC-959]. Note that "BINARY" and "TENEX" are not valid values for MODE and that "OCTET" or "IMAGE" or "LOCAL8" should be used instead. IF MODE is not specified, the default value is "NETASCII" for TFTP and "ASCII" otherwise.







#### 5.2.3.3. The 'anon-ftp' Access-Type

The "anon-ftp" access-type is identical to the "ftp" access type, except that the user need not be asked to provide a name and password for the specified site. Instead, the ftp protocol will be used with login "anonymous" and a password that corresponds to the user's mail address.

#### 5.2.3.4. The 'local-file' Access-Type

An access-type of "local-file" indicates that the actual body is accessible as a file on the local machine. Two additional parameters are defined for this access type:

(1) NAME -- The name of the file that contains the actual body data. This parameter is mandatory for the "local-file" access-type.

(2) SITE -- A domain specifier for a machine or set of machines that are known to have access to the data file. This optional parameter is used to describe the locality of reference for the data, that is, the site or sites at which the file is expected to be visible. Asterisks may be used for wildcard matching to a part of a domain name, such as "*.bellcore.com", to indicate a set of machines on which the data should be directly visible, while a single asterisk may be used to indicate a file that is expected to be universally available, e.g., via a global file system.

#### 5.2.3.5. The 'mail-server' Access-Type

The "mail-server" access-type indicates that the actual body is available from a mail server. Two additional parameters are defined for this access-type:

(1) SERVER -- The addr-spec of the mail server from which the actual body data can be obtained. This parameter is mandatory for the "mail-server" access-type.

(2) SUBJECT -- The subject that is to be used in the mail that is sent to obtain the data. Note that keying mail servers on Subject lines is NOT recommended, but such mail servers are known to exist. This is an optional parameter.

Because mail servers accept a variety of syntaxes, some of which is multiline, the full command to be sent to a mail server is not included as a parameter in the content-type header field. Instead, it is provided as the "phantom body" when the media type is "message/external-body" and the access-type is mail-server.

Note that MIME does not define a mail server syntax. Rather, it allows the inclusion of arbitrary mail server commands in the phantom body. Implementations must include the phantom body in the body of the message it sends to the mail server address to retrieve the relevant data.

Unlike other access-types, mail-server access is asynchronous and will happen at an unpredictable time in the future. For this reason, it is important that there be a mechanism by which the returned data can be matched up with the original "message/external-body" entity. MIME mail servers must use the same Content-ID field on the returned message that was used in the original "message/external-body" entities, to facilitate such matching.

#### 5.2.3.6. External-Body Security Issues

"Message/external-body" entities give rise to two important security issues:

(1) Accessing data via a "message/external-body" reference effectively results in the message recipient performing an operation that was specified by the message originator. It is therefore possible for the message originator to trick a recipient into doing something they would not have done otherwise. For example, an originator could specify a action that attempts retrieval of material that the recipient is not authorized to obtain, causing the recipient to unwittingly violate some security policy. For this reason, user agents capable of resolving external references must always take steps to describe the action they are to take to the recipient and ask for explicit permisssion prior to performing it.

The 'mail-server' access-type is particularly vulnerable, in that it causes the recipient to send a new message whose contents are specified by the original message's originator. Given the potential for abuse, any such request messages that are constructed should contain a clear indication that they were generated automatically (e.g. in a Comments: header field) in an attempt to resolve a MIME







"message/external-body" reference.

(2) MIME will sometimes be used in environments that provide some guarantee of message integrity and authenticity. If present, such guarantees may apply only to the actual direct content of messages -- they may or may not apply to data accessed through MIME's "message/external-body" mechanism. In particular, it may be possible to subvert certain access mechanisms even when the messaging system itself is secure.

It should be noted that this problem exists either with or without the availabilty of MIME mechanisms. A casual reference to an FTP site containing a document in the text of a secure message brings up similar issues -- the only difference is that MIME provides for automatic retrieval of such material, and users may place unwarranted trust is such automatic retrieval mechanisms.

#### 5.2.3.7. Examples and Further Explanations

When the external-body mechanism is used in conjunction with the "multipart/alternative" media type it extends the functionality of "multipart/alternative" to include the case where the same entity is provided in the same format but via different accces mechanisms. When this is done the originator of the message must order the parts first in terms of preferred formats and then by preferred access mechanisms. The recipient's viewer should then evaluate the list both in terms of format and access mechanisms.

With the emerging possibility of very wide-area file systems, it becomes very hard to know in advance the set of machines where a file will and will not be accessible directly from the file system. Therefore it may make sense to provide both a file name, to be tried directly, and the name of one or more sites from which the file is known to be accessible. An implementation can try to retrieve remote files using FTP or any other protocol, using anonymous file retrieval or prompting the user for the necessary name and password. If an external body is accessible via multiple mechanisms, the sender may include multiple entities of type "message/external-body" within the body parts of an enclosing "multipart/alternative" entity.

However, the external-body mechanism is not intended to be limited to file retrieval, as shown by the mail-server access-type. Beyond this, one can imagine, for example, using a video server for external references to video clips.

The embedded message header fields which appear in the body of the "message/external-body" data must be used to declare the media type of the external body if it is anything other than plain US-ASCII text, since the external body does not have a header section to declare its type. Similarly, any Content-transfer-encoding other than "7bit" must also be declared here. Thus a complete "message/external-body" message, referring to an object in PostScript format, might look like this:

```
From: Whomever
To: Someone
Date: Whenever
Subject: whatever
MIME-Version: 1.0
Message-ID: <id1@host.com>
Content-Type: multipart/alternative; boundary=42
Content-ID: <id001@guppylake.bellcore.com>

--42
Content-Type: message/external-body; name="BodyFormats.ps";
              site="thumper.bellcore.com"; mode="image";
              access-type=ANON-FTP; directory="pub";
              expiration="Fri, 14 Jun 1991 19:13:14 -0400 (EDT)"

Content-type: application/postscript
Content-ID: <id42@guppylake.bellcore.com>

--42
Content-Type: message/external-body; access-type=local-file;
              name="/u/nsb/writing/rfcs/RFC-MIME.ps";
              site="thumper.bellcore.com";
              expiration="Fri, 14 Jun 1991 19:13:14 -0400 (EDT)"

Content-type: application/postscript
Content-ID: <id42@guppylake.bellcore.com>

--42
Content-Type: message/external-body;
              access-type=mail-server
              server="listserv@bogus.bitnet";
              expiration="Fri, 14 Jun 1991 19:13:14 -0400 (EDT)"

Content-type: application/postscript
Content-ID: <id42@guppylake.bellcore.com>

get RFC-MIME.DOC

--42--
```







Note that in the above examples, the default Content-transfer-encoding of "7bit" is assumed for the external postscript data.

Like the "message/partial" type, the "message/external-body" media type is intended to be transparent, that is, to convey the data type in the external body rather than to convey a message with a body of that type. Thus the headers on the outer and inner parts must be merged using the same rules as for "message/partial". In particular, this means that the Content-type and Subject fields are overridden, but the From field is preserved.

Note that since the external bodies are not transported along with the external body reference, they need not conform to transport limitations that apply to the reference itself. In particular, Internet mail transports may impose 7bit and line length limits, but these do not automatically apply to binary external body references. Thus a Content-Transfer-Encoding is not generally necessary, though it is permitted.

Note that the body of a message of type "message/external-body" is governed by the basic syntax for an RFC 822 message. In particular, anything before the first consecutive pair of CRLFs is header information, while anything after it is body information, which is ignored for most access-types.

### 5.2.4. Other Message Subtypes

MIME implementations must in general treat unrecognized subtypes of "message" as being equivalent to "application/octet-stream".

Future subtypes of "message" intended for use with email should be restricted to "7bit" encoding. A type other than "message" should be used if restriction to "7bit" is not possible.

## 6. Experimental Media Type Values

A media type value beginning with the characters "X-" is a private value, to be used by consenting systems by mutual agreement. Any format without a rigorous and public definition must be named with an "X-" prefix, and publicly specified values shall never begin with "X-". (Older versions of the widely used Andrew system use the "X-BE2" name, so new systems should probably choose a different name.)

In general, the use of "X-" top-level types is strongly discouraged. Implementors should invent subtypes of the existing types whenever possible. In many cases, a subtype of "application" will be more appropriate than a new top-level type.

## 7. Summary

The five discrete media types provide provide a standardized mechanism for tagging entities as "audio", "image", or several other kinds of data. The composite "multipart" and "message" media types allow mixing and hierarchical structuring of entities of different types in a single message. A distinguished parameter syntax allows further specification of data format details, particularly the specification of alternate character sets. Additional optional header fields provide mechanisms for certain extensions deemed desirable by many implementors. Finally, a number of useful media types are defined for general use by consenting user agents, notably "message/partial" and "message/external-body".

## 9. Security Considerations

Security issues are discussed in the context of the "application/postscript" type, the "message/external-body" type, and in RFC 2048. Implementors should pay special attention to the security implications of any media types that can cause the remote execution of any actions in the recipient's environment. In such cases, the discussion of the "application/postscript" type may serve as a model for considering other media types with remote execution capabilities.







## 9. Authors' Addresses

For more information, the authors of this document are best contacted via Internet mail:

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790
USA

Phone: +1 818 919 3600
Fax: +1 818 919 3614
EMail: ned@innosoft.com

Nathaniel S. Borenstein
First Virtual Holdings
25 Washington Avenue
Morristown, NJ 07960
USA

Phone: +1 201 540 8967
Fax: +1 201 993 3032
EMail: nsb@nsb.fv.com

MIME is a result of the work of the Internet Engineering Task Force Working Group on RFC 822 Extensions. The chairman of that group, Greg Vaudreuil, may be reached at:

Gregory M. Vaudreuil
Octel Network Services
17080 Dallas Parkway
Dallas, TX 75248-1905
USA

EMail: Greg.Vaudreuil@Octel.Com

## Appendix A -- Collected Grammar

This appendix contains the complete BNF grammar for all the syntax specified by this document.

By itself, however, this grammar is incomplete. It refers by name to several syntax rules that are defined by RFC 822. Rather than reproduce those definitions here, and risk unintentional differences between the two, this document simply refers the reader to RFC 822 for the remaining definitions. Wherever a term is undefined, it refers to the RFC 822 definition.

```
boundary := 0*69<bchars> bcharsnospace

bchars := bcharsnospace / " "

bcharsnospace := DIGIT / ALPHA / "'" / "(" / ")" /
                 "+" / "_" / "," / "-" / "." /
                 "/" / ":" / "=" / "?"

body-part := <"message" as defined in RFC 822, with all
              header fields optional, not starting with the
              specified dash-boundary, and with the
              delimiter not occurring anywhere in the
              body part.  Note that the semantics of a
              part differ from the semantics of a message,
              as described in the text.>

close-delimiter := delimiter "--"

dash-boundary := "--" boundary
                 ; boundary taken from the value of
                 ; boundary parameter of the
                 ; Content-Type field.

delimiter := CRLF dash-boundary

discard-text := *(*text CRLF)
                ; May be ignored or discarded.

encapsulation := delimiter transport-padding
                 CRLF body-part

epilogue := discard-text

multipart-body := [preamble CRLF]
                  dash-boundary transport-padding CRLF
                  body-part *encapsulation
```





```
                  close-delimiter transport-padding
                  [CRLF epilogue]

preamble := discard-text

transport-padding := *LWSP-char
                     ; Composers MUST NOT generate
                     ; non-zero length transport
                     ; padding, but receivers MUST
                     ; be able to handle padding
                     ; added by message transports.
```

Network Working Group
Request for Comments: 2047
Obsoletes: 1521, 1522, 1590
Category: Standards Track

K. Moore
University of Tennessee

November 1996

# MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text

## Status of this Memo



## Abstract

STD 11, RFC 822, defines a message representation protocol specifying considerable detail about US-ASCII message headers, and leaves the message content, or message body, as flat US-ASCII text. This set of documents, collectively called the Multipurpose Internet Mail Extensions, or MIME, redefines the format of messages to allow for

(1) textual message bodies in character sets other than US-ASCII,

(2) an extensible set of different formats for non-textual message bodies,

(3) multi-part message bodies, and

(4) textual header information in character sets other than US-ASCII.

These documents are based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said so little about message bodies, these documents are largely orthogonal to (rather than a revision of) RFC 822.

This particular document is the third document in the series. It describes extensions to RFC 822 to allow non-US-ASCII text data in Internet mail header fields.





Other documents in this series include:

+ RFC 2045, which specifies the various headers used to describe the structure of MIME messages.

+ RFC 2046, which defines the general structure of the MIME media typing system and defines an initial set of media types,

+ RFC 2048, which specifies various IANA registration procedures for MIME-related facilities, and

+ RFC 2049, which describes MIME conformance criteria and provides some illustrative examples of MIME message formats, acknowledgements, and the bibliography.

These documents are revisions of RFCs 1521, 1522, and 1590, which themselves were revisions of RFCs 1341 and 1342. An appendix in RFC 2049 describes differences and changes from previous versions.

## 1. Introduction

RFC 2045 describes a mechanism for denoting textual body parts which are coded in various character sets, as well as methods for encoding such body parts as sequences of printable US-ASCII characters. This memo describes similar techniques to allow the encoding of non-ASCII text in various portions of a RFC 822 [2] message header, in a manner which is unlikely to confuse existing message handling software.

Like the encoding techniques described in RFC 2045, the techniques outlined here were designed to allow the use of non-ASCII characters in message headers in a way which is unlikely to be disturbed by the quirks of existing Internet mail handling programs. In particular, some mail relaying programs are known to (a) delete some message header fields while retaining others, (b) rearrange the order of addresses in To or Cc fields, (c) rearrange the (vertical) order of header fields, and/or (d) "wrap" message headers at different places than those in the original message. In addition, some mail reading programs are known to have difficulty correctly parsing message headers which, while legal according to RFC 822, make use of backslash-quoting to "hide" special characters such as "<", ",", or ":", or which exploit other infrequently-used features of that specification.

While it is unfortunate that these programs do not correctly interpret RFC 822 headers, to "break" these programs would cause severe operational problems for the Internet mail system. The extensions described in this memo therefore do not rely on little-used features of RFC 822.





Instead, certain sequences of "ordinary" printable ASCII characters (known as "encoded-words") are reserved for use as encoded data. The syntax of encoded-words is such that they are unlikely to "accidentally" appear as normal text in message headers. Furthermore, the characters used in encoded-words are restricted to those which do not have special meanings in the context in which the encoded-word appears.

Generally, an "encoded-word" is a sequence of printable ASCII characters that begins with "=?", ends with "?=", and has two "?"s in between. It specifies a character set and an encoding method, and also includes the original text encoded as graphic ASCII characters, according to the rules for that encoding method.

A mail composer that implements this specification will provide a means of inputting non-ASCII text in header fields, but will translate these fields (or appropriate portions of these fields) into encoded-words before inserting them into the message header.

A mail reader that implements this specification will recognize encoded-words when they appear in certain portions of the message header. Instead of displaying the encoded-word "as is", it will reverse the encoding and display the original text in the designated character set.

NOTES

This memo relies heavily on notation and terms defined RFC 822 and RFC 2045. In particular, the syntax for the ABNF used in this memo is defined in RFC 822, as well as many of the terminal or nonterminal symbols from RFC 822 are used in the grammar for the header extensions defined here. Among the symbols defined in RFC 822 and referenced in this memo are: 'addr-spec', 'atom', 'CHAR', 'comment', 'CTLs', 'ctext', 'linear-white-space', 'phrase', 'quoted-pair'. 'quoted-string', 'SPACE', and 'word'. Successful implementation of this protocol extension requires careful attention to the RFC 822 definitions of these terms.

When the term "ASCII" appears in this memo, it refers to the "7-Bit American Standard Code for Information Interchange", ANSI X3.4-1986. The MIME charset name for this character set is "US-ASCII". When not specifically referring to the MIME charset name, this document uses the term "ASCII", both for brevity and for consistency with RFC 822. However, implementors are warned that the character set name must be spelled "US-ASCII" in MIME message and body part headers.





This memo specifies a protocol for the representation of non-ASCII text in message headers. It specifically DOES NOT define any translation between "8-bit headers" and pure ASCII headers, nor is any such translation assumed to be possible.

## 2. Syntax of encoded-words

An 'encoded-word' is defined by the following ABNF grammar. The notation of RFC 822 is used, with the exception that white space characters MUST NOT appear between components of an 'encoded-word'.

```
encoded-word = "=?" charset "?" encoding "?" encoded-text "?="

charset = token    ; see section 3

encoding = token   ; see section 4

token = 1*<Any CHAR except SPACE, CTLs, and especials>

especials = "(" / ")" / "<" / ">" / "@" / "," / ";" / ":" / "
            <"> / "/" / "[" / "]" / "?" / "." / "="

encoded-text = 1*<Any printable ASCII character other than "?"
                  or SPACE>
                  ; (but see "Use of encoded-words in message
                  ; headers", section 5)
```

Both 'encoding' and 'charset' names are case-independent. Thus the charset name "ISO-8859-1" is equivalent to "iso-8859-1", and the encoding named "Q" may be spelled either "Q" or "q".

An 'encoded-word' may not be more than 75 characters long, including 'charset', 'encoding', 'encoded-text', and delimiters. If it is desirable to encode more text than will fit in an 'encoded-word' of 75 characters, multiple 'encoded-word's (separated by CRLF SPACE) may be used.

While there is no limit to the length of a multiple-line header field, each line of a header field that contains one or more 'encoded-word's is limited to 76 characters.

The length restrictions are included both to ease interoperability through internetwork mail gateways, and to impose a limit on the amount of lookahead a header parser must employ (while looking for a final ?= delimiter) before it can decide whether a token is an "encoded-word" or something else.





IMPORTANT: 'encoded-word's are designed to be recognized as 'atom's by an RFC 822 parser. As a consequence, unencoded white space characters (such as SPACE and HTAB) are FORBIDDEN within an 'encoded-word'. For example, the character sequence

```
=?iso-8859-1?q?this is some text?=
```

would be parsed as four 'atom's, rather than as a single 'atom' (by an RFC 822 parser) or 'encoded-word' (by a parser which understands 'encoded-words'). The correct way to encode the string "this is some text" is to encode the SPACE characters as well, e.g.

```
=?iso-8859-1?q?this=20is=20some=20text?=
```

The characters which may appear in 'encoded-text' are further restricted by the rules in section 5.

## 3. Character sets

The 'charset' portion of an 'encoded-word' specifies the character set associated with the unencoded text. A 'charset' can be any of the character set names allowed in an MIME "charset" parameter of a "text/plain" body part, or any character set name registered with IANA for use with the MIME text/plain content-type.

Some character sets use code-switching techniques to switch between "ASCII mode" and other modes. If unencoded text in an 'encoded-word' contains a sequence which causes the charset interpreter to switch out of ASCII mode, it MUST contain additional control codes such that ASCII mode is again selected at the end of the 'encoded-word'. (This rule applies separately to each 'encoded-word', including adjacent 'encoded-word's within a single header field.)

When there is a possibility of using more than one character set to

represent the text in an 'encoded-word', and in the absence of private agreements between sender and recipients of a message, it is recommended that members of the ISO-8859-* series be used in preference to other character sets.

## 4. Encodings

Initially, the legal values for "encoding" are "Q" and "B". These encodings are described below. The "Q" encoding is recommended for use when most of the characters to be encoded are in the ASCII character set; otherwise, the "B" encoding should be used. Nevertheless, a mail reader which claims to recognize 'encoded-word's MUST be able to accept either encoding for any character set which it supports.





Only a subset of the printable ASCII characters may be used in 'encoded-text'. Space and tab characters are not allowed, so that the beginning and end of an 'encoded-word' are obvious. The "?" character is used within an 'encoded-word' to separate the various portions of the 'encoded-word' from one another, and thus cannot appear in the 'encoded-text' portion. Other characters are also illegal in certain contexts. For example, an 'encoded-word' in a 'phrase' preceding an address in a From header field may not contain any of the "specials" defined in RFC 822. Finally, certain other characters are disallowed in some contexts, to ensure reliability for messages that pass through internetwork mail gateways.

The "B" encoding automatically meets these requirements. The "Q" encoding allows a wide range of printable characters to be used in non-critical locations in the message header (e.g., Subject), with fewer characters available for use in other locations.

### 4.1. The "B" encoding

The "B" encoding is identical to the "BASE64" encoding defined by RFC 2045.

### 4.2. The "Q" encoding

The "Q" encoding is similar to the "Quoted-Printable" content-transfer-encoding defined in RFC 2045. It is designed to allow text containing mostly ASCII characters to be decipherable on an ASCII terminal without decoding.

(1) Any 8-bit value may be represented by a "=" followed by two hexadecimal digits. For example, if the character set in use were ISO-8859-1, the "=" character would thus be encoded as "=3D", and a SPACE by "=20". (Upper case should be used for hexadecimal digits "A" through "F".)

(2) The 8-bit hexadecimal value 20 (e.g., ISO-8859-1 SPACE) may be represented as "_" (underscore, ASCII 95.). (This character may not pass through some internetwork mail gateways, but its use will greatly enhance readability of "Q" encoded data with mail readers that do not support this encoding.) Note that the "_" always represents hexadecimal 20, even if the SPACE character occupies a different code position in the character set in use.

(3) 8-bit values which correspond to printable ASCII characters other than "=", "?", and "_" (underscore), MAY be represented as those characters. (But see section 5 for restrictions.) In particular, SPACE and TAB MUST NOT be represented as themselves within encoded words.





## 5. Use of encoded-words in message headers

An 'encoded-word' may appear in a message header or body part header according to the following rules:

(1) An 'encoded-word' may replace a 'text' token (as defined by RFC 822) in any Subject or Comments header field, any extension message header field, or any MIME body part field for which the field body is defined as '*text'. An 'encoded-word' may also appear in any user-defined ("X-") message or body part header field.

Ordinary ASCII text and 'encoded-word's may appear together in the same header field. However, an 'encoded-word' that appears in a header field defined as '*text' MUST be separated from any adjacent 'encoded-word' or 'text' by 'linear-white-space'.

(2) An 'encoded-word' may appear within a 'comment' delimited by "(" and ")", i.e., wherever a 'ctext' is allowed. More precisely, the RFC 822 ABNF definition for 'comment' is amended as follows:

comment = "(" *(ctext / quoted-pair / comment / encoded-word) ")"

A "Q"-encoded 'encoded-word' which appears in a 'comment' MUST NOT contain the characters "(", ")" or "
'encoded-word' that appears in a 'comment' MUST be separated from any adjacent 'encoded-word' or 'ctext' by 'linear-white-space'.

It is important to note that 'comment's are only recognized inside "structured" field bodies. In fields whose bodies are defined as '*text', "(" and ")" are treated as ordinary characters rather than comment delimiters, and rule (1) of this section applies. (See RFC 822, sections 3.1.2 and 3.1.3)

(3) As a replacement for a 'word' entity within a 'phrase', for example, one that precedes an address in a From, To, or Cc header. The ABNF definition for 'phrase' from RFC 822 thus becomes:

phrase = 1*( encoded-word / word )

In this case the set of characters that may be used in a "Q"-encoded 'encoded-word' is restricted to: <upper and lower case ASCII letters, decimal digits, "!", "*", "+", "-", "/", "=", and "_" (underscore, ASCII 95.)>. An 'encoded-word' that appears within a 'phrase' MUST be separated from any adjacent 'word', 'text' or 'special' by 'linear-white-space'.

These are the ONLY locations where an 'encoded-word' may appear. In particular:

+ An 'encoded-word' MUST NOT appear in any portion of an 'addr-spec'.

+ An 'encoded-word' MUST NOT appear within a 'quoted-string'.

+ An 'encoded-word' MUST NOT be used in a Received header field.

+ An 'encoded-word' MUST NOT be used in parameter of a MIME Content-Type or Content-Disposition field, or in any structured field body except within a 'comment' or 'phrase'.

The 'encoded-text' in an 'encoded-word' must be self-contained; 'encoded-text' MUST NOT be continued from one 'encoded-word' to another. This implies that the 'encoded-text' portion of a "B" 'encoded-word' will be a multiple of 4 characters long; for a "Q" 'encoded-word', any "=" character that appears in the 'encoded-text' portion will be followed by two hexadecimal characters.

Each 'encoded-word' MUST encode an integral number of octets. The 'encoded-text' in each 'encoded-word' must be well-formed according to the encoding specified; the 'encoded-text' may not be continued in the next 'encoded-word'. (For example, "=?charset?Q?=?= =?charset?Q?AB?=" would be illegal, because the two hex digits "AB" must follow the "=" in the same 'encoded-word'.)

Each 'encoded-word' MUST represent an integral number of characters. A multi-octet character may not be split across adjacent 'encoded-word's.

Only printable and white space character data should be encoded using this scheme. However, since these encoding schemes allow the encoding of arbitrary octet values, mail readers that implement this decoding should also ensure that display of the decoded data on the recipient's terminal will not cause unwanted side-effects.

Use of these methods to encode non-textual data (e.g., pictures or sounds) is not defined by this memo. Use of 'encoded-word's to represent strings of purely ASCII characters is allowed, but discouraged. In rare cases it may be necessary to encode ordinary text that looks like an 'encoded-word'.





## 6. Support of 'encoded-word's by mail readers

### 6.1. Recognition of 'encoded-word's in message headers

A mail reader must parse the message and body part headers according to the rules in RFC 822 to correctly recognize 'encoded-word's.

'encoded-word's are to be recognized as follows:

(1) Any message or body part header field defined as '*text', or any user-defined header field, should be parsed as follows: Beginning at the start of the field-body and immediately following each occurrence of 'linear-white-space', each sequence of up to 75 printable characters (not containing any 'linear-white-space') should be examined to see if it is an 'encoded-word' according to the syntax rules in section 2. Any other sequence of printable characters should be treated as ordinary ASCII text.

(2) Any header field not defined as '*text' should be parsed according to the syntax rules for that header field. However, any 'word' that appears within a 'phrase' should be treated as an 'encoded-word' if it meets the syntax rules in section 2. Otherwise it should be treated as an ordinary 'word'.

(3) Within a 'comment', any sequence of up to 75 printable characters (not containing 'linear-white-space'), that meets the syntax rules in section 2, should be treated as an 'encoded-word'. Otherwise it should be treated as normal comment text.

(4) A MIME-Version header field is NOT required to be present for 'encoded-word's to be interpreted according to this specification. One reason for this is that the mail reader is not expected to parse the entire message header before displaying lines that may contain 'encoded-word's.

### 6.2. Display of 'encoded-word's

Any 'encoded-word's so recognized are decoded, and if possible, the resulting unencoded text is displayed in the original character set.

NOTE: Decoding and display of encoded-words occurs *after* a structured field body is parsed into tokens. It is therefore possible to hide 'special' characters in encoded-words which, when displayed, will be indistinguishable from 'special' characters in the surrounding text. For this and other reasons, it is NOT generally possible to translate a message header containing 'encoded-word's to an unencoded form which can be parsed by an RFC 822 mail reader.





When displaying a particular header field that contains multiple 'encoded-word's, any 'linear-white-space' that separates a pair of adjacent 'encoded-word's is ignored. (This is to allow the use of multiple 'encoded-word's to represent long strings of unencoded text, without having to separate 'encoded-word's where spaces occur in the unencoded text.)

In the event other encodings are defined in the future, and the mail reader does not support the encoding used, it may either (a) display the 'encoded-word' as ordinary text, or (b) substitute an appropriate message indicating that the text could not be decoded.

If the mail reader does not support the character set used, it may (a) display the 'encoded-word' as ordinary text (i.e., as it appears in the header), (b) make a "best effort" to display using such characters as are available, or (c) substitute an appropriate message indicating that the decoded text could not be displayed.

If the character set being used employs code-switching techniques, display of the encoded text implicitly begins in "ASCII mode". In addition, the mail reader must ensure that the output device is once again in "ASCII mode" after the 'encoded-word' is displayed.

### 6.3. Mail reader handling of incorrectly formed 'encoded-word's

It is possible that an 'encoded-word' that is legal according to the syntax defined in section 2, is incorrectly formed according to the rules for the encoding being used. For example:

(1) An 'encoded-word' which contains characters which are not legal for a particular encoding (for example, a "-" in the "B" encoding, or a SPACE or HTAB in either the "B" or "Q" encoding), is incorrectly formed.

(2) Any 'encoded-word' which encodes a non-integral number of characters or octets is incorrectly formed.

A mail reader need not attempt to display the text associated with an 'encoded-word' that is incorrectly formed. However, a mail reader MUST NOT prevent the display or handling of a message because an 'encoded-word' is incorrectly formed.

## 7. Conformance

A mail composing program claiming compliance with this specification MUST ensure that any string of non-white-space printable ASCII characters within a '*text' or '*ctext' that begins with "=?" and ends with "?=" be a valid 'encoded-word'. ("begins" means: at the





start of the field-body, immediately following 'linear-white-space', or immediately following a "(" for an 'encoded-word' within '*ctext'; "ends" means: at the end of the field-body, immediately preceding 'linear-white-space', or immediately preceding a ")" for an 'encoded-word' within '*ctext'.) In addition, any 'word' within a 'phrase' that begins with "=?" and ends with "?=" must be a valid 'encoded-word'.

A mail reading program claiming compliance with this specification must be able to distinguish 'encoded-word's from 'text', 'ctext', or 'word's, according to the rules in section 6, anytime they appear in appropriate places in message headers. It must support both the "B" and "Q" encodings for any character set which it supports. The program must be able to display the unencoded text if the character set is "US-ASCII". For the ISO-8859-* character sets, the mail reading program must at least be able to display the characters which are also in the ASCII set.

## 8. Examples

The following are examples of message headers containing 'encoded-word's:

```
From: =?US-ASCII?Q?Keith_Moore?= <moore@cs.utk.edu>
To: =?ISO-8859-1?Q?Keld_J=F8rn_Simonsen?= <keld@dkuug.dk>
CC: =?ISO-8859-1?Q?Andr=E9?= Pirard <PIRARD@vm1.ulg.ac.be>
Subject: =?ISO-8859-1?B?SWYgeW91IGNhbiByZWFkIHRoaXMgeW8=?=
 =?ISO-8859-2?B?dSB1bmRlcnN0YW5kIHRoZSBleGFtcGxlLg==?=
```

Note: In the first 'encoded-word' of the Subject field above, the last "=" at the end of the 'encoded-text' is necessary because each 'encoded-word' must be self-contained (the "=" character completes a group of 4 base64 characters representing 2 octets). An additional octet could have been encoded in the first 'encoded-word' (so that the encoded-word would contain an exact multiple of 3 encoded octets), except that the second 'encoded-word' uses a different 'charset' than the first one.

```
From: =?ISO-8859-1?Q?Olle_J=E4rnefors?= <ojarnef@admin.kth.se>
To: ietf-822@dimacs.rutgers.edu, ojarnef@admin.kth.se
Subject: Time for ISO 10646?

To: Dave Crocker <dcrocker@mordor.stanford.edu>
Cc: ietf-822@dimacs.rutgers.edu, paf@comsol.se
From: =?ISO-8859-1?Q?Patrik_F=E4ltstr=F6m?= <paf@nada.kth.se>
Subject: Re: RFC-HDR care and feeding
```





```
From: Nathaniel Borenstein <nsb@thumper.bellcore.com>
      (=?iso-8859-8?b?7eXs+SDv4SDp7Oj08A==?=)
To: Greg Vaudreuil <gvaudre@NRI.Reston.VA.US>, Ned Freed
   <ned@innosoft.com>, Keith Moore <moore@cs.utk.edu>
Subject: Test of new header generator
MIME-Version: 1.0
Content-type: text/plain; charset=ISO-8859-1
```

The following examples illustrate how text containing 'encoded-word's which appear in a structured field body. The rules are slightly different for fields defined as '*text' because "(" and ")" are not recognized as 'comment' delimiters. [Section 5, paragraph (1)].

In each of the following examples, if the same sequence were to occur in a '*text' field, the "displayed as" form would NOT be treated as encoded words, but be identical to the "encoded form". This is because each of the encoded-words in the following examples is adjacent to a "(" or ")" character.

| encoded form | displayed as |
|---|---|
| (=?ISO-8859-1?Q?a?=) | (a) |
| (=?ISO-8859-1?Q?a?= b) | (a b) |

Within a 'comment', white space MUST appear between an 'encoded-word' and surrounding text. [Section 5, paragraph (2)]. However, white space is not needed between the initial "(" that begins the 'comment', and the 'encoded-word'.

| | |
|---|---|
| (=?ISO-8859-1?Q?a?= =?ISO-8859-1?Q?b?=) | (ab) |

White space between adjacent 'encoded-word's is not displayed.

(=?ISO-8859-1?Q?a?=  =?ISO-8859-1?Q?b?=)    (ab)

Even multiple SPACEs between 'encoded-word's are ignored for the purpose of display.

(=?ISO-8859-1?Q?a?=
=?ISO-8859-1?Q?b?=)    (ab)

Any amount of linear-space-white between 'encoded-word's, even if it includes a CRLF followed by one or more SPACEs, is ignored for the purposes of display.





(=?ISO-8859-1?Q?a_b?=)    (a b)

In order to cause a SPACE to be displayed within a portion of encoded text, the SPACE MUST be encoded as part of the 'encoded-word'.

(=?ISO-8859-1?Q?a?= =?ISO-8859-2?Q?_b?=)    (a b)

In order to cause a SPACE to be displayed between two strings of encoded text, the SPACE MAY be encoded as part of one of the 'encoded-word's.

## 9. References

[RFC 822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[RFC 2049] Borenstein, N., and N. Freed, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.

[RFC 2045] Borenstein, N., and N. Freed, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC 2046] Borenstein N., and N. Freed, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC 2048] Freed, N., Klensin, J., and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", RFC 2048, November 1996.



## 10. Security Considerations

Security issues are not discussed in this memo.

## 11. Acknowledgements

The author wishes to thank Nathaniel Borenstein, Issac Chan, Lutz Donnerhacke, Paul Eggert, Ned Freed, Andreas M. Kirchwitz, Olle Jarnefors, Mike Rosin, Yutaka Sato, Bart Schaefer, and Kazuhiko Yamamoto, for their helpful advice, insightful comments, and illuminating questions in response to earlier versions of this specification.

## 12. Author's Address

Keith Moore
University of Tennessee
107 Ayres Hall
Knoxville TN 37996-1301

EMail: moore@cs.utk.edu

Appendix - changes since RFC 1522 (in no particular order)

+ explicitly state that the MIME-Version is not requried to use 'encoded-word's.

+ add explicit note that SPACEs and TABs are not allowed within 'encoded-word's, explaining that an 'encoded-word' must look like an 'atom' to an RFC822 parser.values, to be precise).

+ add examples from Olle Jarnefors (thanks!) which illustrate how encoded-words with adjacent linear-white-space are displayed.

+ explicitly list terms defined in RFC822 and referenced in this memo

+ fix transcription typos that caused one or two lines and a couple of characters to disappear in the resulting text, due to nroff quirks.

+ clarify that encoded-words are allowed in '*text' fields in both RFC822 headers and MIME body part headers, but NOT as parameter values.

+ clarify the requirement to switch back to ASCII within the encoded portion of an 'encoded-word', for any charset that uses code switching sequences.

+ add a note about 'encoded-word's being delimited by "(" and ")" within a comment, but not in a *text (how bizarre!).

+ fix the Andre Pirard example to get rid of the trailing "_" after the =E9.  (no longer needed post-1342).

+ clarification: an 'encoded-word' may appear immediately following the initial "(" or immediately before the final ")" that delimits a comment, not just adjacent to "(" and ")" *within* *ctext.

+ add a note to explain that a "B" 'encoded-word' will always have a multiple of 4 characters in the 'encoded-text' portion.

+ add note about the "=" in the examples

+ note that processing of 'encoded-word's occurs *after* parsing, and some of the implications thereof.

+ explicitly state that you can't expect to translate between 1522 and either vanilla 822 or so-called "8-bit headers".

+ explicitly state that 'encoded-word's are not valid within a 'quoted-string'.

Network Working Group
Request for Comments: 2048
BCP: 13
Obsoletes: 1521, 1522, 1590
Category: Best Current Practice

N. Freed
Innosoft
J. Klensin
MCI
J. Postel
ISI

November 1996

# Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures

## Status of this Memo



## Abstract

STD 11, RFC 822, defines a message representation protocol specifying considerable detail about US-ASCII message headers, and leaves the message content, or message body, as flat US-ASCII text. This set of documents, collectively called the Multipurpose Internet Mail Extensions, or MIME, redefines the format of messages to allow for

(1) textual message bodies in character sets other than US-ASCII,

(2) an extensible set of different formats for non-textual message bodies,

(3) multi-part message bodies, and

(4) textual header information in character sets other than US-ASCII.

These documents are based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said so little about message bodies, these documents are largely orthogonal to (rather than a revision of) RFC 822.





This fourth document, RFC 2048, specifies various IANA registration procedures for the following MIME facilities:

(1) media types,

(2) external body access types,

(3) content-transfer-encodings.

Registration of character sets for use in MIME is covered elsewhere and is no longer addressed by this document.

These documents are revisions of RFCs 1521 and 1522, which themselves were revisions of RFCs 1341 and 1342. An appendix in RFC 2049 describes differences and changes from previous versions.

## Table of Contents

## 1. Introduction

Recent Internet protocols have been carefully designed to be easily extensible in certain areas. In particular, MIME [RFC 2045] is an open-ended framework and can accommodate additional object types, character sets, and access methods without any changes to the basic protocol. A registration process is needed, however, to ensure that the set of such values is developed in an orderly, well-specified, and public manner.

This document defines registration procedures which use the Internet Assigned Numbers Authority (IANA) as a central registry for such values.

Historical Note: The registration process for media types was initially defined in the context of the asynchronous Internet mail environment. In this mail environment there is a need to limit the number of possible media types to increase the likelihood of interoperability when the capabilities of the remote mail system are not known. As media types are used in new environments, where the proliferation of media types is not a hindrance to interoperability, the original procedure was excessively restrictive and had to be generalized.





## 2. Media Type Registration

Registration of a new media type or types starts with the construction of a registration proposal. Registration may occur in several different registration trees, which have different requirements as discussed below. In general, the new registration proposal is circulated and reviewed in a fashion appropriate to the tree involved. The media type is then registered if the proposal is acceptable. The following sections describe the requirements and procedures used for each of the different registration trees.

### 2.1. Registration Trees and Subtype Names

In order to increase the efficiency and flexibility of the registration process, different structures of subtype names may be registered to accomodate the different natural requirements for, e.g., a subtype that will be recommended for wide support and implementation by the Internet Community or a subtype that is used to move files associated with proprietary software. The following subsections define registration "trees", distinguished by the use of faceted names (e.g., names of the form "tree.subtree...type"). Note that some media types defined prior to this document do not conform to the naming conventions described below. See Appendix A for a discussion of them.

#### 2.1.1. IETF Tree

The IETF tree is intended for types of general interest to the Internet Community. Registration in the IETF tree requires approval by the IESG and publication of the media type registration as some form of RFC.

Media types in the IETF tree are normally denoted by names that are not explicitly faceted, i.e., do not contain period (".", full stop) characters.

The "owner" of a media type registration in the IETF tree is assumed to be the IETF itself. Modification or alteration of the specification requires the same level of processing (e.g. standards track) required for the initial registration.

#### 2.1.2. Vendor Tree

The vendor tree is used for media types associated with commercially available products. "Vendor" or "producer" are construed as equivalent and very broadly in this context.

A registration may be placed in the vendor tree by anyone who has need to interchange files associated with the particular product. However, the registration formally belongs to the vendor or organization producing the software or file format. Changes to the specification will be made at their request, as discussed in subsequent sections.

Registrations in the vendor tree will be distinguished by the leading facet "vnd.". That may be followed, at the discretion of the registration, by either a media type name from a well-known producer (e.g., "vnd.mudpie") or by an IANA-approved designation of the producer's name which is then followed by a media type or product designation (e.g., vnd.bigcompany.funnypictures).

While public exposure and review of media types to be registered in the vendor tree is not required, using the ietf-types list for review is strongly encouraged to improve the quality of those specifications. Registrations in the vendor tree may be submitted directly to the IANA.

#### 2.1.3. Personal or Vanity Tree

Registrations for media types created experimentally or as part of products that are not distributed commercially may be registered in the personal or vanity tree. The registrations are distinguished by the leading facet "prs.".

The owner of "personal" registrations and associated specifications is the person or entity making the registration, or one to whom responsibility has been transferred as described below.

While public exposure and review of media types to be registered in the personal tree is not required, using the ietf-types list for review is strongly encouraged to improve the quality of those specifications. Registrations in the personl tree may be submitted directly to the IANA.

#### 2.1.4. Special `x.' Tree

For convenience and symmetry with this registration scheme, media type names with "x." as the first facet may be used for the same purposes for which names starting in "x-" are normally used. These types are unregistered, experimental, and should be used only with the active agreement of the parties exchanging them.







However, with the simplified registration procedures described above for vendor and personal trees, it should rarely, if ever, be necessary to use unregistered experimental types, and as such use of both "x-" and "x." forms is discouraged.

#### 2.1.5. Additional Registration Trees

From time to time and as required by the community, the IANA may, with the advice and consent of the IESG, create new top-level registration trees. It is explicitly assumed that these trees may be created for external registration and management by well-known permanent bodies, such as scientific societies for media types specific to the sciences they cover. In general, the quality of review of specifications for one of these additional registration trees is expected to be equivalent to that which IETF would give to registrations in its own tree. Establishment of these new trees will be announced through RFC publication approved by the IESG.

### 2.2. Registration Requirements

Media type registration proposals are all expected to conform to various requirements laid out in the following sections. Note that requirement specifics sometimes vary depending on the registration tree, again as detailed in the following sections.

#### 2.2.1. Functionality Requirement

Media types must function as an actual media format: Registration of things that are better thought of as a transfer encoding, as a character set, or as a collection of separate entities of another type, is not allowed. For example, although applications exist to decode the base64 transfer encoding [RFC 2045], base64 cannot be registered as a media type.

This requirement applies regardless of the registration tree involved.

#### 2.2.2. Naming Requirements

All registered media types must be assigned MIME type and subtype names. The combination of these names then serves to uniquely identify the media type and the format of the subtype name identifies the registration tree.

The choice of top-level type name must take the nature of media type involved into account. For example, media normally used for representing still images should be a subtype of the image content type, whereas media capable of representing audio information belongs

under the audio content type. See RFC 2046 for additional information on the basic set of top-level types and their characteristics.

New subtypes of top-level types must conform to the restrictions of the top-level type, if any. For example, all subtypes of the multipart content type must use the same encapsulation syntax.

In some cases a new media type may not "fit" under any currently defined top-level content type. Such cases are expected to be quite rare. However, if such a case arises a new top-level type can be defined to accommodate it. Such a definition must be done via standards-track RFC; no other mechanism can be used to define additional top-level content types.

These requirements apply regardless of the registration tree involved.

#### 2.2.3. Parameter Requirements

Media types may elect to use one or more MIME content type parameters, or some parameters may be automatically made available to the media type by virtue of being a subtype of a content type that defines a set of parameters applicable to any of its subtypes. In either case, the names, values, and meanings of any parameters must be fully specified when a media type is registered in the IETF tree, and should be specified as completely as possible when media types are registered in the vendor or personal trees.

New parameters must not be defined as a way to introduce new functionality in types registered in the IETF tree, although new parameters may be added to convey additional information that does not otherwise change existing functionality. An example of this would be a "revision" parameter to indicate a revision level of an external specification such as JPEG. Similar behavior is encouraged for media types registered in the vendor or personal trees but is not required.

#### 2.2.4. Canonicalization and Format Requirements

All registered media types must employ a single, canonical data format, regardless of registration tree.

A precise and openly available specification of the format of each media type is required for all types registered in the IETF tree and must at a minimum be referenced by, if it isn't actually included in, the media type registration proposal itself.







The specifications of format and processing particulars may or may not be publically available for media types registered in the vendor tree, and such registration proposals are explicitly permitted to include only a specification of which software and version produce or process such media types. References to or inclusion of format specifications in registration proposals is encouraged but not required.

Format specifications are still required for registration in the personal tree, but may be either published as RFCs or otherwise deposited with IANA. The deposited specifications will meet the same criteria as those required to register a well-known TCP port and, in particular, need not be made public.

Some media types involve the use of patented technology. The registration of media types involving patented technology is specifically permitted. However, the restrictions set forth in RFC 1602 on the use of patented technology in standards-track protocols must be respected when the specification of a media type is part of a standards-track protocol.

#### 2.2.5. Interchange Recommendations

Media types should, whenever possible, interoperate across as many systems and applications as possible. However, some media types will inevitably have problems interoperating across different platforms. Problems with different versions, byte ordering, and specifics of gateway handling can and will arise.

Universal interoperability of media types is not required, but known interoperability issues should be identified whenever possible. Publication of a media type does not require an exhaustive review of interoperability, and the interoperability considerations section is subject to continuing evaluation.

These recommendations apply regardless of the registration tree involved.

#### 2.2.6. Security Requirements

An analysis of security issues is required for for all types registered in the IETF Tree. (This is in accordance with the basic requirements for all IETF protocols.) A similar analysis for media types registered in the vendor or personal trees is encouraged but not required. However, regardless of what security analysis has or has not been done, all descriptions of security issues must be as accurate as possible regardless of registration tree. In particular, a statement that there are "no security issues associated with this

type" must not be confused with "the security issues associates with this type have not been assessed".

There is absolutely no requirement that media types registered in any tree be secure or completely free from risks. Nevertheless, all known security risks must be identified in the registration of a media type, again regardless of registration tree.

The security considerations section of all registrations is subject to continuing evaluation and modification, and in particular may be extended by use of the "comments on media types" mechanism described in subsequent sections.

Some of the issues that should be looked at in a security analysis of a media type are:

(1) Complex media types may include provisions for directives that institute actions on a recipient's files or other resources. In many cases provision is made for originators to specify arbitrary actions in an unrestricted fashion which may then have devastating effects. See the registration of the application/postscript media type in RFC 2046 for an example of such directives and how to handle them.

(2) Complex media types may include provisions for directives that institute actions which, while not directly harmful to the recipient, may result in disclosure of information that either facilitates a subsequent attack or else violates a recipient's privacy in some way. Again, the registration of the application/postscript media type illustrates how such directives can be handled.

(3) A media type might be targeted for applications that require some sort of security assurance but not provide the necessary security mechanisms themselves. For example, a media type could be defined for storage of confidential medical information which in turn requires an external confidentiality service.

### 2.2.7. Usage and Implementation Non-requirements

In the asynchronous mail environment, where information on the capabilities of the remote mail agent is frequently not available to the sender, maximum interoperability is attained by restricting the number of media types used to those "common" formats expected to be widely implemented. This was asserted in the past as a reason to







limit the number of possible media types and resulted in a registration process with a significant hurdle and delay for those registering media types.

However, the need for "common" media types does not require limiting the registration of new media types. If a limited set of media types is recommended for a particular application, that should be asserted by a separate applicability statement specific for the application and/or environment.

As such, universal support and implementation of a media type is NOT a requirement for registration. If, however, a media type is explicitly intended for limited use, this should be noted in its registration.

### 2.2.8. Publication Requirements

Proposals for media types registered in the IETF tree must be published as RFCs. RFC publication of vendor and personal media type proposals is encouraged but not required. In all cases IANA will retain copies of all media type proposals and "publish" them as part of the media types registration tree itself.

Other than in the IETF tree, the registration of a data type does not imply endorsement, approval, or recommendation by IANA or IETF or even certification that the specification is adequate. To become Internet Standards, protocol, data objects, or whatever must go through the IETF standards process. This is too difficult and too lengthy a process for the convenient registration of media types.

The IETF tree exists for media types that do require require a substantive review and approval process with the vendor and personal trees exist for those that do not. It is expected that applicability statements for particular applications will be published from time to time that recommend implementation of, and support for, media types that have proven particularly useful in those contexts.

As discussed above, registration of a top-level type requires standards-track processing and, hence, RFC publication.

### 2.2.9. Additional Information

Various sorts of optional information may be included in the specification of a media type if it is available:

(1) Magic number(s) (length, octet values). Magic numbers are byte sequences that are always present and thus can be used to identify entities as being of a given media

type.

(2) File extension(s) commonly used on one or more platforms to indicate that some file containing a given type of media.

(3) Macintosh File Type code(s) (4 octets) used to label files containing a given type of media.

Such information is often quite useful to implementors and if available should be provided.

### 2.3. Registration Procedure

The following procedure has been implemented by the IANA for review and approval of new media types. This is not a formal standards process, but rather an administrative procedure intended to allow community comment and sanity checking without excessive time delay. For registration in the IETF tree, the normal IETF processes should be followed, treating posting of an internet-draft and announcement on the ietf-types list (as described in the next subsection) as a first step. For registrations in the vendor or personal tree, the initial review step described below may be omitted and the type registered directly by submitting the template and an explanation directly to IANA (at iana@iana.org). However, authors of vendor or personal media type specifications are encouraged to seek community review and comment whenever that is feasible.

#### 2.3.1. Present the Media Type to the Community for Review

Send a proposed media type registration to the "ietf-types@iana.org" mailing list for a two week review period. This mailing list has been established for the purpose of reviewing proposed media and access types. Proposed media types are not formally registered and must not be used; the "x-" prefix specified in RFC 2045 can be used until registration is complete.

The intent of the public posting is to solicit comments and feedback on the choice of type/subtype name, the unambiguity of the references with respect to versions and external profiling information, and a review of any interoperability or security considerations. The submitter may submit a revised registration, or withdraw the registration completely, at any time.







#### 2.3.2. IESG Approval

Media types registered in the IETF tree must be submitted to the IESG for approval.

#### 2.3.3. IANA Registration

Provided that the media type meets the requirements for media types and has obtained approval that is necessary, the author may submit the registration request to the IANA, which will register the media type and make the media type registration available to the community.

### 2.4. Comments on Media Type Registrations

Comments on registered media types may be submitted by members of the community to IANA. These comments will be passed on to the "owner" of the media type if possible. Submitters of comments may request that their comment be attached to the media type registration itself, and if IANA approves of this the comment will be made accessible in conjunction with the type registration itself.

### 2.5. Location of Registered Media Type List

Media type registrations will be posted in the anonymous FTP directory "ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/" and all registered media types will be listed in the periodically issued "Assigned Numbers" RFC [currently STD 2, RFC 1700]. The media type description and other supporting material may also be published as an Informational RFC by sending it to "rfc-editor@isi.edu" (please follow the instructions to RFC authors [RFC-1543]).

### 2.6. IANA Procedures for Registering Media Types

The IANA will only register media types in the IETF tree in response to a communication from the IESG stating that a given registration has been approved. Vendor and personal types will be registered by the IANA automatically and without any formal review as long as the following minimal conditions are met:

(1) Media types must function as an actual media format. In particular, character sets and transfer encodings may not be registered as media types.

(2) All media types must have properly formed type and subtype names. All type names must be defined by a standards-track RFC. All subtype names must be unique, must conform to the MIME grammar for such names, and must contain the proper tree prefix.

(3) Types registered in the personal tree must either provide a format specification or a pointer to one.

(4) Any security considerations given must not be obviously bogus. (It is neither possible nor necessary for the IANA to conduct a comprehensive security review of media type registrations. Nevertheless, IANA has the authority to identify obviously incompetent material and exclude it.)

## 2.7. Change Control

Once a media type has been published by IANA, the author may request a change to its definition. The descriptions of the different registration trees above designate the "owners" of each type of registration. The change request follows the same procedure as the registration request:

(1) Publish the revised template on the ietf-types list.

(2) Leave at least two weeks for comments.

(3) Publish using IANA after formal review if required.

Changes should be requested only when there are serious omission or errors in the published specification. When review is required, a change request may be denied if it renders entities that were valid under the previous definition invalid under the new definition.

The owner of a content type may pass responsibility for the content type to another person or agency by informing IANA and the ietf-types list; this can be done without discussion or review.

The IESG may reassign responsibility for a media type. The most common case of this will be to enable changes to be made to types where the author of the registration has died, moved out of contact or is otherwise unable to make changes that are important to the community.

Media type registrations may not be deleted; media types which are no longer believed appropriate for use can be declared OBSOLETE by a change to their "intended use" field; such media types will be clearly marked in the lists published by IANA.







## 2.8. Registration Template

To: ietf-types@iana.org
Subject: Registration of MIME media type XXX/YYY

MIME media type name:

MIME subtype name:

Required parameters:

Optional parameters:

Encoding considerations:

Security considerations:

Interoperability considerations:

Published specification:

Applications which use this media type:

Additional information:

  Magic number(s):
  File extension(s):
  Macintosh File Type Code(s):

Person & email address to contact for further information:

Intended usage:

(One of COMMON, LIMITED USE or OBSOLETE)

Author/Change controller:

(Any other information that the author deems interesting may be added below this line.)

# 3. External Body Access Types

RFC 2046 defines the message/external-body media type, whereby a MIME entity can act as pointer to the actual body data in lieu of including the data directly in the entity body. Each message/external-body reference specifies an access type, which determines the mechanism used to retrieve the actual body data. RFC 2046 defines an initial set of access types, but allows for the

registration of additional access types to accommodate new retrieval mechanisms.

### 3.1. Registration Requirements

New access type specifications must conform to a number of requirements as described below.

#### 3.1.1. Naming Requirements

Each access type must have a unique name. This name appears in the access-type parameter in the message/external-body content-type header field, and must conform to MIME content type parameter syntax.

#### 3.1.2. Mechanism Specification Requirements

All of the protocols, transports, and procedures used by a given access type must be described, either in the specification of the access type itself or in some other publicly available specification, in sufficient detail for the access type to be implemented by any competent implementor. Use of secret and/or proprietary methods in access types are expressly prohibited. The restrictions imposed by RFC 1602 on the standardization of patented algorithms must be respected as well.

#### 3.1.3. Publication Requirements

All access types must be described by an RFC. The RFC may be informational rather than standards-track, although standard-track review and approval are encouraged for all access types.

#### 3.1.4. Security Requirements

Any known security issues that arise from the use of the access type must be completely and fully described. It is not required that the access type be secure or that it be free from risks, but that the known risks be identified. Publication of a new access type does not require an exhaustive security review, and the security considerations section is subject to continuing evaluation. Additional security considerations should be addressed by publishing revised versions of the access type specification.

### 3.2. Registration Procedure

Registration of a new access type starts with the construction of a draft of an RFC.







#### 3.2.1. Present the Access Type to the Community

Send a proposed access type specification to the "ietf-types@iana.org" mailing list for a two week review period. This mailing list has been established for the purpose of reviewing proposed access and media types. Proposed access types are not formally registered and must not be used.

The intent of the public posting is to solicit comments and feedback on the access type specification and a review of any security considerations.

#### 3.2.2. Access Type Reviewer

When the two week period has passed, the access type reviewer, who is appointed by the IETF Applications Area Director, either forwards the request to iana@isi.edu, or rejects it because of significant objections raised on the list.

Decisions made by the reviewer must be posted to the ietf-types mailing list within 14 days. Decisions made by the reviewer may be appealed to the IESG.

#### 3.2.3. IANA Registration

Provided that the access type has either passed review or has been successfully appealed to the IESG, the IANA will register the access type and make the registration available to the community. The specification of the access type must also be published as an RFC. Informational RFCs are published by sending them to "rfc-editor@isi.edu" (please follow the instructions to RFC authors [RFC-1543]).

### 3.3. Location of Registered Access Type List

Access type registrations will be posted in the anonymous FTP directory "ftp://ftp.isi.edu/in-notes/iana/assignments/access-types/" and all registered access types will be listed in the periodically issued "Assigned Numbers" RFC [currently RFC-1700].

### 3.4. IANA Procedures for Registering Access Types

The identity of the access type reviewer is communicated to the IANA by the IESG. The IANA then only acts in response to access type definitions that either are approved by the access type reviewer and forwarded by the reviewer to the IANA for registration, or in response to a communication from the IESG that an access type definition appeal has overturned the access type reviewer's ruling.

## 4. Transfer Encodings

Transfer encodings are tranformations applied to MIME media types after conversion to the media type's canonical form. Transfer encodings are used for several purposes:

(1) Many transports, especially message transports, can only handle data consisting of relatively short lines of text. There can also be severe restrictions on what characters can be used in these lines of text -- some transports are restricted to a small subset of US-ASCII and others cannot handle certain character sequences. Transfer encodings are used to transform binary data into textual form that can survive such transports. Examples of this sort of transfer encoding include the base64 and quoted-printable transfer encodings defined in RFC 2045.

(2) Image, audio, video, and even application entities are sometimes quite large. Compression algorithms are often quite effective in reducing the size of large entities. Transfer encodings can be used to apply general-purpose non-lossy compression algorithms to MIME entities.

(3) Transport encodings can be defined as a means of representing existing encoding formats in a MIME context.

IMPORTANT: The standardization of a large numbers of different transfer encodings is seen as a significant barrier to widespread interoperability and is expressely discouraged. Nevertheless, the following procedure has been defined to provide a means of defining additional transfer encodings, should standardization actually be justified.

### 4.1. Transfer Encoding Requirements

Transfer encoding specifications must conform to a number of requirements as described below.

#### 4.1.1. Naming Requirements

Each transfer encoding must have a unique name. This name appears in the Content-Transfer-Encoding header field and must conform to the syntax of that field.





#### 4.1.2. Algorithm Specification Requirements

All of the algorithms used in a transfer encoding (e.g. conversion to printable form, compression) must be described in their entirety in the transfer encoding specification. Use of secret and/or proprietary algorithms in standardized transfer encodings are expressly prohibited. The restrictions imposed by RFC 1602 on the standardization of patented algorithms must be respected as well.

#### 4.1.3. Input Domain Requirements

All transfer encodings must be applicable to an arbitrary sequence of octets of any length. Dependence on particular input forms is not allowed.

It should be noted that the 7bit and 8bit encodings do not conform to this requirement. Aside from the undesireability of having specialized encodings, the intent here is to forbid the addition of additional encodings along the lines of 7bit and 8bit.

#### 4.1.4. Output Range Requirements

There is no requirement that a particular tranfer encoding produce a particular form of encoded output. However, the output format for each transfer encoding must be fully and completely documented. In particular, each specification must clearly state whether the output format always lies within the confines of 7bit data, 8bit data, or is simply pure binary data.

#### 4.1.5. Data Integrity and Generality Requirements

All transfer encodings must be fully invertible on any platform; it must be possible for anyone to recover the original data by performing the corresponding decoding operation. Note that this requirement effectively excludes all forms of lossy compression as well as all forms of encryption from use as a transfer encoding.

#### 4.1.6. New Functionality Requirements

All transfer encodings must provide some sort of new functionality. Some degree of functionality overlap with previously defined transfer encodings is acceptable, but any new transfer encoding must also offer something no other transfer encoding provides.

### 4.2. Transfer Encoding Definition Procedure

Definition of a new transfer encoding starts with the construction of a draft of a standards-track RFC. The RFC must define the transfer encoding precisely and completely, and must also provide substantial justification for defining and standardizing a new transfer encoding. This specification must then be presented to the IESG for consideration. The IESG can

(1) reject the specification outright as being inappropriate for standardization,

(2) approve the formation of an IETF working group to work on the specification in accordance with IETF procedures, or,

(3) accept the specification as-is and put it directly on the standards track.

Transfer encoding specifications on the standards track follow normal IETF rules for standards track documents. A transfer encoding is considered to be defined and available for use once it is on the standards track.

### 4.3. IANA Procedures for Transfer Encoding Registration

There is no need for a special procedure for registering Transfer Encodings with the IANA. All legitimate transfer encoding registrations must appear as a standards-track RFC, so it is the IESG's responsibility to notify the IANA when a new transfer encoding has been approved.

### 4.4. Location of Registered Transfer Encodings List

Transfer encoding registrations will be posted in the anonymous FTP directory "ftp://ftp.isi.edu/in-notes/iana/assignments/transfer-encodings/" and all registered transfer encodings will be listed in the periodically issued "Assigned Numbers" RFC [currently RFC-1700].







## 5. Authors' Addresses

For more information, the authors of this document are best contacted via Internet mail:

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790
USA

Phone: +1 818 919 3600
Fax: +1 818 919 3614
EMail: ned@innosoft.com

John Klensin
MCI
2100 Reston Parkway
Reston, VA 22091

Phone: +1 703 715-7361
Fax: +1 703 715-7436
EMail: klensin@mci.net

Jon Postel
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
USA

Phone: +1 310 822 1511
Fax: +1 310 823 6714
EMail: Postel@ISI.EDU

Appendix A -- Grandfathered Media Types

A number of media types, registered prior to 1996, would, if registered under the guidelines in this document, be placed into either the vendor or personal trees. Reregistration of those types to reflect the appropriate trees is encouraged, but not required. Ownership and change control principles outlined in this document apply to those types as if they had been registered in the trees described above.

Network Working Group N. Freed
Request for Comments: 2049 Innosoft
Obsoletes: 1521, 1522, 1590 N. Borenstein
Category: Standards Track First Virtual
November 1996

# Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples

## Status of this Memo



## Abstract

STD 11, RFC 822, defines a message representation protocol specifying considerable detail about US-ASCII message headers, and leaves the message content, or message body, as flat US-ASCII text. This set of documents, collectively called the Multipurpose Internet Mail Extensions, or MIME, redefines the format of messages to allow for

(1) textual message bodies in character sets other than US-ASCII,

(2) an extensible set of different formats for non-textual message bodies,

(3) multi-part message bodies, and

(4) textual header information in character sets other than US-ASCII.

These documents are based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said so little about message bodies, these documents are largely orthogonal to (rather than a revision of) RFC 822.

The initial document in this set, RFC 2045, specifies the various headers used to describe the structure of MIME messages. The second document defines the general structure of the MIME media typing system and defines an initial set of media types. The third document, RFC 2047, describes extensions to RFC 822 to allow non-US-ASCII text data in Internet mail header fields. The fourth document, RFC 2048, specifies various IANA registration procedures for MIME-related facilities. This fifth and final document describes MIME conformance criteria as well as providing some illustrative examples of MIME message formats, acknowledgements, and the bibliography.

These documents are revisions of RFCs 1521, 1522, and 1590, which themselves were revisions of RFCs 1341 and 1342. Appendix B of this document describes differences and changes from previous versions.





## Table of Contents



## 1. Introduction

The first and second documents in this set define MIME header fields and the initial set of MIME media types. The third document describes extensions to RFC822 formats to allow for character sets other than US-ASCII. This document describes what portions of MIME must be supported by a conformant MIME implementation. It also describes various pitfalls of contemporary messaging systems as well as the canonical encoding model MIME is based on.

## 2. MIME Conformance

The mechanisms described in these documents are open-ended. It is definitely not expected that all implementations will support all available media types, nor that they will all share the same extensions. In order to promote interoperability, however, it is useful to define the concept of "MIME-conformance" to define a certain level of implementation that allows the useful interworking of messages with content that differs from US-ASCII text. In this section, we specify the requirements for such conformance.

A mail user agent that is MIME-conformant MUST:

(1) Always generate a "MIME-Version: 1.0" header field in any message it creates.

(2) Recognize the Content-Transfer-Encoding header field and decode all received data encoded by either quoted-printable or base64 implementations. The identity transformations 7bit, 8bit, and binary must also be recognized.

Any non-7bit data that is sent without encoding must be properly labelled with a content-transfer-encoding of 8bit or binary, as appropriate. If the underlying transport does not support 8bit or binary (as SMTP [RFC-821] does not), the sender is required to both encode and label data using an appropriate Content-Transfer-Encoding such as quoted-printable or base64.

(3) Must treat any unrecognized Content-Transfer-Encoding as if it had a Content-Type of "application/octet-stream", regardless of whether or not the actual Content-Type is recognized.

(4) Recognize and interpret the Content-Type header field, and avoid showing users raw data with a Content-Type field other than text. Implementations must be able to send at least text/plain messages, with the character set specified with the charset parameter if it is not US-ASCII.

(5) Ignore any content type parameters whose names they do not recognize.

(6) Explicitly handle the following media type values, to at least the following extents:

Text:

-- Recognize and display "text" mail with the character set "US-ASCII."

-- Recognize other character sets at least to the extent of being able to inform the user about what character set the message uses.





-- Recognize the "ISO-8859-*" character sets to the extent of being able to display those characters that are common to ISO-8859-* and US-ASCII, namely all characters represented by octet values 1-127.

-- For unrecognized subtypes in a known character set, show or offer to show the user the "raw" version of the data after conversion of the content from canonical form to local form.

-- Treat material in an unknown character set as if it were "application/octet-stream".

Image, audio, and video:

-- At a minumum provide facilities to treat any unrecognized subtypes as if they were "application/octet-stream".

Application:

-- Offer the ability to remove either of the quoted-printable or base64 encodings defined in this document if they were used and put the resulting information in a user file.

Multipart:

-- Recognize the mixed subtype. Display all relevant information on the message level and the body part header level and then display or offer to display each of the body parts individually.

-- Recognize the "alternative" subtype, and avoid showing the user redundant parts of multipart/alternative mail.

-- Recognize the "multipart/digest" subtype, specifically using "message/rfc822" rather than "text/plain" as the default media type for body parts inside "multipart/digest" entities.

-- Treat any unrecognized subtypes as if they were "mixed".

Message:

-- Recognize and display at least the RFC822 message encapsulation (message/rfc822) in such a way as to preserve any recursive structure, that is, displaying or offering to display the encapsulated data in accordance with its media type.

-- Treat any unrecognized subtypes as if they were "application/octet-stream".

(7) Upon encountering any unrecognized Content-Type field, an implementation must treat it as if it had a media type of "application/octet-stream" with no parameter sub-arguments. How such data are handled is up to an implementation, but likely options for handling such unrecognized data include offering the user to write it into a file (decoded from its mail transport format) or offering the user to name a program to which the decoded data should be passed as input.

(8) Conformant user agents are required, if they provide non-standard support for non-MIME messages employing character sets other than US-ASCII, to do so on received messages only. Conforming user agents must not send non-MIME messages containing anything other than US-ASCII text.

In particular, the use of non-US-ASCII text in mail messages without a MIME-Version field is strongly discouraged as it impedes interoperability when sending messages between regions with different localization conventions. Conforming user agents MUST include proper MIME labelling when sending anything other than plain text in the US-ASCII character set.

In addition, non-MIME user agents should be upgraded if at all possible to include appropriate MIME header information in the messages they send even if nothing else in MIME is supported. This upgrade will have little, if any, effect on non-MIME recipients and will aid MIME in correctly displaying such messages. It also provides a smooth transition path to eventual adoption of other MIME capabilities.

(9) Conforming user agents must ensure that any string of non-white-space printable US-ASCII characters within a "*text" or "*ctext" that begins with "=?" and ends with "?=" be a valid encoded-word. ("begins" means: At the start of the field-body or immediately following linear-white-space; "ends" means: At the end of the field-body or immediately preceding linear-white-space.) In addition, any "word" within a "phrase" that begins with "=?" and ends with "?=" must be a valid encoded-word.

(10) Conforming user agents must be able to distinguish encoded-words from "text", "ctext", or "word"s, according to the rules in section 4, anytime they appear in appropriate places in message headers. It must support both the "B" and "Q" encodings for any character set which it supports. The program must be able to display the unencoded text if the character set is "US-ASCII". For the ISO-8859-* character sets, the mail reading program must at least be able to display the characters which are also in the US-ASCII set.

A user agent that meets the above conditions is said to be MIME-conformant. The meaning of this phrase is that it is assumed to be "safe" to send virtually any kind of properly-marked data to users of such mail systems, because such systems will at least be able to treat the data as undifferentiated binary, and will not simply splash it onto the screen of unsuspecting users.

There is another sense in which it is always "safe" to send data in a format that is MIME-conformant, which is that such data will not break or be broken by any known systems that are conformant with RFC 821 and RFC 822. User agents that are MIME-conformant have the additional guarantee that the user will not be shown data that were never intended to be viewed as text.

## 3. Guidelines for Sending Email Data

Internet email is not a perfect, homogeneous system. Mail may become corrupted at several stages in its travel to a final destination. Specifically, email sent throughout the Internet may travel across many networking technologies. Many networking and mail technologies do not support the full functionality possible in the SMTP transport environment. Mail traversing these systems is likely to be modified in order that it can be transported.

There exist many widely-deployed non-conformant MTAs in the Internet. These MTAs, speaking the SMTP protocol, alter messages on the fly to take advantage of the internal data structure of the hosts they are implemented on, or are just plain broken.

The following guidelines may be useful to anyone devising a data format (media type) that is supposed to survive the widest range of networking technologies and known broken MTAs unscathed. Note that anything encoded in the base64 encoding will satisfy these rules, but that some well-known mechanisms, notably the UNIX uuencode facility, will not. Note also that anything encoded in the Quoted-Printable encoding will survive most gateways intact, but possibly not some gateways to systems that use the EBCDIC character set.

(1) Under some circumstances the encoding used for data may change as part of normal gateway or user agent operation. In particular, conversion from base64 to quoted-printable and vice versa may be necessary. This may result in the confusion of CRLF sequences with line breaks in text bodies. As such, the persistence of CRLF as something other than a line break must not be relied on.

(2) Many systems may elect to represent and store text data using local newline conventions. Local newline conventions may not match the RFC822 CRLF convention -- systems are known that use plain CR, plain LF, CRLF, or counted records. The result is that isolated CR and LF characters are not well tolerated in general; they may be lost or converted to delimiters on some systems, and hence must not be relied on.

(3) The transmission of NULs (US-ASCII value 0) is problematic in Internet mail. (This is largely the result of NULs being used as a termination character by many of the standard runtime library routines in the C programming language.) The practice of using NULs as termination characters is so entrenched now that messages should not rely on them being preserved.

(4) TAB (HT) characters may be misinterpreted or may be automatically converted to variable numbers of spaces. This is unavoidable in some environments, notably those not based on the US-ASCII character set. Such conversion is STRONGLY DISCOURAGED, but it may occur, and mail formats must not rely on the persistence of TAB (HT) characters.

(5) Lines longer than 76 characters may be wrapped or truncated in some environments. Line wrapping or line truncation imposed by mail transports is STRONGLY DISCOURAGED, but unavoidable in some cases. Applications which require long lines must somehow







differentiate between soft and hard line breaks. (A simple way to do this is to use the quoted-printable encoding.)

(6) Trailing "white space" characters (SPACE, TAB (HT)) on a line may be discarded by some transport agents, while other transport agents may pad lines with these characters so that all lines in a mail file are of equal length. The persistence of trailing white space, therefore, must not be relied on.

(7) Many mail domains use variations on the US-ASCII character set, or use character sets such as EBCDIC which contain most but not all of the US-ASCII characters. The correct translation of characters not in the "invariant" set cannot be depended on across character converting gateways. For example, this situation is a problem when sending uuencoded information across BITNET, an EBCDIC system. Similar problems can occur without crossing a gateway, since many Internet hosts use character sets other than US-ASCII internally. The definition of Printable Strings in X.400 adds further restrictions in certain special cases. In particular, the only characters that are known to be consistent across all gateways are the 73 characters that correspond to the upper and lower case letters A-Z and a-z, the 10 digits 0-9, and the following eleven special characters:

"'" (US-ASCII decimal value 39)
"(" (US-ASCII decimal value 40)
")" (US-ASCII decimal value 41)
"+" (US-ASCII decimal value 43)
"," (US-ASCII decimal value 44)
"-" (US-ASCII decimal value 45)
"." (US-ASCII decimal value 46)
"/" (US-ASCII decimal value 47)
":" (US-ASCII decimal value 58)
"=" (US-ASCII decimal value 61)
"?" (US-ASCII decimal value 63)

A maximally portable mail representation will confine itself to relatively short lines of text in which the only meaningful characters are taken from this set of 73 characters. The base64 encoding follows this rule.

(8) Some mail transport agents will corrupt data that includes certain literal strings. In particular, a

period (".") alone on a line is known to be corrupted by some (incorrect) SMTP implementations, and a line that starts with the five characters "From " (the fifth character is a SPACE) are commonly corrupted as well. A careful composition agent can prevent these corruptions by encoding the data (e.g., in the quoted-printable encoding using "=46rom " in place of "From " at the start of a line, and "=2E" in place of "." alone on a line).

Please note that the above list is NOT a list of recommended practices for MTAs. RFC 821 MTAs are prohibited from altering the character of white space or wrapping long lines. These BAD and invalid practices are known to occur on established networks, and implementations should be robust in dealing with the bad effects they can cause.

## 4. Canonical Encoding Model

There was some confusion, in earlier versions of these documents, regarding the model for when email data was to be converted to canonical form and encoded, and in particular how this process would affect the treatment of CRLFs, given that the representation of newlines varies greatly from system to system. For this reason, a canonical model for encoding is presented below.

The process of composing a MIME entity can be modeled as being done in a number of steps. Note that these steps are roughly similar to those steps used in PEM [RFC-1421] and are performed for each "innermost level" body:

(1) Creation of local form.

The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end of line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.





(2) Conversion to canonical form.

The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semantics of the media type, which may have strong implications for any character set conversion, e.g. with regard to syntactically meaningful characters in a text subtype other than "plain".

For example, in the case of text/plain data, the text must be converted to a supported character set and lines must be delimited with CRLF delimiters in accordance with RFC 822. Note that the restriction on line lengths implied by RFC 822 is eliminated if the next step employs either quoted-printable or base64 encoding.

(3) Apply transfer encoding.

A Content-Transfer-Encoding appropriate for this body is applied. Note that there is no fixed relationship between the media type and the transfer encoding. In particular, it may be appropriate to base the choice of base64 or quoted-printable on character frequency counts which are specific to a given instance of a body.

(4) Insertion into entity.

The encoded body is inserted into a MIME entity with appropriate headers. The entity is then inserted into the body of a higher-level entity (message or multipart) as needed.

Conversion from entity form to local form is accomplished by reversing these steps. Note that reversal of these steps may produce differing results since there is no guarantee that the original and final local forms are the same.

It is vital to note that these steps are only a model; they are specifically NOT a blueprint for how an actual system would be built. In particular, the model fails to account for two common designs:

(1) In many cases the conversion to a canonical form prior to encoding will be subsumed into the encoder itself, which understands local formats directly. For example, the local newline convention for text bodies might be carried through to the encoder itself along with knowledge of what that format is.

(2) The output of the encoders may have to pass through one or more additional steps prior to being transmitted as a message. As such, the output of the encoder may not be conformant with the formats specified by RFC 822. In particular, once again it may be appropriate for the converter's output to be expressed using local newline conventions rather than using the standard RFC 822 CRLF delimiters.

Other implementation variations are conceivable as well. The vital aspect of this discussion is that, in spite of any optimizations, collapsings of required steps, or insertion of additional processing, the resulting messages must be consistent with those produced by the model described here. For example, a message with the following header fields:

```
Content-type: text/foo; charset=bar
Content-Transfer-Encoding: base64
```

must be first represented in the text/foo form, then (if necessary) represented in the "bar" character set, and finally transformed via the base64 algorithm into a mail-safe form.

NOTE: Some confusion has been caused by systems that represent messages in a format which uses local newline conventions which differ from the RFC822 CRLF convention. It is important to note that these formats are not canonical RFC822/MIME. These formats are instead *encodings* of RFC822, where CRLF sequences in the canonical representation of the message are encoded as the local newline convention. Note that formats which encode CRLF sequences as, for example, LF are not capable of representing MIME messages containing binary data which contains LF octets not part of CRLF line separation sequences.







## 5. Summary

This document defines what is meant by MIME Conformance. It also details various problems known to exist in the Internet email system and how to use MIME to overcome them. Finally, it describes MIME's canonical encoding model.

## 6. Security Considerations

Security issues are discussed in the second document in this set, RFC 2046.

## 7. Authors' Addresses

For more information, the authors of this document are best contacted via Internet mail:

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790
USA

Phone: +1 818 919 3600
Fax: +1 818 919 3614
EMail: ned@innosoft.com

Nathaniel S. Borenstein
First Virtual Holdings
25 Washington Avenue
Morristown, NJ 07960
USA

Phone: +1 201 540 8967
Fax: +1 201 993 3032
EMail: nsb@nsb.fv.com

MIME is a result of the work of the Internet Engineering Task Force Working Group on RFC 822 Extensions. The chairman of that group, Greg Vaudreuil, may be reached at:

Gregory M. Vaudreuil
Octel Network Services
17080 Dallas Parkway
Dallas, TX 75248-1905
USA

EMail: Greg.Vaudreuil@Octel.Com

## 8. Acknowledgements

This document is the result of the collective effort of a large number of people, at several IETF meetings, on the IETF-SMTP and IETF-822 mailing lists, and elsewhere. Although any enumeration seems doomed to suffer from egregious omissions, the following are among the many contributors to this effort:

| | |
|---|---|
| Harald Tveit Alvestrand | Marc Andreessen |
| Randall Atkinson | Bob Braden |
| Philippe Brandon | Brian Capouch |
| Kevin Carosso | Uhhyung Choi |
| Peter Clitherow | Dave Collier-Brown |
| Cristian Constantinof | John Coonrod |
| Mark Crispin | Dave Crocker |
| Stephen Crocker | Terry Crowley |
| Walt Daniels | Jim Davis |
| Frank Dawson | Axel Deininger |
| Hitoshi Doi | Kevin Donnelly |
| Steve Dorner | Keith Edwards |
| Chris Eich | Dana S. Emery |
| Johnny Eriksson | Craig Everhart |
| Patrik Faltstrom | Erik E. Fair |
| Roger Fajman | Alain Fontaine |
| Martin Forssen | James M. Galvin |
| Stephen Gildea | Philip Gladstone |
| Thomas Gordon | Keld Simonsen |
| Terry Gray | Phill Gross |
| James Hamilton | David Herron |
| Mark Horton | Bruce Howard |
| Bill Janssen | Olle Jarnefors |
| Risto Kankkunen | Phil Karn |
| Alan Katz | Tim Kehres |
| Neil Katin | Steve Kille |
| Kyuho Kim | Anders Klemets |
| John Klensin | Valdis Kletniek |
| Jim Knowles | Stev Knowles |
| Bob Kummerfeld | Pekka Kytolaakso |
| Stellan Lagerstrom | Vincent Lau |
| Timo Lehtinen | Donald Lindsay |
| Warner Losh | Carlyn Lowery |
| Laurence Lundblade | Charles Lynn |
| John R. MacMillan | Larry Masinter |
| Rick McGowan | Michael J. McInerny |
| Leo Mclaughlin | Goli Montaser-Kohsari |
| Tom Moore | John Gardiner Myers |
| Erik Naggum | Mark Needleman |
| Chris Newman | John Noerenberg |
| Mats Ohrman | Julian Onions |
| Michael Patton | David J. Pepper |
| Erik van der Poel | Blake C. Ramsdell |
| Christer Romson | Luc Rooijakkers |
| Marshall T. Rose | Jonathan Rosenberg |
| Guido van Rossum | Jan Rynning |
| Harri Salminen | Michael Sanderson |
| Yutaka Sato | Markku Savela |
| Richard Alan Schafer | Masahiro Sekiguchi |
| Mark Sherman | Bob Smart |
| Peter Speck | Henry Spencer |
| Einar Stefferud | Michael Stein |
| Klaus Steinberger | Peter Svanberg |
| James Thompson | Steve Uhler |
| Stuart Vance | Peter Vanderbilt |
| Greg Vaudreuil | Ed Vielmetti |
| Larry W. Virden | Ryan Waldron |
| Rhys Weatherly | Jay Weber |
| Dave Wecker | Wally Wedel |
| Sven-Ove Westberg | Brian Wideen |
| John Wobus | Glenn Wright |
| Rayan Zachariassen | David Zimmerman |

The authors apologize for any omissions from this list, which are certainly unintentional.

## Appendix A -- A Complex Multipart Example

What follows is the outline of a complex multipart message. This message contains five parts that are to be displayed serially: two introductory plain text objects, an embedded multipart message, a text/enriched object, and a closing encapsulated text message in a non-ASCII character set. The embedded multipart message itself contains two objects to be displayed in parallel, a picture and an audio fragment.

```
MIME-Version: 1.0
From: Nathaniel Borenstein <nsb@nsb.fv.com>
To: Ned Freed <ned@innosoft.com>
Date: Fri, 07 Oct 1994 16:15:05 -0700 (PDT)
Subject: A multipart example
Content-Type: multipart/mixed;
              boundary=unique-boundary-1

This is the preamble area of a multipart message.
Mail readers that understand multipart format
should ignore this preamble.

If you are reading this text, you might want to
consider changing to a mail reader that understands
how to properly display multipart messages.

--unique-boundary-1

  ... Some text appears here ...

[Note that the blank between the boundary and the start
 of the text in this part means no header fields were
 given and this is text in the US-ASCII character set.
 It could have been done with explicit typing as in the
 next part.]

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part, but
illustrates explicit versus implicit typing of body
parts.

--unique-boundary-1
Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

  ... base64-encoded 8000 Hz single-channel
      mu-law-format audio data goes here ...

--unique-boundary-2
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

  ... base64-encoded image data goes here ...

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>enriched.</italic></bold>
<smaller>as defined in RFC 1896</smaller>

Isn't it
<bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

  ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

## Appendix B -- Changes from RFC 1521, 1522, and 1590

These documents are a revision of RFC 1521, 1522, and 1590. For the convenience of those familiar with the earlier documents, the changes from those documents are summarized in this appendix. For further history, note that Appendix H in RFC 1521 specified how that document differed from its predecessor, RFC 1341.

(1) This document has been completely reformatted and split into multiple documents. This was done to improve the quality of the plain text version of this document, which is required to be the reference copy.

(2) BNF describing the overall structure of MIME object headers has been added. This is a documentation change only -- the underlying syntax has not changed in any way.

(3) The specific BNF for the seven media types in MIME has been removed. This BNF was incorrect, incomplete, amd inconsistent with the type-indendependent BNF. And since the type-independent BNF already fully specifies the syntax of the various MIME headers, the type-specific BNF was, in the final analysis, completely unnecessary and caused more problems than it solved.

(4) The more specific "US-ASCII" character set name has replaced the use of the informal term ASCII in many parts of these documents.

(5) The informal concept of a primary subtype has been removed.

(6) The term "object" was being used inconsistently. The definition of this term has been clarified, along with the related terms "body", "body part", and "entity", and usage has been corrected where appropriate.

(7) The BNF for the multipart media type has been rearranged to make it clear that the CRLF preceeding the boundary marker is actually part of the marker itself rather than the preceeding body part.

(8) The prose and BNF describing the multipart media type have been changed to make it clear that the body parts within a multipart object MUST NOT contain any lines beginning with the boundary parameter string.

(9) In the rules on reassembling "message/partial" MIME entities, "Subject" is added to the list of headers to take from the inner message, and the example is modified to clarify this point.

(10) "Message/partial" fragmenters are restricted to splitting MIME objects only at line boundaries.

(11) In the discussion of the application/postscript type, an additional paragraph has been added warning about possible interoperability problems caused by embedding of binary data inside a PostScript MIME entity.







(12) Added a clarifying note to the basic syntax rules for the Content-Type header field to make it clear that the following two forms:

```
Content-type: text/plain; charset=us-ascii (comment)

Content-type: text/plain; charset="us-ascii"
```

are completely equivalent.

(13) The following sentence has been removed from the discussion of the MIME-Version header: "However, conformant software is encouraged to check the version number and at least warn the user if an unrecognized MIME-version is encountered."

(14) A typo was fixed that said "application/external-body" instead of "message/external-body".

(15) The definition of a character set has been reorganized to make the requirements clearer.

(16) The definition of the "image/gif" media type has been moved to a separate document. This change was made because of potential conflicts with IETF rules governing the standardization of patented technology.

(17) The definitions of "7bit" and "8bit" have been tightened so that use of bare CR, LF can only be used as end-of-line sequences. The document also no longer requires that NUL characters be preserved, which brings MIME into alignment with real-world implementations.

(18) The definition of canonical text in MIME has been tightened so that line breaks must be represented by a CRLF sequence. CR and LF characters are not allowed outside of this usage. The definition of quoted-printable encoding has been altered accordingly.

(19) The definition of the quoted-printable encoding now includes a number of suggestions for how quoted-printable encoders might best handle improperly encoded material.

(20) Prose was added to clarify the use of the "7bit", "8bit", and "binary" transfer-encodings on multipart or message entities encapsulating "8bit" or "binary" data.

(21) In the section on MIME Conformance, "multipart/digest" support was added to the list of requirements for minimal MIME conformance. Also, the requirement for "message/rfc822" support were strengthened to clarify the importance of recognizing recursive structure.

(22) The various restrictions on subtypes of "message" are now specified entirely on a subtype by subtype basis.

(23) The definition of "message/rfc822" was changed to indicate that at least one of the "From", "Subject", or "Date" headers must be present.

(24) The required handling of unrecognized subtypes as "application/octet-stream" has been made more explicit in both the type definitions sections and the conformance guidelines.

(25) Examples using text/richtext were changed to text/enriched.

(26) The BNF definition of subtype has been changed to make it clear that either an IANA registered subtype or a nonstandard "X-" subtype must be used in a Content-Type header field.

(27) MIME media types that are simply registered for use and those that are standardized by the IETF are now distinguished in the MIME BNF.

(28) All of the various MIME registration procedures have been extensively revised. IANA registration procedures for character sets have been moved to a separate document that is no included in this set of documents.

(29) The use of escape and shift mechanisms in the US-ASCII and ISO-8859-X character sets these documents define have been clarified: Such mechanisms should never be used in conjunction with these character sets and their effect if they are used is undefined.

(30) The definition of the AFS access-type for message/external-body has been removed.

(31) The handling of the combination of multipart/alternative and message/external-body is now specifically addressed.







(32) Security issues specific to message/external-body are now discussed in some detail.

Appendix C -- References

[ATK]
Borenstein, Nathaniel S., Multimedia Applications Development with the Andrew Toolkit, Prentice-Hall, 1990.

[ISO-2022]
International Standard -- Information Processing -- Character Code Structure and Extension Techniques, ISO/IEC 2022:1994, 4th ed.

[ISO-8859]
International Standard -- Information Processing -- 8-bit Single-Byte Coded Graphic Character Sets
- Part 1: Latin Alphabet No. 1, ISO 8859-1:1987, 1st ed.
- Part 2: Latin Alphabet No. 2, ISO 8859-2:1987, 1st ed.
- Part 3: Latin Alphabet No. 3, ISO 8859-3:1988, 1st ed.
- Part 4: Latin Alphabet No. 4, ISO 8859-4:1988, 1st ed.
- Part 5: Latin/Cyrillic Alphabet, ISO 8859-5:1988, 1st ed.
- Part 6: Latin/Arabic Alphabet, ISO 8859-6:1987, 1st ed.
- Part 7: Latin/Greek Alphabet, ISO 8859-7:1987, 1st ed.
- Part 8: Latin/Hebrew Alphabet, ISO 8859-8:1988, 1st ed.
- Part 9: Latin Alphabet No. 5, ISO/IEC 8859-9:1989, 1st ed.
International Standard -- Information Technology -- 8-bit Single-Byte Coded Graphic Character Sets
- Part 10: Latin Alphabet No. 6, ISO/IEC 8859-10:1992, 1st ed.

[ISO-646]
International Standard -- Information Technology -- ISO 7-bit Coded Character Set for Information Interchange, ISO 646:1991, 3rd ed..

[JPEG]
JPEG Draft Standard ISO 10918-1 CD.

[MPEG]
Video Coding Draft Standard ISO 11172 CD, ISO IEC/JTC1/SC2/WG11 (Motion Picture Experts Group), May, 1991.

[PCM]
CCITT, Fascicle III.4 - Recommendation G.711, "Pulse Code Modulation (PCM) of Voice Frequencies", Geneva, 1972.

[POSTSCRIPT]
Adobe Systems, Inc., PostScript Language Reference Manual, Addison-Wesley, 1985.

[POSTSCRIPT2]
Adobe Systems, Inc., PostScript Language Reference Manual, Addison-Wesley, Second Ed., 1990.

[RFC-783]
Sollins, K.R., "TFTP Protocol (revision 2)", RFC-783, MIT, June 1981.

[RFC-821]
Postel, J.B., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[RFC-822]
Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[RFC-934]
Rose, M. and E. Stefferud, "Proposed Standard for Message Encapsulation", RFC 934, Delaware and NMA, January 1985.

[RFC-959]
Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, USC/Information Sciences Institute, October 1985.

[RFC-1049]
Sirbu, M., "Content-Type Header Field for Internet Messages", RFC 1049, CMU, March 1988.

[RFC-1154]
Robinson, D., and R. Ullmann, "Encoding Header Field for Internet Messages", RFC 1154, Prime Computer, Inc., April 1990.

[RFC-1341]
Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1341, Bellcore, Innosoft, June 1992.





[RFC-1342]
Moore, K., "Representation of Non-Ascii Text in Internet Message Headers", RFC 1342, University of Tennessee, June 1992.

[RFC-1344]
Borenstein, N., "Implications of MIME for Internet Mail Gateways", RFC 1344, Bellcore, June 1992.

[RFC-1345]
Simonsen, K., "Character Mnemonics & Character Sets", RFC 1345, Rationel Almen Planlaegning, June 1992.

[RFC-1421]
Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encryption and Authentication Procedures", RFC 1421, IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1422]
Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management", RFC 1422, IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1423]
Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers", IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1424]
Kaliski, B., "Privacy Enhancement for Internet Electronic Mail: Part IV -- Key Certification and Related Services", IAB IRTF PSRG, IETF PEM WG, February 1993.

[RFC-1521]
Borenstein, N., and Freed, N., "MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September, 1993.

[RFC-1522]
Moore, K., "Representation of Non-ASCII Text in Internet Message Headers", RFC 1522, University of Tennessee, September 1993.

[RFC-1524]
Borenstein, N., "A User Agent Configuration Mechanism for Multimedia Mail Format Information", RFC 1524, Bellcore, September 1993.

[RFC-1543]
Postel, J., "Instructions to RFC Authors", RFC 1543, USC/Information Sciences Institute, October 1993.

[RFC-1556]
Nussbacher, H., "Handling of Bi-directional Texts in MIME", RFC 1556, Israeli Inter-University Computer Center, December 1993.

[RFC-1590]
Postel, J., "Media Type Registration Procedure", RFC 1590, USC/Information Sciences Institute, March 1994.

[RFC-1602]
Internet Architecture Board, Internet Engineering Steering Group, Huitema, C., Gross, P., "The Internet Standards Process -- Revision 2", March 1994.

[RFC-1652]
Klensin, J., (WG Chair), Freed, N., (Editor), Rose, M., Stefferud, E., and Crocker, D., "SMTP Service Extension for 8bit-MIME transport", RFC 1652, United Nations University, Innosoft, Dover Beach Consulting, Inc., Network Management Associates, Inc., The Branch Office, March 1994.

[RFC-1700]
Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[RFC-1741]
Faltstrom, P., Crocker, D., and Fair, E., "MIME Content Type for BinHex Encoded Files", December 1994.

[RFC-1896]
Resnick, P., and A. Walker, "The text/enriched MIME Content-type", RFC 1896, February, 1996.







[RFC-2045]
Freed, N., and and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, Innosoft, First Virtual Holdings, November 1996.

[RFC-2046]
Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, Innosoft, First Virtual Holdings, November 1996.

[RFC-2047]
Moore, K., "Multipurpose Internet Mail Extensions (MIME) Part Three: Representation of Non-ASCII Text in Internet Message Headers", RFC 2047, University of Tennessee, November 1996.

[RFC-2048]
Freed, N., Klensin, J., and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: MIME Registration Procedures", RFC 2048, Innosoft, MCI, ISI, November 1996.

[RFC-2049]
Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049 (this document), Innosoft, First Virtual Holdings, November 1996.

[US-ASCII]
Coded Character Set -- 7-Bit American Standard Code for Information Interchange, ANSI X3.4-1986.

[X400]
Schicker, Pietro, "Message Handling Systems, X.400", Message Handling Systems and Distributed Applications, E. Stefferud, O-j. Jacobsen, and P. Schicker, eds., North-Holland, 1989, pp. 3-41.

Network Working Group
Request for Comments: 1891
Category: Standards Track

K. Moore
University of Tennessee
January 1996

# SMTP Service Extension for Delivery Status Notifications

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## 1. Abstract

This memo defines an extension to the SMTP service, which allows an SMTP client to specify (a) that delivery status notifications (DSNs) should be generated under certain conditions, (b) whether such notifications should return the contents of the message, and (c) additional information, to be returned with a DSN, that allows the sender to identify both the recipient(s) for which the DSN was issued, and the transaction in which the original message was sent.

Any questions, comments, and reports of defects or ambiguities in this specification may be sent to the mailing list for the NOTARY working group of the IETF, using the address <notifications@cs.utk.edu>. Requests to subscribe to the mailing list should be addressed to <notifications-request@cs.utk.edu>. Implementors of this specification are encouraged to subscribe to the mailing list, so that they will quickly be informed of any problems which might hinder interoperability.

NOTE: This document is a Proposed Standard. If and when this protocol is submitted for Draft Standard status, any normative text (phrases containing SHOULD, SHOULD NOT, MUST, MUST NOT, or MAY) in this document will be re-evaluated in light of implementation experience, and are thus subject to change.

## 2. Introduction

The SMTP protocol [1] requires that an SMTP server provide notification of delivery failure, if it determines that a message cannot be delivered to one or more recipients. Traditionally, such notification consists of an ordinary Internet mail message (format defined by [2]), sent to the envelope sender address (the argument of





the SMTP MAIL command), containing an explanation of the error and at least the headers of the failed message.

Experience with large mail distribution lists [3] indicates that such messages are often insufficient to diagnose problems, or even to determine at which host or for which recipients a problem occurred. In addition, the lack of a standardized format for delivery notifications in Internet mail makes it difficult to exchange such notifications with other message handling systems.

Such experience has demonstrated a need for a delivery status notification service for Internet electronic mail, which:

(a) is reliable, in the sense that any DSN request will either be honored at the time of final delivery, or result in a response that indicates that the request cannot be honored,

(b) when both success and failure notifications are requested, provides an unambiguous and nonconflicting indication of whether delivery of a message to a recipient succeeded or failed,

(c) is stable, in that a failed attempt to deliver a DSN should never result in the transmission of another DSN over the network,

(d) preserves sufficient information to allow the sender to identify both the mail transaction and the recipient address which caused the notification, even when mail is forwarded or gatewayed to foreign environments, and

(e) interfaces acceptably with non-SMTP and non-822-based mail systems, both so that notifications returned from foreign mail systems may be useful to Internet users, and so that the notification requests from foreign environments may be honored. Among the requirements implied by this goal are the ability to request non-return-of-content, and the ability to specify whether positive delivery notifications, negative delivery notifications, both, or neither, should be issued.

In an attempt to provide such a service, this memo uses the mechanism defined in [4] to define an extension to the SMTP protocol. Using this mechanism, an SMTP client may request that an SMTP server issue or not issue a delivery status notification (DSN) under certain conditions. The format of a DSN is defined in [5].

## 3. Framework for the Delivery Status Notification Extension

The following service extension is therefore defined:

(1) The name of the SMTP service extension is "Delivery Status Notification";

(2) the EHLO keyword value associated with this extension is "DSN", the meaning of which is defined in section 4 of this memo;

(3) no parameters are allowed with this EHLO keyword value;

(4) two optional parameters are added to the RCPT command, and two optional parameters are added to the MAIL command:

An optional parameter for the RCPT command, using the esmtp-keyword "NOTIFY", (to specify the conditions under which a delivery status notification should be generated), is defined in section 5.1,

An optional parameter for the RCPT command, using the esmtp-keyword "ORCPT", (used to convey the "original" (sender-specified) recipient address), is defined in section 5.2, and

An optional parameter for the MAIL command, using the esmtp-keyword "RET", (to request that DSNs containing an indication of delivery failure either return the entire contents of a message or only the message headers), is defined in section 5.3,

An optional parameter for the MAIL command, using the esmtp-keyword "ENVID", (used to propagate an identifier for this message transmission envelope, which is also known to the sender and will, if present, be returned in any DSNs issued for this transmission), is defined in section 5.4;

(5) no additional SMTP verbs are defined by this extension.

The remainder of this memo specifies how support for the extension effects the behavior of a message transfer agent.

## 4. The Delivery Status Notification service extension

An SMTP client wishing to request a DSN for a message may issue the EHLO command to start an SMTP session, to determine if the server supports any of several service extensions. If the server responds with code 250 to the EHLO command, and the response includes the EHLO





keyword DSN, then the Delivery Status Notification extension (as described in this memo) is supported.

Ordinarily, when an SMTP server returns a positive (2xx) reply code in response to a RCPT command, it agrees to accept responsibility for either delivering the message to the named recipient, or sending a notification to the sender of the message indicating that delivery has failed. However, an extended SMTP ("ESMTP") server which implements this service extension will accept an optional NOTIFY parameter with the RCPT command. If present, the NOTIFY parameter alters the conditions for generation of delivery status notifications from the default (issue notifications only on failure) specified in [1]. The ESMTP client may also request (via the RET parameter) whether the entire contents of the original message should be returned (as opposed to just the headers of that message), along with the DSN.

In general, an ESMTP server which implements this service extension will propagate delivery status notification requests when relaying mail to other SMTP-based MTAs which also support this extension, and make a "best effort" to ensure that such requests are honored when messages are passed into other environments.

In order that any delivery status notifications thus generated will be meaningful to the sender, any ESMTP server which supports this extension will attempt to propagate the following information to any other MTAs that are used to relay the message, for use in generating DSNs:

(a) for each recipient, a copy of the original recipient address, as used by the sender of the message.

This address need not be the same as the mailbox specified in the RCPT command. For example, if a message was originally addressed to A@B.C and later forwarded to A@D.E, after such forwarding has taken place, the RCPT command will specify a mailbox of A@D.E. However, the original recipient address remains A@B.C.

Also, if the message originated from an environment which does not use Internet-style user@domain addresses, and was gatewayed into SMTP, the original recipient address will preserve the original form of the recipient address.

(b) for the entire SMTP transaction, an envelope identification string, which may be used by the sender to associate any delivery status notifications with the transaction used to send the original message.

## 5. Additional parameters for RCPT and MAIL commands

The extended RCPT and MAIL commands are issued by a client when it wishes to request a DSN from the server, under certain conditions, for a particular recipient. The extended RCPT and MAIL commands are identical to the RCPT and MAIL commands defined in [1], except that one or more of the following parameters appear after the sender or recipient address, respectively. The general syntax for extended SMTP commands is defined in [4].

NOTE: Although RFC 822 ABNF is used to describe the syntax of these parameters, they are not, in the language of that document, "structured field bodies". Therefore, while parentheses MAY appear within an emstp-value, they are not recognized as comment delimiters.

The syntax for "esmtp-value" in [4] does not allow SP, "=", control characters, or characters outside the traditional ASCII range of 1-127 decimal to be transmitted in an esmtp-value. Because the ENVID and ORCPT parameters may need to convey values outside this range, the esmtp-values for these parameters are encoded as "xtext". "xtext" is formally defined as follows:

```
  xtext = *( xchar / hexchar )

  xchar = any ASCII CHAR between "!" (33) and "~" (126) inclusive,
       except for "+" and "=".

; "hexchar"s are intended to encode octets that cannot appear
; as ASCII characters within an esmtp-value.

  hexchar = ASCII "+" immediately followed by two upper case
       hexadecimal digits
```

When encoding an octet sequence as xtext:

+ Any ASCII CHAR between "!" and "~" inclusive, except for "+" and "=", MAY be encoded as itself. (A CHAR in this range MAY instead be encoded as a "hexchar", at the implementor's discretion.)

+ ASCII CHARs that fall outside the range above must be encoded as "hexchar".

### 5.1 The NOTIFY parameter of the ESMTP RCPT command

A RCPT command issued by a client may contain the optional esmtp-keyword "NOTIFY", to specify the conditions under which the SMTP server should generate DSNs for that recipient. If the NOTIFY esmtp-keyword is used, it MUST have an associated esmtp-value, formatted according to the following rules, using the ABNF of RFC 822:

```
  notify-esmtp-value = "NEVER" / 1#notify-list-element

  notify-list-element = "SUCCESS" / "FAILURE" / "DELAY"
```

Notes:

a. Multiple notify-list-elements, separated by commas, MAY appear in a NOTIFY parameter; however, the NEVER keyword MUST appear by itself.

b. Any of the keywords NEVER, SUCCESS, FAILURE, or DELAY may be spelled in any combination of upper and lower case letters.

The meaning of the NOTIFY parameter values is generally as follows:

+ A NOTIFY parameter value of "NEVER" requests that a DSN not be returned to the sender under any conditions.

+ A NOTIFY parameter value containing the "SUCCESS" or "FAILURE" keywords requests that a DSN be issued on successful delivery or delivery failure, respectively.

+ A NOTIFY parameter value containing the keyword "DELAY" indicates the sender's willingness to receive "delayed" DSNs. Delayed DSNs may be issued if delivery of a message has been delayed for an unusual amount of time (as determined by the MTA at which the message is delayed), but the final delivery status (whether successful or failure) cannot be determined. The absence of the DELAY keyword in a NOTIFY parameter requests that a "delayed" DSN NOT be issued under any conditions.

The actual rules governing interpretation of the NOTIFY parameter are given in section 6.

For compatibility with SMTP clients that do not use the NOTIFY facility, the absence of a NOTIFY parameter in a RCPT command may be interpreted as either NOTIFY=FAILURE or NOTIFY=FAILURE,DELAY.

### 5.2 The ORCPT parameter to the ESMTP RCPT command

The ORCPT esmtp-keyword of the RCPT command is used to specify an "original" recipient address that corresponds to the actual recipient to which the message is to be delivered. If the ORCPT esmtp-keyword is used, it MUST have an associated esmtp-value, which consists of the original recipient address, encoded according to the rules below. The ABNF for the ORCPT parameter is:

```
orcpt-parameter = "ORCPT=" original-recipient-address

original-recipient-address = addr-type ";" xtext

addr-type = atom
```

The "addr-type" portion MUST be an IANA-registered electronic mail address-type (as defined in [5]), while the "xtext" portion contains an encoded representation of the original recipient address using the rules in section 5 of this document. The entire ORCPT parameter MAY be up to 500 characters in length.

When initially submitting a message via SMTP, if the ORCPT parameter is used, it MUST contain the same address as the RCPT TO address (unlike the RCPT TO address, the ORCPT parameter will be encoded as xtext). Likewise, when a mailing list submits a message via SMTP to be distributed to the list subscribers, if ORCPT is used, the ORCPT parameter MUST match the new RCPT TO address of each recipient, not the address specified by the original sender of the message.)

The "addr-type" portion of the original-recipient-address is used to indicate the "type" of the address which appears in the ORCPT parameter value. However, the address associated with the ORCPT keyword is NOT constrained to conform to the syntax rules for that "addr-type".

Ideally, the "xtext" portion of the original-recipient-address should contain, in encoded form, the same sequence of characters that the sender used to specify the recipient. However, for a message gatewayed from an environment (such as X.400) in which a recipient address is not a simple string of printable characters, the representation of recipient address must be defined by a specification for gatewaying between DSNs and that environment.

### 5.3 The RET parameter of the ESMTP MAIL command

The RET esmtp-keyword on the extended MAIL command specifies whether or not the message should be included in any failed DSN issued for this message transmission. If the RET esmtp-keyword is used, it MUST have an associated esmtp-value, which is one of the following keywords:

FULL requests that the entire message be returned in any "failed" delivery status notification issued for this recipient.

HDRS requests that only the headers of the message be returned.





The FULL and HDRS keywords may be spelled in any combination of upper and lower case letters.

If no RET parameter is supplied, the MTA MAY return either the headers of the message or the entire message for any DSN containing indication of failed deliveries.

Note that the RET parameter only applies to DSNs that indicate delivery failure for at least one recipient. If a DSN contains no indications of delivery failure, only the headers of the message should be returned.

### 5.4 The ENVID parameter to the ESMTP MAIL command

The ENVID esmtp-keyword of the SMTP MAIL command is used to specify an "envelope identifier" to be transmitted along with the message and included in any DSNs issued for any of the recipients named in this SMTP transaction. The purpose of the envelope identifier is to allow the sender of a message to identify the transaction for which the DSN was issued.

The ABNF for the ENVID parameter is:

```
envid-parameter = "ENVID=" xtext
```

The ENVID esmtp-keyword MUST have an associated esmtp-value. No meaning is assigned by the mail system to the presence or absence of this parameter or to any esmtp-value associated with this parameter; the information is used only by the sender or his user agent. The ENVID parameter MAY be up to 100 characters in length.

### 5.5 Restrictions on the use of Delivery Status Notification parameters

The RET and ENVID parameters MUST NOT appear more than once each in any single MAIL command. If more than one of either of these parameters appears in a MAIL command, the ESMTP server SHOULD respond with "501 syntax error in parameters or arguments".

The NOTIFY and ORCPT parameters MUST NOT appear more than once in any RCPT command. If more than one of either of these parameters appears in a RCPT command, the ESMTP server SHOULD respond with "501 syntax error in parameters or arguments".

## 6. Conformance requirements

The Simple Mail Transfer Protocol (SMTP) is used by Message Transfer Agents (MTAs) when accepting, relaying, or gatewaying mail, as well as User Agents (UAs) when submitting mail to the mail transport

system. The DSN extension to SMTP may be used to allow UAs to convey the sender's requests as to when DSNs should be issued. A UA which claims to conform to this specification must meet certain requirements as described below.

Typically, a message transfer agent (MTA) which supports SMTP will assume, at different times, both the role of a SMTP client and an SMTP server, and may also provide local delivery, gatewaying to foreign environments, forwarding, and mailing list expansion. An MTA which, when acting as an SMTP server, issues the DSN keyword in response to the EHLO command, MUST obey the rules below for a "conforming SMTP client" when acting as a client, and a "conforming SMTP server" when acting as a server. The term "conforming MTA" refers to an MTA which conforms to this specification, independent of its role of client or server.

### 6.1 SMTP protocol interactions

The following rules apply to SMTP transactions in which any of the ENVID, NOTIFY, RET, or ORCPT keywords are used:

(a) If an SMTP client issues a MAIL command containing a valid ENVID parameter and associated esmtp-value and/or a valid RET parameter and associated esmtp-value, a conforming SMTP server MUST return the same reply-code as it would to the same MAIL command without the ENVID and/or RET parameters. A conforming SMTP server MUST NOT refuse a MAIL command based on the absence or presence of valid ENVID or RET parameters, or on their associated esmtp-values.

However, if the associated esmtp-value is not valid (i.e. contains illegal characters), or if there is more than one ENVID or RET parameter in a particular MAIL command, the server MUST issue the reply-code 501 with an appropriate message (e.g. "syntax error in parameter").

(b) If an SMTP client issues a RCPT command containing any valid NOTIFY and/or ORCPT parameters, a conforming SMTP server MUST return the same response as it would to the same RCPT command without those NOTIFY and/or ORCPT parameters. A conforming SMTP server MUST NOT refuse a RCPT command based on the presence or absence of any of these parameters.

However, if any of the associated esmtp-values are not valid, or if there is more than one of any of these parameters in a particular RCPT command, the server SHOULD issue the response "501 syntax error in parameter".







### 6.2 Handling of messages received via SMTP

This section describes how a conforming MTA should handle any messages received via SMTP.

NOTE: A DSN MUST NOT be returned to the sender for any message for which the return address from the SMTP MAIL command was NULL ("<>"), even if the sender's address is available from other sources (e.g. the message header). However, the MTA which would otherwise issue a DSN SHOULD inform the local postmaster of delivery failures through some appropriate mechanism that will not itself result in the generation of DSNs.

DISCUSSION: RFC 1123, section 2.3.3 requires error notifications to be sent with a NULL return address ("reverse-path"). This creates an interesting situation when a message arrives with one or more nonfunctional recipient addresses in addition to a nonfunctional return address. When delivery to one of the recipient addresses fails, the MTA will attempt to send a nondelivery notification to the return address, setting the return address on the notification to NULL. When the delivery of this notification fails, the MTA attempting delivery of that notification sees a NULL return address. If that MTA were not to inform anyone of the situation, the original message would be silently lost. Furthermore, a nonfunctional return address is often indicative of a configuration problem in the sender's MTA. Reporting the condition to the local postmaster may help to speed correction of such errors.

#### 6.2.1 Relay of messages to other conforming SMTP servers

The following rules govern the behavior of a conforming MTA, when relaying a message which was received via the SMTP protocol, to an SMTP server that supports the Delivery Status Notification service extension:

(a) Any ENVID parameter included in the MAIL command when a message was received, MUST also appear on the MAIL command with which the message is relayed, with the same associated esmtp-value. If no ENVID parameter was included in the MAIL command when the message was received, the ENVID parameter MUST NOT be supplied when the message is relayed.

(b) Any RET parameter included in the MAIL command when a message was received, MUST also appear on the MAIL command with which the message is relayed, with the same associated esmtp-value. If no RET parameter was included in the MAIL command when the message was received, the RET parameter MUST NOT supplied when the message is relayed.

(c) If the NOTIFY parameter was supplied for a recipient when the message was received, the RCPT command issued when the message is relayed MUST also contain the NOTIFY parameter along with its associated esmtp-value. If the NOTIFY parameter was not supplied for a recipient when the message was received, the NOTIFY parameter MUST NOT be supplied for that recipient when the message is relayed.

(d) If any ORCPT parameter was present in the RCPT command for a recipient when the message was received, an ORCPT parameter with the identical original-recipient-address MUST appear in the RCPT command issued for that recipient when relaying the message. (For example, the MTA therefore MUST NOT change the case of any alphabetic characters in an ORCPT parameter.)

If no ORCPT parameter was present in the RCPT command when the message was received, an ORCPT parameter MAY be added to the RCPT command when the message is relayed. If an ORCPT parameter is added by the relaying MTA, it MUST contain the recipient address from the RCPT command used when the message was received by that MTA.

6.2.2 Relay of messages to non-conforming SMTP servers

The following rules govern the behavior of a conforming MTA (in the role of client), when relaying a message which was received via the SMTP protocol, to an SMTP server that does not support the Delivery Status Notification service extension:

(a) ENVID, NOTIFY, RET, or ORCPT parameters MUST NOT be issued when relaying the message.

(b) If the NOTIFY parameter was supplied for a recipient, with an esmtp-value containing the keyword SUCCESS, and the SMTP server returns a success (2xx) reply-code in response to the RCPT command, the client MUST issue a "relayed" DSN for that recipient.

(c) If the NOTIFY parameter was supplied for a recipient with an esmtp-value containing the keyword FAILURE, and the SMTP server returns a permanent failure (5xx) reply-code in response to the RCPT command, the client MUST issue a "failed" DSN for that recipient.

(d) If the NOTIFY parameter was supplied for a recipient with an esmtp-value of NEVER, the client MUST NOT issue a DSN for that recipient, regardless of the reply-code returned by the SMTP server. However, if the server returned a failure (5xx) reply-code, the client MAY inform the local postmaster of the delivery failure via an appropriate mechanism that will not itself result in the generation of DSNs.





When attempting to relay a message to an SMTP server that does not support this extension, and if NOTIFY=NEVER was specified for some recipients of that message, a conforming SMTP client MAY relay the message for those recipients in a separate SMTP transaction, using an empty reverse-path in the MAIL command. This will prevent DSNs from being issued for those recipients by MTAs that conform to [1].

(e) If a NOTIFY parameter was not supplied for a recipient, and the SMTP server returns a success (2xx) reply-code in response to a RCPT command, the client MUST NOT issue any DSN for that recipient.

(f) If a NOTIFY parameter was not supplied for a recipient, and the SMTP server returns a permanent failure (5xx) reply-code in response to a RCPT command, the client MUST issue a "failed" DSN for that recipient.

6.2.3 Local delivery of messages

The following rules govern the behavior of a conforming MTA upon successful delivery of a message that was received via the SMTP protocol, to a local recipient's mailbox:

"Delivery" means that the message has been placed in the recipient's mailbox. For messages which are transmitted to a mailbox for later retrieval via IMAP [6], POP [7] or a similar message access protocol, "delivery" occurs when the message is made available to the IMAP (POP, etc.) service, rather than when the message is retrieved by the recipient's user agent.

Similarly, for a recipient address which corresponds to a mailing list exploder, "delivery" occurs when the message is made available to that list exploder, even though the list exploder might refuse to deliver that message to the list recipients.

(a) If the NOTIFY parameter was supplied for that recipient, with an esmtp-value containing the SUCCESS keyword, the MTA MUST issue a "delivered" DSN for that recipient.

(b) If the NOTIFY parameter was supplied for that recipient which did not contain the SUCCESS keyword, the MTA MUST NOT issue a DSN for that recipient.

(c) If the NOTIFY parameter was not supplied for that recipient, the MTA MUST NOT issue a DSN.

6.2.4 Gatewaying a message into a foreign environment

The following rules govern the behavior of a conforming MTA, when gatewaying a message that was received via the SMTP protocol, into a foreign (non-SMTP) environment:

(a) If the the foreign environment is capable of issuing appropriate notifications under the conditions requested by the NOTIFY parameter, and the conforming MTA can ensure that any notification thus issued will be translated into a DSN and delivered to the original sender, then the MTA SHOULD gateway the message into the foreign environment, requesting notification under the desired conditions, without itself issuing a DSN.

(b) If a NOTIFY parameter was supplied with the SUCCESS keyword, but the destination environment cannot return an appropriate notification on successful delivery, the MTA SHOULD issue a "relayed" DSN for that recipient.

(c) If a NOTIFY parameter was supplied with an esmtp-keyword of NEVER, a DSN MUST NOT be issued. If possible, the MTA SHOULD direct the destination environment to not issue delivery notifications for that recipient.

(d) If the NOTIFY parameter was not supplied for a particular recipient, a DSN SHOULD NOT be issued by the gateway. The gateway SHOULD attempt to ensure that appropriate notification will be provided by the foreign mail environment if eventual delivery failure occurs, and that no notification will be issued on successful delivery.

(e) When gatewaying a message into a foreign environment, the return-of-content conditions specified by any RET parameter are nonbinding; however, the MTA SHOULD attempt to honor the request using whatever mechanisms exist in the foreign environment.

6.2.5 Delays in delivery

If a conforming MTA receives a message via the SMTP protocol, and is unable to deliver or relay the message to one or more recipients for an extended length of time (to be determined by the MTA), it MAY issue a "delayed" DSN for those recipients, subject to the following conditions:

(a) If the NOTIFY parameter was supplied for a recipient and its value included the DELAY keyword, a "delayed" DSN MAY be issued.

(b) If the NOTIFY parameter was not supplied for a recipient, a "delayed" DSN MAY be issued.







(c) If the NOTIFY parameter was supplied which did not contain the DELAY keyword, a "delayed" DSN MUST NOT be issued.

NOTE: Although delay notifications are common in present-day electronic mail, a conforming MTA is never required to issue "delayed" DSNs. The DELAY keyword of the NOTIFY parameter is provided to allow the SMTP client to specifically request (by omitting the DELAY parameter) that "delayed" DSNs NOT be issued.

6.2.6 Failure of a conforming MTA to deliver a message

The following rules govern the behavior of a conforming MTA which received a message via the SMTP protocol, and is unable to deliver a message to a recipient specified in the SMTP transaction:

(a) If a NOTIFY parameter was supplied for the recipient with an esmtp-keyword containing the value FAILURE, a "failed" DSN MUST be issued by the MTA.

(b) If a NOTIFY parameter was supplied for the recipient which did not contain the value FAILURE, a DSN MUST NOT be issued for that recipient. However, the MTA MAY inform the local postmaster of the delivery failure via some appropriate mechanism which does not itself result in the generation of DSNs.

(c) If no NOTIFY parameter was supplied for the recipient, a "failed" DSN MUST be issued.

NOTE: Some MTAs are known to forward undeliverable messages to the local postmaster or "dead letter" mailbox. This is still considered delivery failure, and does not diminish the requirement to issue a "failed" DSN under the conditions defined elsewhere in this memo. If a DSN is issued for such a recipient, the Action value MUST be "failed".

6.2.7 Forwarding, aliases, and mailing lists

Delivery of a message to a local email address usually causes the message to be stored in the recipient's mailbox. However, MTAs commonly provide a facility where a local email address can be designated as an "alias" or "mailing list"; delivery to that address then causes the message to be forwarded to each of the (local or remote) recipient addresses associated with the alias or list. It is also common to allow a user to optionally "forward" her mail to one or more alternate addresses. If this feature is enabled, her mail is redistributed to those addresses instead of being deposited in her mailbox.

Following the example of [9] (section 5.3.6), this document defines the difference between an "alias" and "mailing list" as follows: When forwarding a message to the addresses associated with an "alias", the envelope return address (e.g. SMTP MAIL FROM) remains intact. However, when forwarding a message to the addresses associated with a "mailing list", the envelope return address is changed to that of the administrator of the mailing list. This causes DSNs and other nondelivery reports resulting from delivery to the list members to be sent to the list administrator rather than the sender of the original message.

The DSN processing for aliases and mailing lists is as follows:

6.2.7.1 mailing lists

When a message is delivered to a list submission address (i.e. placed in the list's mailbox for incoming mail, or accepted by the process that redistributes the message to the list subscribers), this is considered final delivery for the original message. If the NOTIFY parameter for the list submission address contained the SUCCESS keyword, a "delivered" DSN MUST be returned to the sender of the original message.

NOTE: Some mailing lists are able to reject message submissions, based on the content of the message, the sender's address, or some other criteria. While the interface between such a mailing list and its MTA is not well-defined, it is important that DSNs NOT be issued by both the MTA (to report successful delivery to the list), and the list (to report message rejection using a "failure" DSN.)

However, even if a "delivered" DSN was issued by the MTA, a mailing list which rejects a message submission MAY notify the sender that the message was rejected using an ordinary message instead of a DSN.

Whenever a message is redistributed to an mailing list,

(a) The envelope return address is rewritten to point to the list maintainer. This address MAY be that of a process that recognizes DSNs and processes them automatically, but it MUST forward unrecognized messages to the human responsible for the list.

(b) The ENVID, NOTIFY, RET, and ORCPT parameters which accompany the redistributed message MUST NOT be derived from those of the original message.

(c) The NOTIFY and RET parameters MAY be specified by the local postmaster or the list administrator. If ORCPT parameters are supplied during redistribution to the list subscribers, they SHOULD







contain the addresses of the list subscribers in the format used by the mailing list.

6.2.7.2 single-recipient aliases

Under normal circumstances, when a message arrives for an "alias" which has a single forwarding address, a DSN SHOULD NOT be issued. Any ENVID, NOTIFY, RET, or ORCPT parameters SHOULD be propagated with the message as it is redistributed to the forwarding address.

6.2.7.3 multiple-recipient aliases

An "alias" with multiple recipient addresses may be handled in any of the following ways:

(a) Any ENVID, NOTIFY, RET, or ORCPT parameters are NOT propagated when relaying the message to any of the forwarding addresses. If the NOTIFY parameter for the alias contained the SUCCESS keyword, the MTA issues a "relayed" DSN. (In effect, the MTA treats the message as if it were being relayed into an environment that does not support DSNs.)

(b) Any ENVID, NOTIFY, RET, or ORCPT parameters (or the equivalent requests if the message is gatewayed) are propagated to EXACTLY one of the forwarding addresses. No DSN is issued. (This is appropriate when aliasing is used to forward a message to a "vacation" auto-responder program in addition to the local mailbox.)

(c) Any ENVID, RET, or ORCPT parameters are propagated to all forwarding addresses associated with that alias. The NOTIFY parameter is propagated to the forwarding addresses, except that it any SUCCESS keyword is removed. If the original NOTIFY parameter for the alias contained the SUCCESS keyword, an "expanded" DSN is issued for the alias. If the NOTIFY parameter for the alias did not contain the SUCCESS keyword, no DSN is issued for the alias.

6.2.7.4 confidential forwarding addresses

If it is desired to maintain the confidentiality of a recipient's forwarding address, the forwarding may be treated as if it were a mailing list. A DSN will be issued, if appropriate, upon "delivery" to the recipient address specified by the sender. When the message is forwarded it will have a new envelope return address. Any DSNs which result from delivery failure of the forwarded message will not be returned to the original sender of the message and thus not expose the recipient's forwarding address.

6.2.8 DSNs describing delivery to multiple recipients

A single DSN may describe attempts to deliver a message to multiple recipients of that message. If a DSN is issued for some recipients in an SMTP transaction and not for others according to the rules above, the DSN SHOULD NOT contain information for recipients for whom DSNs would not otherwise have been issued.

6.3 Handling of messages from other sources

For messages which originated from "local" users (whatever that means), the specifications under which DSNs should be generated can be communicated to the MTA via any protocol agreed on between the sender's mail composer (user agent) and the MTA. The local MTA can then either relay the message, or issue appropriate delivery status notifications. However, if such requests are transmitted within the message itself (for example in the message headers), the requests MUST be removed from the message before it is transmitted via SMTP.

For messages gatewayed from non-SMTP sources and further relayed by SMTP, the gateway SHOULD, using the SMTP extensions described here, attempt to provide the delivery reporting conditions expected by the source mail environment. If appropriate, any DSNs returned to the source environment SHOULD be translated into the format expected in that environment.

6.4 Implementation limits

A conforming MTA MUST accept ESMTP parameters of at least the following sizes:

(a) ENVID parameter: 100 characters.

(b) NOTIFY parameter: 28 characters.

(c) ORCPT parameter: 500 characters.

(d) RET parameter: 8 characters.

The maximum sizes for the ENVID and ORCPT parameters are intended to be adequate for the transmission of "foreign" envelope identifier and original recipient addresses. However, user agents which use SMTP as a message submission protocol SHOULD NOT generate ENVID parameters which are longer than 38 characters in length.

A conforming MTA MUST be able to accept SMTP command-lines which are at least 1036 characters long (530 characters for the ORCPT and NOTIFY parameters of the RCPT command, in addition to the 512





characters required by [1]). If other SMTP extensions are supported by the MTA, the MTA MUST be able to accept a command-line large enough for each SMTP command and any combination of ESMTP parameters which may be used with that command.

7. Format of delivery notifications

The format of delivery status notifications is defined in [5], which uses the framework defined in [8]. Delivery status notifications are to be returned to the sender of the original message as outlined below.

7.1 SMTP Envelope to be used with delivery status notifications

The DSN sender address (in the SMTP MAIL command) MUST be a null reverse-path ("<>"), as required by section 5.3.3 of [9]. The DSN recipient address (in the RCPT command) is copied from the MAIL command which accompanied the message for which the DSN is being issued. When transmitting a DSN via SMTP, the RET parameter MUST NOT be used. The NOTIFY parameter MAY be used, but its value MUST be NEVER. The ENVID parameter (with a newly generated envelope-id) and/or ORCPT parameter MAY be used.

7.2 Contents of the DSN

A DSN is transmitted as a MIME message with a top-level content-type of multipart/report (as defined in [5]).

The multipart/report content-type may be used for any of several kinds of reports generated by the mail system. When multipart/report is used to convey a DSN, the report-type parameter of the multipart/report content-type is "delivery-status".

As described in [8], the first component of a multipart/report content-type is a human readable explanation of the report. For a DSN, the second component of the multipart/report is of content-type message/delivery-status (defined in [5]). The third component of the multipart/report consists of the original message or some portion thereof. When the value of the RET parameter is FULL, the full message SHOULD be returned for any DSN which conveys notification of delivery failure. (However, if the length of the message is greater than some implementation-specified length, the MTA MAY return only the headers even if the RET parameter specified FULL.) If a DSN contains no notifications of delivery failure, the MTA SHOULD return only the headers.

The third component must have an appropriate content-type label. Issues concerning selection of the content-type are discussed in [8].

### 7.3 Message/delivery-status fields

The message/delivery-status content-type defines a number of fields, with general specifications for their contents. The following requirements for any DSNs generated in response to a message received by the SMTP protocol by a conforming SMTP server, are in addition to the requirements defined in [5] for the message/delivery-status type.

When generating a DSN for a message which was received via the SMTP protocol, a conforming MTA will generate the following fields of the message/delivery-status body part:

(a) if an ENVID parameter was present on the MAIL command, an Original-Envelope-ID field MUST be supplied, and the value associated with the ENVID parameter must appear in that field. If the message was received via SMTP with no ENVID parameter, the Original-Envelope-ID field MUST NOT be supplied.

Since the ENVID parameter is encoded as xtext, but the Original-Envelope-ID header is NOT encoded as xtext, the MTA must decode the xtext encoding when copying the ENVID value to the Original-Envelope-ID field.

(b) The Reporting-MTA field MUST be supplied. If Reporting MTA can determine its fully-qualified Internet domain name, the MTA-name-type subfield MUST be "dns", and the field MUST contain the fully-qualified domain name of the Reporting MTA. If the fully-qualified Internet domain name of the Reporting MTA is not known (for example, for an SMTP server which is not directly connected to the Internet), the Reporting-MTA field may contain any string identifying the MTA, however, in this case the MTA-name-type subfield MUST NOT be "dns". A MTA-name-type subfield value of "x-local-hostname" is suggested.

(c) Other per-message fields as defined in [5] MAY be supplied as appropriate.

(d) If the ORCPT parameter was provided for this recipient, the Original-Recipient field MUST be supplied, with its value taken from the ORCPT parameter. If no ORCPT parameter was provided for this recipient, the Original-Recipient field MUST NOT appear.

(e) The Final-Recipient field MUST be supplied. It MUST contain the recipient address from the message envelope. If the message was received via SMTP, the address-type will be "rfc822".

(f) The Action field MUST be supplied.





(g) The Status field MUST be supplied, using a status-code from [10]. If there is no specific code which suitably describes a delivery failure, either 4.0.0 (temporary failure), or 5.0.0 (permanent failure) MUST be used.

(h) For DSNs resulting from attempts to relay a message to one or more recipients via SMTP, the Remote-MTA field MUST be supplied for each of those recipients. The mta-name-type subfields of those Remote-MTA fields will be "dns".

(i) For DSNs resulting from attempts to relay a message to one or more recipients via SMTP, the Diagnostic-Code MUST be supplied for each of those recipients. The diagnostic-type subfield will be "smtp". See section 9.2(a) of this document for a description of the "smtp" diagnostic-code.

(j) For DSNs resulting from attempts to relay a message to one or more recipients via SMTP, an SMTP-Remote-Recipient extension field MAY be supplied for each recipient, which contains the address of that recpient which was presented to the remote SMTP server.

(k) Other per-recipient fields defined in [5] MAY appear, as appropriate.

## 8. Acknowledgments

The author wishes to thank Eric Allman, Harald Alvestrand, Jim Conklin, Bryan Costales, Peter Cowen, Dave Crocker, Roger Fajman, Ned Freed, Marko Kaittola, Steve Kille, John Klensin, Anastasios Kotsikonas, John Gardiner Myers, Julian Onions, Jacob Palme, Marshall Rose, Greg Vaudreuil, and Klaus Weide for their suggestions for improvement of this document.

## 9. Appendix - Type-Name Definitions

The following type names are defined for use in DSN fields generated by conforming SMTP-based MTAs:

### 9.1 "rfc822" address-type

The "rfc822" address-type is to be used when reporting Internet electronic mail address in the Original-Recipient and Final-Recipient DSN fields.

(a) address-type name: rfc822

(b) syntax for mailbox addresses

RFC822 mailbox addresses are generally expected to be of the form

[route] addr-spec

where "route" and "addr-spec" are defined in [2], and the "domain" portions of both "route" and "addr-spec" are fully-qualified domain names that are registered in the DNS. However, an MTA MUST NOT modify an address obtained from the message envelope to force it to conform to syntax rules.

(c) If addresses of this type are not composed entirely of graphic characters from the US-ASCII repertoire, a specification for how they are to be encoded as graphic US-ASCII characters in a DSN Original-Recipient or Final-Recipient DSN field.

RFC822 addresses consist entirely of graphic characters from the US-ASCII repertoire, so no translation is necessary.

### 9.2 "smtp" diagnostic-type

The "smtp" diagnostic-type is to be used when reporting SMTP reply-codes in Diagnostic-Code DSN fields.

(a) diagnostic-type name: SMTP

(b) A description of the syntax to be used for expressing diagnostic codes of this type as graphic characters from the US-ASCII repertoire.

An SMTP diagnostic-code is of the form

*( 3*DIGIT "-" *text ) 3*DIGIT SPACE *text





For a single-line SMTP reply to an SMTP command, the diagnostic-code SHOULD be an exact transcription of the reply. For multi-line SMTP replies, it is necessary to insert a SPACE before each line after the first. For example, an SMTP reply of:

550-mailbox unavailable
550 user has moved with no forwarding address

could appear as follows in a Diagnostic-Code DSN field:

Diagnostic-Code: smtp ; 550-mailbox unavailable
550 user has moved with no forwarding address

(c) A list of valid diagnostic codes of this type and the meaning of each code.

SMTP reply-codes are currently defined in [1], [4], and [9]. Additional codes may be defined by other RFCs.

### 9.3 "dns" MTA-name-type

The "dns" MTA-name-type should be used in the Reporting-MTA field. An MTA-name of type "dns" is a fully-qualified domain name. The name must be registered in the DNS, and the address Postmaster@{mta-name} must be valid.

(a) MTA-name-type name: dns

(b) A description of the syntax of MTA names of this type, using BNF, regular expressions, ASN.1, or other non-ambiguous language.

MTA names of type "dns" SHOULD be valid Internet domain names. If such domain names are not available, a domain-literal containing the internet protocol address is acceptable. Such domain names generally conform to the following syntax:

domain = real-domain / domain-literal

real-domain = sub-domain *("." sub-domain)

sub-domain = atom

domain-literal = "[" 1*3DIGIT 3("." 1*3DIGIT) "]"

where "atom" and "DIGIT" are defined in [2].

(c) If MTA names of this type do not consist entirely of graphic characters from the US-ASCII repertoire, a specification for how an MTA name of this type should be expressed as a sequence of graphic US-ASCII characters.

MTA names of type "dns" consist entirely of graphic US-ASCII characters, so no translation is needed.

## 10. Appendix - Example

This example traces the flow of a single message addressed to multiple recipients. The message is sent by Alice@Pure-Heart.ORG to Bob@Big-Bucks.COM, Carol@Ivory.EDU, Dana@Ivory.EDU, Eric@Bombs.AF.MIL, Fred@Bombs.AF.MIL, and George@Tax-ME.GOV, with a variety of per-recipient options. The message is successfully delivered to Bob, Dana (via a gateway), Eric, and Fred. Delivery fails for Carol and George.

NOTE: Formatting rules for RFCs require that no line be longer than 72 characters. Therefore, in the following examples, some SMTP commands longer than 72 characters are printed on two lines, with the first line ending in "\". In an actual SMTP transaction, such a command would be sent as a single line (i.e. with no embedded CRLFs), and without the "\" character that appears in these examples.

### 10.1 Submission

Alice's user agent sends the message to the SMTP server at Pure-Heart.ORG. Note that while this example uses SMTP as a mail submission protocol, other protocols could also be used.

```
<<< 220 Pure-Heart.ORG SMTP server here
>>> EHLO Pure-Heart.ORG
<<< 250-Pure-Heart.ORG
<<< 250-DSN
<<< 250-EXPN
<<< 250 SIZE
>>> MAIL FROM:<Alice@Pure-Heart.ORG> RET=HDRS ENVID=QQ314159
<<< 250 <Alice@Pure-Heart.ORG> sender ok
>>> RCPT TO:<Bob@Big-Bucks.COM> NOTIFY=SUCCESS \
    ORCPT=rfc822;Bob@Big-Bucks.COM
<<< 250 <Bob@Big-Bucks.COM> recipient ok
>>> RCPT TO:<Carol@Ivory.EDU> NOTIFY=FAILURE \
    ORCPT=rfc822;Carol@Ivory.EDU
<<< 250 <Carol@Ivory.EDU> recipient ok
>>> RCPT TO:<Dana@Ivory.EDU> NOTIFY=SUCCESS,FAILURE \
    ORCPT=rfc822;Dana@Ivory.EDU
<<< 250 <Dana@Ivory.EDU> recipient ok
```





```
>>> RCPT TO:<Eric@Bombs.AF.MIL> NOTIFY=FAILURE \
    ORCPT=rfc822;Eric@Bombs.AF.MIL
<<< 250 <Eric@Bombs.AF.MIL> recipient ok
>>> RCPT TO:<Fred@Bombs.AF.MIL> NOTIFY=NEVER
<<< 250 <Fred@Bombs.AF.MIL> recipient ok
>>> RCPT TO:<George@Tax-ME.GOV> NOTIFY=FAILURE \
    ORCPT=rfc822;George@Tax-ME.GOV
<<< 250 <George@Tax-ME.GOV> recipient ok
>>> DATA
<<< 354 okay, send message
>>> (message goes here)
>>> .
<<< 250 message accepted
>>> QUIT
<<< 221 goodbye
```

### 10.2 Relay to Big-Bucks.COM

The SMTP at Pure-Heart.ORG then relays the message to Big-Bucks.COM. (For the purpose of this example, mail.Big-Bucks.COM is the primary mail exchanger for Big-Bucks.COM).

```
<<< 220 mail.Big-Bucks.COM says hello
>>> EHLO Pure-Heart.ORG
<<< 250-mail.Big-Bucks.COM
<<< 250 DSN
>>> MAIL FROM:<Alice@Pure-Heart.ORG> RET=HDRS ENVID=QQ314159
<<< 250 sender okay
>>> RCPT TO:<Bob@Big-Bucks.COM> NOTIFY=SUCCESS \
    ORCPT=rfc822;Bob@Big-Bucks.COM
<<< 250 recipient okay
>>> DATA
<<< 354 send message
>>> (message goes here)
>>> .
<<< 250 message received
>>> QUIT
<<< 221 bcnu
```

### 10.3 Relay to Ivory.EDU

The SMTP at Pure-Heart.ORG relays the message to Ivory.EDU, which (as it happens) is a gateway to a LAN-based mail system that accepts SMTP mail and supports the DSN extension.

```
<<< 220 Ivory.EDU gateway to FooMail(tm) here
>>> EHLO Pure-Heart.ORG
<<< 250-Ivory.EDU
```

```
<<< 250 DSN
>>> MAIL FROM:<Alice@Pure-Heart.ORG> RET=HDRS ENVID=QQ314159
<<< 250 ok
>>> RCPT TO:<Carol@Ivory.EDU> NOTIFY=FAILURE \
    ORCPT=rfc822;Carol@Ivory.EDU
<<< 550 error - no such recipient
>>> RCPT TO:<Dana@Ivory.EDU> NOTIFY=SUCCESS,FAILURE \
    ORCPT=rfc822;Dana@Ivory.EDU
<<< 250 recipient ok
>>> DATA
<<< 354 send message, end with '.'
>>> (message goes here)
>>> .
<<< 250 message received
>>> QUIT
<<< 221 bye
```

Note that since the Ivory.EDU refused to accept mail for Carol@Ivory.EDU, and the sender specified NOTIFY=FAILURE, the sender-SMTP (in this case Pure-Heart.ORG) must generate a DSN.

### 10.4 Relay to Bombs.AF.MIL

The SMTP at Pure-Heart.ORG relays the message to Bombs.AF.MIL, which does not support the SMTP extension. Because the sender specified NOTIFY=NEVER for recipient Fred@Bombs.AF.MIL, the SMTP at Pure-Heart.ORG chooses to send the message for that recipient in a separate transaction with a reverse-path of <>.

```
<<< 220-Bombs.AF.MIL reporting for duty.
<<< 220 Electronic mail is to be used for official business only.
>>> EHLO Pure-Heart.ORG
<<< 502 command not implemented
>>> RSET
<<< 250 reset
>>> HELO Pure-Heart.ORG
<<< 250 Bombs.AF.MIL
>>> MAIL FROM:<Alice@Pure-Heart.ORG>
<<< 250 ok
>>> RCPT TO:<Eric@Bombs.AF.MIL>
<<< 250 ok
>>> DATA
<<< 354 send message
>>> (message goes here)
>>> .
<<< 250 message accepted
>>> MAIL FROM:<>
<<< 250 ok
```





```
>>> RCPT TO:<Fred@Bombs.AF.MIL>
<<< 250 ok
>>> DATA
<<< 354 send message
>>> (message goes here)
>>> .
<<< 250 message accepted
>>> QUIT
<<< 221 Bombs.AF.MIL closing connection
```

### 10.5 Forward from George@Tax-ME.GOV to Sam@Boondoggle.GOV

The SMTP at Pure-Heart.ORG relays the message to Tax-ME.GOV. (this step is not shown). MTA Tax-ME.GOV then forwards the message to Sam@Boondoggle.GOV (shown below). Both Tax-ME.GOV and Pure-Heart.ORG support the SMTP DSN extension. Note that RET, ENVID, and ORCPT all retain their original values.

```
<<< 220 BoonDoggle.GOV says hello
>>> EHLO Pure-Heart.ORG
<<< 250-mail.Big-Bucks.COM
<<< 250 DSN
>>> MAIL FROM:<Alice@Pure-Heart.ORG> RET=HDRS ENVID=QQ314159
<<< 250 sender okay
>>> RCPT TO:<Sam@Boondoggle.GOV> NOTIFY=SUCCESS \
    ORCPT=rfc822;George@Tax-ME.GOV
<<< 250 recipient okay
>>> DATA
<<< 354 send message
>>> (message goes here)
>>> .
<<< 250 message received
>>> QUIT
<<< 221 bcnu
```

10.6 "Delivered" DSN for Bob@Big-Bucks.COM

MTA mail.Big-Bucks.COM successfully delivers the message to Bob@Big-Bucks.COM. Because the sender specified NOTIFY=SUCCESS, mail.Big-Bucks.COM issues the following DSN, and sends it to Alice@Pure-Heart.ORG.

```
To: Alice@Pure-Heart.ORG
From: postmaster@mail.Big-Bucks.COM
Subject: Delivery Notification (success) for Bob@Big-Bucks.COM
Content-Type: multipart/report; report-type=delivery-status;
    boundary=abcde
MIME-Version: 1.0

--abcde
Content-type: text/plain; charset=us-ascii

Your message (id QQ314159) was successfully delivered to
Bob@Big-Bucks.COM.

--abcde
Content-type: message/delivery-status

Reporting-MTA: dns; mail.Big-Bucks.COM
Original-Envelope-ID: QQ314159

Original-Recipient: rfc822;Bob@Big-Bucks.COM
Final-Recipient: rfc822;Bob@Big-Bucks.COM
Action: delivered
Status: 2.0.0

--abcde
Content-type: message/rfc822

(headers of returned message go here)

--abcde--
```





10.7 Failed DSN for Carol@Ivory.EDU

Because delivery to Carol failed and the sender specified NOTIFY=FAILURE for Carol@Ivory.EDU, MTA Pure-Heart.ORG (the SMTP client to which the failure was reported via SMTP) issues the following DSN.

```
To: Alice@Pure-Heart.ORG
From: postmaster@Pure-Heart.ORG
Subject: Delivery Notification (failure) for Carol@Ivory.EDU
Content-Type: multipart/report; report-type=delivery-status;
              boundary=bcdef
MIME-Version: 1.0

--bcdef
Content-type: text/plain; charset=us-ascii

Your message (id QQ314159) could not be delivered to
Carol@Ivory.EDU.

A transcript of the session follows:

(while talking to Ivory.EDU)
>>> RCPT TO:<Carol@Ivory.EDU> NOTIFY=FAILURE
<<< 550 error - no such recipient

--bcdef
Content-type: message/delivery-status

Reporting-MTA: dns; Pure-Heart.ORG
Original-Envelope-ID: QQ314159

Original-Recipient: rfc822;Carol@Ivory.EDU
Final-Recipient: rfc822;Carol@Ivory.EDU
SMTP-Remote-Recipient: Carol@Ivory.EDU
Diagnostic-Code: smtp; 550 error - no such recipient
Action: failed
Status: 5.0.0

--bcdef
Content-type: message/rfc822

(headers of returned message go here)

--bcdef--
```

10.8 Relayed DSN For Dana@Ivory.EDU

Although the mail gateway Ivory.EDU supports the DSN SMTP extension, the LAN mail system attached to its other side does not generate positive delivery confirmations. So Ivory.EDU issues a "relayed" DSN:

```
To: Alice@Pure-Heart.ORG
From: postmaster@Ivory.EDU
Subject: mail relayed for Dana@Ivory.EDU
Content-Type: multipart/report; report-type=delivery-status;
    boundary=cdefg
MIME-Version: 1.0

--cdefg
Content-type: text/plain; charset=us-ascii

Your message (addressed to Dana@Ivory.EDU) was successfully
relayed to:

ymail!Dana

by the FooMail gateway at Ivory.EDU.

Unfortunately, the remote mail system does not support
confirmation of actual delivery.  Unless delivery to ymail!Dana
fails, this will be the only delivery status notification sent.

--cdefg
Content-type: message/delivery-status

Reporting-MTA: dns; Ivory.EDU
Original-Envelope-ID: QQ314159

Original-Recipient: rfc822;Dana@Ivory.EDU
Final-Recipient: rfc822;Dana@Ivory.EDU
Action: relayed
Status: 2.0.0

--cdefg
Content-type: message/rfc822

(headers of returned message go here)

--cdefg--
```





10.9 Failure notification for Sam@Boondoggle.GOV

The message originally addressed to George@Tax-ME.GOV was forwarded to Sam@Boondoggle.GOV, but the MTA for Boondoggle.GOV was unable to deliver the message due to a lack of disk space in Sam's mailbox. After trying for several days, Boondoggle.GOV returned the following DSN:

```
To: Alice@BigHeart.ORG
From: Postmaster@Boondoggle.GOV
Subject: Delivery failure for Sam@Boondoggle.GOV
Content-Type: multipart/report; report-type=delivery-status;
              boundary=defgh
MIME-Version: 1.0

--defgh
Your message, originally addressed to George@Tax-ME.GOV, and forwarded
from there to Sam@Boondoggle.GOV could not be delivered, for the
following reason:

write error to mailbox, disk quota exceeded

--defgh
Content-type: message/delivery-status

Reporting-MTA: Boondoggle.GOV
Original-Envelope-ID: QQ314159

Original-Recipient: rfc822;George@Tax-ME.GOV
Final-Recipient: rfc822;Sam@Boondoggle.GOV
Action: failed
Status: 4.2.2 (disk quota exceeded)

--defgh
Content-type: message/rfc822

(headers of returned message go here)

--defgh--
```

## 11. References

[1] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[2] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[3] Westine, A., and J. Postel, "Problems with the Maintenance of Large Mailing Lists.", RFC 1211, USC/Information Sciences Institute, March 1991.

[4] Klensin, J., Freed, N., Rose, M., Stefferud, E., and D. Crocker, "SMTP Service Extensions", RFC 1651, MCI, Innosoft, Dover Beach Consulting, Inc., Network Management Associates, Inc., Silicon Graphics, Inc., July 1994.

[5] Moore, K., and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 1894, University of Tennessee, Octel Network Services, January 1996.

[6] Crispin, M., "Internet Message Access Protocol - Version 4", RFC 1730, University of Washington, 20 December 1994.

[7] Myers, J., and M. Rose, "Post Office Protocol - Version 3", RFC 1725, Carnegie Mellon, Dover Beach Consulting, November 1994.

[8] Vaudreuil, G., "The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages", RFC 1892, Octel Network Services, January 1996.

[9] Braden, R., Editor, "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, IETF, October 1989.

[10] Vaudreuil, G., "Enhanced Mail System Status Codes", RFC 1893, Octel Network Services, January 1996.

## 12. Author's Address

Keith Moore
University of Tennessee
107 Ayres Hall
Knoxville, TN 37996-1301
USA

EMail: moore@cs.utk.edu

Network Working Group G. Vaudreuil
Request for Comments: 1892 Octel Network Services
Category: Standards Track January 1996

# The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## 1. The Multipart/Report MIME content-type

The Multipart/Report MIME content-type is a general "family" or "container" type for electronic mail reports of any kind. Although this memo defines only the use of the Multipart/Report content-type with respect to delivery status reports, mail processing programs will benefit if a single content-type is used to for all kinds of reports.

The Multipart/Report content-type is defined as follows:

MIME type name: multipart
MIME subtype name: report
Required parameters: boundary, report-type
Optional parameters: none
Encoding considerations: 7bit should always be adequate
Security considerations: see section 4 of this memo.

The syntax of Multipart/Report is identical to the Multipart/Mixed content type defined in [MIME]. When used to send a report, the Multipart/Report content-type must be the top-level MIME content type for any report message. The report-type parameter identifies the type of report. The parameter is the MIME content sub-type of the second body part of the Multipart/Report.

User agents and gateways must be able to automatically determine that a message is a mail system report and should be processed as such. Placing the Multipart/Report as the outermost content provides a mechanism whereby an auto-processor may detect through parsing the RFC 822 headers that the message is a report.





The Multipart/Report content-type contains either two or three sub-parts, in the following order:

(1) [required] The first body part contains human readable message. The purpose of this message is to provide an easily-understood description of the condition(s) that caused the report to be generated, for a human reader who may not have an user agent capable of interpreting the second section of the Multipart/Report.

The text in the first section may be in any MIME standards-track content-type, charset, or language. Where a description of the error is desired in several languages or several media, a Multipart/Alternative construct may be used.

This body part may also be used to send detailed information that cannot be easily formatted into a Message/Report body part.

(2) [required] A machine parsable body part containing an account of the reported message handling event. The purpose of this body part is to provide a machine-readable description of the condition(s) which caused the report to be generated, along with details not present in the first body part that may be useful to human experts. An initial body part, Message/delivery-status is defined in [DSN]

(3) [optional] A body part containing the returned message or a portion thereof. This information may be useful to aid human experts in diagnosing problems. (Although it may also be useful to allow the sender to identify the message which the report was issued, it is hoped that the envelope-id and original-recipient-address returned in the Message/Report body part will replace the traditional use of the returned content for this purpose.)

Return of content may be wasteful of network bandwidth and a variety of implementation strategies can be used. Generally the sender should choose the appropriate strategy and inform the recipient of the required level of returned content required. In the absence of an explicit request for level of return of content such as that provided in [DRPT], the agent which generated the delivery service report should return the full message content.

When data not encoded in 7 bits is to be returned, and the return path is not guaranteed to be 8-bit capable, two options are available. The origional message MAY be reencoded into a legal 7 bit MIME message or the Text/RFC822-Headers content-type MAY be used to return only the origional message headers.

## 2. The Text/RFC822-Headers MIME content-type

The Text/RFC822-Headers MIME content-type provides a mechanism to label and return only the RFC 822 headers of a failed message. These headers are not the complete message and should not be returned as a Message/RFC822. The returned headers are useful for identifying the failed message and for diagnostics based on the received: lines.

The Text/RFC822-Headers content-type is defined as follows:

```
MIME type name: Text
MIME subtype name: RFC822-Headers
Required parameters: None
Optional parameters: none
Encoding considerations: 7 bit is sufficient for normal RFC822
       headers, however, if the headers are broken and require
       encoding, they may be encoded in quoted-printable.
Security considerations: see section 4 of this memo.
```

The Text/RFC822-headers body part should contain all the RFC822 header lines from the message which caused the report. The RFC822 headers include all lines prior to the blank line in the message. They include the MIME-Version and MIME Content- headers.

## 3. References

[DSN] Moore, K., and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 1894, University of Tennessee, Octel Network Services, January 1996.

[RFC822] Crocker, D., "Standard for the format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[MIME] Borenstein, N., and N. Freed, "Multipurpose Internet Mail Extensions", RFC 1521, Bellcore, Innosoft, June 1992.

[DRPT] Moore, K., "SMTP Service Extension for Delivery Status Notifications", RFC 1891, University of Tennessee, January 1996.

## 4. Security Considerations

Automated use of report types without authentication presents several security issues. Forging negative reports presents the opportunity for denial-of-service attacks when the reports are used for automated maintenance of directories or mailing lists. Forging positive reports may cause the sender to incorrectly believe a message was delivered when it was not.







## 5. Author's Address

Gregory M. Vaudreuil
Octel Network Services
17060 Dallas Parkway
Dallas, TX 75248-1905

Phone: +1-214-733-2722
EMail: Greg.Vaudreuil@Octel.com

Network Working Group
Request for Comments: 1893
Category: Standards Track

G. Vaudreuil
Octel Network Services

January 1996

# Enhanced Mail System Status Codes

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## 1. Overview

There currently is not a standard mechanism for the reporting of mail system errors except for the limited set offered by SMTP and the system specific text descriptions sent in mail messages. There is a pressing need for a rich machine readable status code for use in delivery status notifications [DSN]. This document proposes a new set of status codes for this purpose.

SMTP [SMTP] error codes have historically been used for reporting mail system errors. Because of limitations in the SMTP code design, these are not suitable for use in delivery status notifications. SMTP provides about 12 useful codes for delivery reports. The majority of the codes are protocol specific response codes such as the 354 response to the SMTP data command. Each of the 12 useful codes are each overloaded to indicate several error conditions each. SMTP suffers some scars from history, most notably the unfortunate damage to the reply code extension mechanism by uncontrolled use. This proposal facilitates future extensibility by requiring the client to interpret unknown error codes according to the theory of codes while requiring servers to register new response codes.

The SMTP theory of reply codes partitioned in the number space such a manner that the remaining available codes will not provide the space needed. The most critical example is the existence of only 5 remaining codes for mail system errors. The mail system classification includes both host and mailbox error conditions. The remaining third digit space would be completely consumed as needed to indicate MIME and media conversion errors and security system errors.

A revision to the SMTP theory of reply codes to better distribute the error conditions in the number space will necessarily be incompatible with SMTP. Further, consumption of the remaining reply-code number





space for delivery notification reporting will reduce the available codes for new ESMTP extensions.

The following proposal is based on the SMTP theory of reply codes. It adopts the success, permanent error, and transient error semantics of the first value, with a further description and classification in the second. This proposal re-distributes the classifications to better distribute the error conditions, such as separating mailbox from host errors.

## 2. Status Codes

This document defines a new set of status codes to report mail system conditions. These status codes are intended to be used for media and language independent status reporting. They are not intended for system specific diagnostics.

The syntax of the new status codes is defined as:

```
status-code = class "." subject "." detail
class = "2"/"4"/"5"
subject = 1*3digit
detail = 1*3digit
```

White-space characters and comments are NOT allowed within a status-code. Each numeric sub-code within the status-code MUST be expressed without leading zero digits.

Status codes consist of three numerical fields separated by ".". The first sub-code indicates whether the delivery attempt was successful. The second sub-code indicates the probable source of any delivery anomalies, and the third sub-code indicates a precise error condition.

The codes space defined is intended to be extensible only by standards track documents. Mail system specific status codes should be mapped as close as possible to the standard status codes. Servers should send only defined, registered status codes. System specific errors and diagnostics should be carried by means other than status codes.

New subject and detail codes will be added over time. Because the number space is large, it is not intended that published status codes will ever be redefined or eliminated. Clients should preserve the extensibility of the code space by reporting the general error described in the subject sub-code when the specific detail is unrecognized.

The class sub-code provides a broad classification of the status. The enumerated values the class are defined as:

2.X.X Success

Success specifies that the DSN is reporting a positive delivery action. Detail sub-codes may provide notification of transformations required for delivery.

4.X.X Persistent Transient Failure

A persistent transient failure is one in which the message as sent is valid, but some temporary event prevents the successful sending of the message. Sending in the future may be successful.

5.X.X Permanent Failure

A permanent failure is one which is not likely to be resolved by resending the message in the current form. Some change to the message or the destination must be made for successful delivery.

A client must recognize and report class sub-code even where subsequent subject sub-codes are unrecognized.

The subject sub-code classifies the status. This value applies to each of the three classifications. The subject sub-code, if recognized, must be reported even if the additional detail provided by the detail sub-code is not recognized. The enumerated values for the subject sub-code are:

X.0.X Other or Undefined Status

There is no additional subject information available.

X.1.X Addressing Status

The address status reports on the originator or destination address. It may include address syntax or validity. These errors can generally be corrected by the sender and retried.

X.2.X Mailbox Status

Mailbox status indicates that something having to do with the mailbox has cause this DSN. Mailbox issues are assumed to be under the general control of the recipient.





X.3.X Mail System Status

Mail system status indicates that something having to do with the destination system has caused this DSN. System issues are assumed to be under the general control of the destination system administrator.

X.4.X Network and Routing Status

The networking or routing codes report status about the delivery system itself. These system components include any necessary infrastructure such as directory and routing services. Network issues are assumed to be under the control of the destination or intermediate system administrator.

X.5.X Mail Delivery Protocol Status

The mail delivery protocol status codes report failures involving the message delivery protocol. These failures include the full range of problems resulting from implementation errors or an unreliable connection. Mail delivery protocol issues may be controlled by many parties including the originating system, destination system, or intermediate system administrators.

X.6.X Message Content or Media Status

The message content or media status codes report failures involving the content of the message. These codes report failures due to translation, transcoding, or otherwise unsupported message media. Message content or media issues are under the control of both the sender and the receiver, both of whom must support a common set of supported content-types.

X.7.X Security or Policy Status

The security or policy status codes report failures involving policies such as per-recipient or per-host filtering and cryptographic operations. Security and policy status issues are assumed to be under the control of either or both the sender and recipient. Both the sender and recipient must permit the exchange of messages and arrange the exchange of necessary keys and certificates for cryptographic operations.

## 3. Enumerated Status Codes

The following section defines and describes the detail sub-code. The detail value provides more information about the status and is defined relative to the subject of the status.

### 3.1 Other or Undefined Status

#### X.0.0 Other undefined Status

Other undefined status is the only undefined error code. It should be used for all errors for which only the class of the error is known.

### 3.2 Address Status

#### X.1.0 Other address status

Something about the address specified in the message caused this DSN.

#### X.1.1 Bad destination mailbox address

The mailbox specified in the address does not exist. For Internet mail names, this means the address portion to the left of the "@" sign is invalid. This code is only useful for permanent failures.

#### X.1.2 Bad destination system address

The destination system specified in the address does not exist or is incapable of accepting mail. For Internet mail names, this means the address portion to the right of the "@" is invalid for mail. This codes is only useful for permanent failures.

#### X.1.3 Bad destination mailbox address syntax

The destination address was syntactically invalid. This can apply to any field in the address. This code is only useful for permanent failures.

#### X.1.4 Destination mailbox address ambiguous

The mailbox address as specified matches one or more recipients on the destination system. This may result if a heuristic address mapping algorithm is used to map the specified address to a local mailbox name.





#### X.1.5 Destination address valid

This mailbox address as specified was valid. This status code should be used for positive delivery reports.

#### X.1.6 Destination mailbox has moved, No forwarding address

The mailbox address provided was at one time valid, but mail is no longer being accepted for that address. This code is only useful for permanent failures.

#### X.1.7 Bad sender's mailbox address syntax

The sender's address was syntactically invalid. This can apply to any field in the address.

#### X.1.8 Bad sender's system address

The sender's system specified in the address does not exist or is incapable of accepting return mail. For domain names, this means the address portion to the right of the "@" is invalid for mail.

### 3.3 Mailbox Status

#### X.2.0 Other or undefined mailbox status

The mailbox exists, but something about the destination mailbox has caused the sending of this DSN.

#### X.2.1 Mailbox disabled, not accepting messages

The mailbox exists, but is not accepting messages. This may be a permanent error if the mailbox will never be re-enabled or a transient error if the mailbox is only temporarily disabled.

#### X.2.2 Mailbox full

The mailbox is full because the user has exceeded a per-mailbox administrative quota or physical capacity. The general semantics implies that the recipient can delete messages to make more space available. This code should be used as a persistent transient failure.

X.2.3 Message length exceeds administrative limit

A per-mailbox administrative message length limit has been exceeded. This status code should be used when the per-mailbox message length limit is less than the general system limit. This code should be used as a permanent failure.

X.2.4 Mailing list expansion problem

The mailbox is a mailing list address and the mailing list was unable to be expanded. This code may represent a permanent failure or a persistent transient failure.

3.4 Mail system status

X.3.0 Other or undefined mail system status

The destination system exists and normally accepts mail, but something about the system has caused the generation of this DSN.

X.3.1 Mail system full

Mail system storage has been exceeded. The general semantics imply that the individual recipient may not be able to delete material to make room for additional messages. This is useful only as a persistent transient error.

X.3.2 System not accepting network messages

The host on which the mailbox is resident is not accepting messages. Examples of such conditions include an immanent shutdown, excessive load, or system maintenance. This is useful for both permanent and permanent transient errors.

X.3.3 System not capable of selected features

Selected features specified for the message are not supported by the destination system. This can occur in gateways when features from one domain cannot be mapped onto the supported feature in another.







X.3.4 Message too big for system

The message is larger than per-message size limit. This limit may either be for physical or administrative reasons. This is useful only as a permanent error.

X.3.5 System incorrectly configured

The system is not configured in a manner which will permit it to accept this message.

3.5 Network and Routing Status

X.4.0 Other or undefined network or routing status

Something went wrong with the networking, but it is not clear what the problem is, or the problem cannot be well expressed with any of the other provided detail codes.

X.4.1 No answer from host

The outbound connection attempt was not answered, either because the remote system was busy, or otherwise unable to take a call. This is useful only as a persistent transient error.

X.4.2 Bad connection

The outbound connection was established, but was otherwise unable to complete the message transaction, either because of time-out, or inadequate connection quality. This is useful only as a persistent transient error.

X.4.3 Directory server failure

The network system was unable to forward the message, because a directory server was unavailable. This is useful only as a persistent transient error.

The inability to connect to an Internet DNS server is one example of the directory server failure error.

X.4.4 Unable to route

The mail system was unable to determine the next hop for the message because the necessary routing information was unavailable from the directory server. This is useful for both permanent and persistent transient errors.

A DNS lookup returning only an SOA (Start of Administration) record for a domain name is one example of the unable to route error.

X.4.5 Mail system congestion

The mail system was unable to deliver the message because the mail system was congested. This is useful only as a persistent transient error.

X.4.6 Routing loop detected

A routing loop caused the message to be forwarded too many times, either because of incorrect routing tables or a user forwarding loop. This is useful only as a persistent transient error.

X.4.7 Delivery time expired

The message was considered too old by the rejecting system, either because it remained on that host too long or because the time-to-live value specified by the sender of the message was exceeded. If possible, the code for the actual problem found when delivery was attempted should be returned rather than this code. This is useful only as a persistent transient error.

3.6 Mail Delivery Protocol Status

X.5.0 Other or undefined protocol status

Something was wrong with the protocol necessary to deliver the message to the next hop and the problem cannot be well expressed with any of the other provided detail codes.

X.5.1 Invalid command

A mail transaction protocol command was issued which was either out of sequence or unsupported. This is useful only as a permanent error.

X.5.2 Syntax error

A mail transaction protocol command was issued which could not be interpreted, either because the syntax was wrong or the command is unrecognized. This is useful only as a permanent error.







X.5.3 Too many recipients

More recipients were specified for the message than could have been delivered by the protocol. This error should normally result in the segmentation of the message into two, the remainder of the recipients to be delivered on a subsequent delivery attempt. It is included in this list in the event that such segmentation is not possible.

X.5.4 Invalid command arguments

A valid mail transaction protocol command was issued with invalid arguments, either because the arguments were out of range or represented unrecognized features. This is useful only as a permanent error.

X.5.5 Wrong protocol version

A protocol version mis-match existed which could not be automatically resolved by the communicating parties.

3.7 Message Content or Message Media Status

X.6.0 Other or undefined media error

Something about the content of a message caused it to be considered undeliverable and the problem cannot be well expressed with any of the other provided detail codes.

X.6.1 Media not supported

The media of the message is not supported by either the delivery protocol or the next system in the forwarding path. This is useful only as a permanent error.

X.6.2 Conversion required and prohibited

The content of the message must be converted before it can be delivered and such conversion is not permitted. Such prohibitions may be the expression of the sender in the message itself or the policy of the sending host.

X.6.3 Conversion required but not supported

The message content must be converted to be forwarded but such conversion is not possible or is not practical by a host in the forwarding path. This condition may result when an ESMTP gateway supports 8bit transport but is not able to

downgrade the message to 7 bit as required for the next hop.

X.6.4 Conversion with loss performed

This is a warning sent to the sender when message delivery was successfully but when the delivery required a conversion in which some data was lost. This may also be a permanant error if the sender has indicated that conversion with loss is prohibited for the message.

X.6.5 Conversion Failed

A conversion was required but was unsuccessful. This may be useful as a permanent or persistent temporary notification.

3.8 Security or Policy Status

X.7.0 Other or undefined security status

Something related to security caused the message to be returned, and the problem cannot be well expressed with any of the other provided detail codes. This status code may also be used when the condition cannot be further described because of security policies in force.

X.7.1 Delivery not authorized, message refused

The sender is not authorized to send to the destination. This can be the result of per-host or per-recipient filtering. This memo does not discuss the merits of any such filtering, but provides a mechanism to report such. This is useful only as a permanent error.

X.7.2 Mailing list expansion prohibited

The sender is not authorized to send a message to the intended mailing list. This is useful only as a permanent error.

X.7.3 Security conversion required but not possible

A conversion from one secure messaging protocol to another was required for delivery and such conversion was not possible. This is useful only as a permanent error.







X.7.4 Security features not supported

A message contained security features such as secure authentication which could not be supported on the delivery protocol. This is useful only as a permanent error.

X.7.5 Cryptographic failure

A transport system otherwise authorized to validate or decrypt a message in transport was unable to do so because necessary information such as key was not available or such information was invalid.

X.7.6 Cryptographic algorithm not supported

A transport system otherwise authorized to validate or decrypt a message was unable to do so because the necessary algorithm was not supported.

X.7.7 Message integrity failure

A transport system otherwise authorized to validate a message was unable to do so because the message was corrupted or altered. This may be useful as a permanent, transient persistent, or successful delivery code.

## 4. References

[SMTP] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[DSN] Moore, K., and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 1894, University of Tennessee, Octel Network Services, January 1996.

## 5. Security Considerations

This document describes a status code system with increased precision. Use of these status codes may disclose additional information about how an internal mail system is implemented beyond that currently available.

## 6. Acknowledgments

The author wishes to offer special thanks to Harald Alvestrand, Marko Kaittola, and Keith Moore for their extensive review and constructive suggestions.

## 7. Author's Address

Gregory M. Vaudreuil
Octel Network Services
17060 Dallas Parkway
Suite 214
Dallas, TX 75248-1905

Voice/Fax: +1-214-733-2722
EMail: Greg.Vaudreuil@Octel.com







## 8. Appendix - Collected Status Codes

| | |
|---|---|
| X.1.0 | Other address status |
| X.1.1 | Bad destination mailbox address |
| X.1.2 | Bad destination system address |
| X.1.3 | Bad destination mailbox address syntax |
| X.1.4 | Destination mailbox address ambiguous |
| X.1.5 | Destination mailbox address valid |
| X.1.6 | Mailbox has moved |
| X.1.7 | Bad sender's mailbox address syntax |
| X.1.8 | Bad sender's system address |
| X.2.0 | Other or undefined mailbox status |
| X.2.1 | Mailbox disabled, not accepting messages |
| X.2.2 | Mailbox full |
| X.2.3 | Message length exceeds administrative limit. |
| X.2.4 | Mailing list expansion problem |
| X.3.0 | Other or undefined mail system status |
| X.3.1 | Mail system full |
| X.3.2 | System not accepting network messages |
| X.3.3 | System not capable of selected features |
| X.3.4 | Message too big for system |
| X.4.0 | Other or undefined network or routing status |
| X.4.1 | No answer from host |
| X.4.2 | Bad connection |
| X.4.3 | Routing server failure |
| X.4.4 | Unable to route |
| X.4.5 | Network congestion |
| X.4.6 | Routing loop detected |
| X.4.7 | Delivery time expired |
| X.5.0 | Other or undefined protocol status |
| X.5.1 | Invalid command |
| X.5.2 | Syntax error |
| X.5.3 | Too many recipients |
| X.5.4 | Invalid command arguments |
| X.5.5 | Wrong protocol version |
| X.6.0 | Other or undefined media error |
| X.6.1 | Media not supported |
| X.6.2 | Conversion required and prohibited |
| X.6.3 | Conversion required but not supported |
| X.6.4 | Conversion with loss performed |
| X.6.5 | Conversion failed |

| | |
|---|---|
| X.7.0 | Other or undefined security status |
| X.7.1 | Delivery not authorized, message refused |
| X.7.2 | Mailing list expansion prohibited |
| X.7.3 | Security conversion required but not possible |
| X.7.4 | Security features not supported |
| X.7.5 | Cryptographic failure |
| X.7.6 | Cryptographic algorithm not supported |
| X.7.7 | Message integrity failure |

Network Working Group
Request for Comments: 1894
Category: Standards Track

K. Moore
University of Tennessee
G. Vaudreuil
Octel Network Services
January 1996

# An Extensible Message Format for Delivery Status Notifications

## Status of this Memo



## Abstract

This memo defines a MIME content-type that may be used by a message transfer agent (MTA) or electronic mail gateway to report the result of an attempt to deliver a message to one or more recipients. This content-type is intended as a machine-processable replacement for the various types of delivery status notifications currently used in Internet electronic mail.

Because many messages are sent between the Internet and other messaging systems (such as X.400 or the so-called "LAN-based" systems), the DSN protocol is designed to be useful in a multi-protocol messaging environment. To this end, the protocol described in this memo provides for the carriage of "foreign" addresses and error codes, in addition to those normally used in Internet mail. Additional attributes may also be defined to support "tunneling" of foreign notifications through Internet mail.

Any questions, comments, and reports of defects or ambiguities in this specification may be sent to the mailing list for the NOTARY working group of the IETF, using the address <notifications@cs.utk.edu>. Requests to subscribe to the mailing list should be addressed to <notifications-request@cs.utk.edu>. Implementors of this specification are encouraged to subscribe to the mailing list, so that they will quickly be informed of any problems which might hinder interoperability.

NOTE: This document is a Proposed Standard. If and when this protocol is submitted for Draft Standard status, any normative text (phrases containing SHOULD, SHOULD NOT, MUST, MUST NOT, or MAY) in this document will be re-evaluated in light of implementation





experience, and are thus subject to change.

## 1. Introduction

This memo defines a MIME [1] content-type for delivery status notifications (DSNs). A DSN can be used to notify the sender of a message of any of several conditions: failed delivery, delayed delivery, successful delivery, or the gatewaying of a message into an environment that may not support DSNs. The "message/delivery-status" content-type defined herein is intended for use within the framework of the "multipart/report" content type defined in [2].

This memo defines only the format of the notifications. An extension to the Simple Message Transfer Protocol (SMTP) [3] to fully support such notifications is the subject of a separate memo [4].

### 1.1 Purposes

The DSNs defined in this memo are expected to serve several purposes:

(a) Inform human beings of the status of message delivery processing, as well as the reasons for any delivery problems or outright failures, in a manner which is largely independent of human language;

(b) Allow mail user agents to keep track of the delivery status of messages sent, by associating returned DSNs with earlier message transmissions;

(c) Allow mailing list exploders to automatically maintain their subscriber lists when delivery attempts repeatedly fail;

(d) Convey delivery and non-delivery notifications resulting from attempts to deliver messages to "foreign" mail systems via a gateway;

(e) Allow "foreign" notifications to be tunneled through a MIME-capable message system and back into the original messaging system that issued the original notification, or even to a third messaging system;

(f) Allow language-independent, yet reasonably precise, indications of the reason for the failure of a message to be delivered (once status codes of sufficient precision are defined); and

(g) Provide sufficient information to remote MTA maintainers (via "trouble tickets") so that they can understand the nature of reported errors. This feature is used in the case that failure to deliver a message is due to the malfunction of a remote MTA and the

sender wants to report the problem to the remote MTA administrator.

1.2 Requirements

These purposes place the following constraints on the notification protocol:

(a) It must be readable by humans as well as being machine-parsable.

(b) It must provide enough information to allow message senders (or the user agents) to unambiguously associate a DSN with the message that was sent and the original recipient address for which the DSN is issued (if such information is available), even if the message was forwarded to another recipient address.

(c) It must be able to preserve the reason for the success or failure of a delivery attempt in a remote messaging system, using the "language" (mailbox addresses and status codes) of that remote system.

(d) It must also be able to describe the reason for the success or failure of a delivery attempt, independent of any particular human language or of the "language" of any particular mail system.

(e) It must preserve enough information to allow the maintainer of a remote MTA to understand (and if possible, reproduce) the conditions that caused a delivery failure at that MTA.

(f) For any notifications issued by foreign mail systems, which are translated by a mail gateway to the DSN format, the DSN must preserve the "type" of the foreign addresses and error codes, so that these may be correctly interpreted by gateways.

A DSN contains a set of per-message fields which identify the message and the transaction during which the message was submitted, along with other fields that apply to all delivery attempts described by the DSN. The DSN also includes a set of per-recipient fields to convey the result of the attempt to deliver the message to each of one or more recipients.

1.3 Terminology

A message may be transmitted through several message transfer agents (MTAs) on its way to a recipient. For a variety of reasons, recipient addresses may be rewritten during this process, so each MTA may potentially see a different recipient address. Depending on the purpose for which a DSN is used, different formats of a particular recipient address will be needed.







Several DSN fields are defined in terms of the view from a particular MTA in the transmission. The MTAs are assigned the following names:

(a) Original MTA

The Original MTA is the one to which the message is submitted for delivery by the sender of the message.

(b) Reporting MTA

For any DSN, the Reporting MTA is the one which is reporting the results of delivery attempts described in the DSN.

If the delivery attempts described occurred in a "foreign" (non-Internet) mail system, and the DSN was produced by translating the foreign notice into DSN format, the Reporting MTA will still identify the "foreign" MTA where the delivery attempts occurred.

(c) Received-From MTA

The Received-From MTA is the MTA from which the Reporting MTA received the message, and accepted responsibility for delivery of the message.

(d) Remote MTA

If an MTA determines that it must relay a message to one or more recipients, but the message cannot be transferred to its "next hop" MTA, or if the "next hop" MTA refuses to accept responsibility for delivery of the message to one or more of its intended recipients, the relaying MTA may need to issue a DSN on behalf of the recipients for whom the message cannot be delivered. In this case the relaying MTA is the Reporting MTA, and the "next hop" MTA is known as the Remote MTA.

Figure 1 illustrates the relationship between the various MTAs.

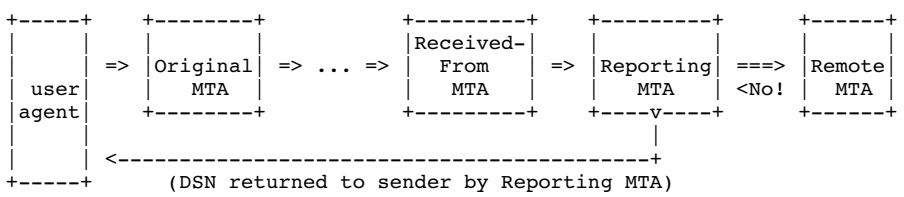



Figure 1. Original, Received-From, Reporting and Remote MTAs

Each of these MTAs may provide information which is useful in a DSN:

+ Ideally, the DSN will contain the address of each recipient as originally specified to the Original MTA by the sender of the message. This version of the address is needed (rather than a forwarding address or some modified version of the original address) so that the sender may compare the recipient address in the DSN with the address in the sender's records (e.g. an address book for an individual, the list of subscribers for a mailing list) and take appropriate action.

  Similarly, the DSN might contain an "envelope identifier" that was known to both the sender's user agent and the Original MTA at the time of message submission, and which, if included in the DSN, can be used by the sender to keep track of which messages were or were not delivered.

+ If a message was (a) forwarded to a different address than that specified by the sender, (b) gatewayed to a different mail system than that used by the sender, or (c) subjected to address rewriting during transmission, the "final" form of the recipient address (i.e. the one seen by the Reporting MTA) will be different than the original (sender-specified) recipient address. Just as the sender's user agent (or the sender) prefers the original recipient address, so the "final" address is needed when reporting a problem to the postmaster of the site where message delivery failed, because only the final recipient address will allow her to reproduce the conditions that caused the failure.

+ A "failed" DSN should contain the most accurate explanation for the delivery failure that is available. For ease of interpretation, this information should be a format which is independent of the mail transport system that issued the DSN. However, if a foreign error





code is translated into some transport-independent format, some information may be lost. It is therefore desirable to provide both a transport-independent status code and a mechanism for reporting transport-specific codes. Depending on the circumstances that produced delivery failure, the transport-specific code might be obtained from either the Reporting MTA or the Remote MTA.

Since different values for "recipient address" and "delivery status code" are needed according to the circumstance in which a DSN will be used, and since the MTA that issues the DSN cannot anticipate those circumstances, the DSN format described here may contain both the original and final forms of a recipient address, and both a transport-independent and a transport-specific indication of delivery status.

Extension fields may also be added by the Reporting MTA as needed to provide additional information for use in a trouble ticket or to preserve information for tunneling of foreign delivery reports through Internet DSNs.

The Original, Reporting, and Remote MTAs may exist in very different environments and use dissimilar transport protocols, MTA names, address formats, and delivery status codes. DSNs therefore do not assume any particular format for mailbox addresses, MTA names, or transport-specific status codes. Instead, the various DSN fields that carry such quantities consist of a "type" subfield followed by a subfield whose contents are ordinary text characters, and the format of which is indicated by the "type" subfield. This allows a DSN to convey these quantities regardless of format.

## 2. Format of a Delivery Status Notification

A DSN is a MIME message with a top-level content-type of multipart/report (defined in [2]). When a multipart/report content is used to transmit a DSN:

(a) The report-type parameter of the multipart/report content is "delivery-status".

(b) The first component of the multipart/report contains a human-readable explanation of the DSN, as described in [2].

(c) The second component of the multipart/report is of content-type message/delivery-status, described in section 2.1 of this document.

(d) If the original message or a portion of the message is to be returned to the sender, it appears as the third component of the multipart/report.

NOTE: For delivery status notifications gatewayed from foreign systems, the headers of the original message may not be available. In this case the third component of the DSN may be omitted, or it may contain "simulated" RFC 822 headers which contain equivalent information. In particular, it is very desirable to preserve the subject, date, and message-id (or equivalent) fields from the original message.

The DSN MUST be addressed (in both the message header and the transport envelope) to the return address from the transport envelope which accompanied the original message for which the DSN was generated. (For a message that arrived via SMTP, the envelope return address appears in the MAIL FROM command.)

The From field of the message header of the DSN SHOULD contain the address of a human who is responsible for maintaining the mail system at the Reporting MTA site (e.g. Postmaster), so that a reply to the DSN will reach that person. Exception: if a DSN is translated from a foreign delivery report, and the gateway performing the translation cannot determine the appropriate address, the From field of the DSN MAY be the address of a human who is responsible for maintaining the gateway.

The envelope sender address of the DSN SHOULD be chosen to ensure that no delivery status reports will be issued in response to the DSN itself, and MUST be chosen so that DSNs will not generate mail loops. Whenever an SMTP transaction is used to send a DSN, the MAIL FROM command MUST use a NULL return address, i.e. "MAIL FROM:<>".

A particular DSN describes the delivery status for exactly one message. However, an MTA MAY report on the delivery status for several recipients of the same message in a single DSN. Due to the nature of the mail transport system (where responsibility for delivery of a message to its recipients may be split among several MTAs, and delivery to any particular recipient may be delayed), multiple DSNs may be still be issued in response to a single message submission.







## 2.1 The message/delivery-status content-type

The message/delivery-status content-type is defined as follows:

| | |
|---|---|
| MIME type name: | message |
| MIME subtype name: | delivery-status |
| Optional parameters: | none |
| Encoding considerations: | "7bit" encoding is sufficient and MUST be used to maintain readability when viewed by non-MIME mail readers. |
| Security considerations: | discussed in section 4 of this memo. |

The message/delivery-status report type for use in the multipart/report is "delivery-status".

The body of a message/delivery-status consists of one or more "fields" formatted according to the ABNF of RFC 822 header "fields" (see [6]). The per-message fields appear first, followed by a blank line. Following the per-message fields are one or more groups of per-recipient fields. Each group of per-recipient fields is preceded by a blank line. Using the ABNF of RFC 822, the syntax of the message/delivery-status content is as follows:

```
delivery-status-content =
     per-message-fields 1*( CRLF per-recipient-fields )
```

The per-message fields are described in section 2.2. The per-recipient fields are described in section 2.3.

### 2.1.1 General conventions for DSN fields

Since these fields are defined according to the rules of RFC 822, the same conventions for continuation lines and comments apply. Notification fields may be continued onto multiple lines by beginning each additional line with a SPACE or HTAB. Text which appears in parentheses is considered a comment and not part of the contents of that notification field. Field names are case-insensitive, so the names of notification fields may be spelled in any combination of upper and lower case letters. Comments in DSN fields may use the "encoded-word" construct defined in [7].

A number of DSN fields are defined to have a portion of a field body of "xtext". "xtext" is used to allow encoding sequences of octets which contain values outside the range [1-127 decimal] of traditional ASCII characters, and also to allow comments to be inserted in the data. Any octet may be encoded as "+" followed by two upper case

hexadecimal digits. (The "+" character MUST be encoded as "+2B".) With certain exceptions, octets that correspond to ASCII characters may be represented as themselves. SPACE and HTAB characters are ignored. Comments may be included by enclosing them in parenthesis. Except within comments, encoded-words such as defined in [7] may NOT be used in xtext.

"xtext" is formally defined as follows:

```
xtext = *( xchar / hexchar / linear-white-space / comment )

xchar = any ASCII CHAR between "!" (33) and "~" (126) inclusive,
     except for "+", "\" and "(".
```

"hexchar"s are intended to encode octets that cannot be represented as plain text, either because they are reserved, or because they are non-printable. However, any octet value may be represented by a "hexchar".

```
hexchar = ASCII "+" immediately followed by two upper case
     hexadecimal digits
```

When encoding an octet sequence as xtext:

+ Any ASCII CHAR between "!" and "~" inclusive, except for "+", "\", and "(", MAY be encoded as itself. (Some CHARs in this range may also be encoded as "hexchar"s, at the implementor's discretion.)

+ ASCII CHARs that fall outside the range above must be encoded as "hexchar".

+ Line breaks (CR LF SPACE) MAY be inserted as necessary to keep line lengths from becoming excessive.

+ Comments MAY be added to clarify the meaning for human readers.

### 2.1.2 "*-type" subfields

Several DSN fields consist of a "-type" subfield, followed by a semicolon, followed by "*text". For these fields, the keyword used in the address-type, diagnostic-type, or MTA-name-type subfield indicates the expected format of the address, status-code, or MTA-name which follows.





The "-type" subfields are defined as follows:

(a) An "address-type" specifies the format of a mailbox address. For example, Internet mail addresses use the "rfc822" address-type.

```
address-type = atom
```

(b) A "diagnostic-type" specifies the format of a status code. For example, when a DSN field contains a reply code reported via the Simple Mail Transfer Protocol [3], the "smtp" diagnostic-type is used.

```
diagnostic-type = atom
```

(c) An "MTA-name-type" specifies the format of an MTA name. For example, for an SMTP server on an Internet host, the MTA name is the domain name of that host, and the "dns" MTA-name-type is used.

```
mta-name-type = atom
```

Values for address-type, diagnostic-type, and MTA-name-type are case-insensitive. Thus address-type values of "RFC822" and "rfc822" are equivalent.

The Internet Assigned Numbers Authority (IANA) will maintain a registry of address-types, diagnostic-types, and MTA-name-types, along with descriptions of the meanings and acceptable values of each, or a reference to a one or more specifications that provide such descriptions. (The "rfc822" address-type, "smtp" diagnostic-type, and "dns" MTA-name-type are defined in [4].) Registration forms for address-type, diagnostic-type, and MTA-name-type appear in section 8 of this document.

IANA will not accept registrations for any address-type, diagnostic-type, or MTA-name-type name that begins with "X-". These type names are reserved for experimental use.

### 2.1.3 Lexical tokens imported from RFC 822

The following lexical tokens, defined in [6], are used in the ABNF grammar for DSNs: atom, CHAR, comment, CR, CRLF, DIGIT, LF, linear-white-space, SPACE, text. The date-time lexical token is defined in [8].

## 2.2 Per-Message DSN Fields

Some fields of a DSN apply to all of the delivery attempts described by that DSN. These fields may appear at most once in any DSN. These fields are used to correlate the DSN with the original message transaction and to provide additional information which may be useful to gateways.

```
per-message-fields =
     [ original-envelope-id-field CRLF ]
     reporting-mta-field CRLF
     [ dsn-gateway-field CRLF ]
     [ received-from-mta-field CRLF ]
     [ arrival-date-field CRLF ]
     *( extension-field CRLF )
```

### 2.2.1 The Original-Envelope-Id field

The optional Original-Envelope-Id field contains an "envelope identifier" which uniquely identifies the transaction during which the message was submitted, and was either (a) specified by the sender and supplied to the sender's MTA, or (b) generated by the sender's MTA and made available to the sender when the message was submitted. Its purpose is to allow the sender (or her user agent) to associate the returned DSN with the specific transaction in which the message was sent.

If such an envelope identifier was present in the envelope which accompanied the message when it arrived at the Reporting MTA, it SHOULD be supplied in the Original-Envelope-Id field of any DSNs issued as a result of an attempt to deliver the message. Except when a DSN is issued by the sender's MTA, an MTA MUST NOT supply this field unless there is an envelope-identifier field in the envelope which accompanied this message on its arrival at the Reporting MTA.

The Original-Envelope-Id field is defined as follows:

```
original-envelope-id-field =
     "Original-Envelope-Id" ":" envelope-id

envelope-id = *text
```

There may be at most one Original-Envelope-Id field per DSN.

The envelope-id is CASE-SENSITIVE. The DSN MUST preserve the original case and spelling of the envelope-id.





NOTE: The Original-Envelope-Id is NOT the same as the Message-Id from the message header. The Message-Id identifies the content of the message, while the Original-Envelope-Id identifies the transaction in which the message is sent.

### 2.2.2 The Reporting-MTA DSN field

```
reporting-mta-field =
     "Reporting-MTA" ":" mta-name-type ";" mta-name

mta-name = *text
```

The Reporting-MTA field is defined as follows:

A DSN describes the results of attempts to deliver, relay, or gateway a message to one or more recipients. In all cases, the Reporting-MTA is the MTA which attempted to perform the delivery, relay, or gateway operation described in the DSN. This field is required.

Note that if an SMTP client attempts to relay a message to an SMTP server and receives an error reply to a RCPT command, the client is responsible for generating the DSN, and the client's domain name will appear in the Reporting-MTA field. (The server's domain name will appear in the Remote-MTA field.)

Note that the Reporting-MTA is not necessarily the MTA which actually issued the DSN. For example, if an attempt to deliver a message outside of the Internet resulted in a nondelivery notification which was gatewayed back into Internet mail, the Reporting-MTA field of the resulting DSN would be that of the MTA that originally reported the delivery failure, not that of the gateway which converted the foreign notification into a DSN. See Figure 2.

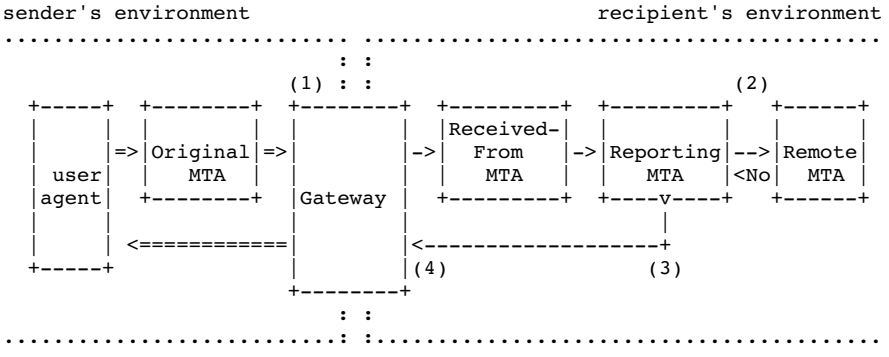



Figure 2. DSNs in the presence of gateways

(1) message is gatewayed into recipient's environment
(2) attempt to relay message fails
(3) reporting-mta (in recipient's environment) returns nondelivery notification
(4) gateway translates foreign notification into a DSN

The mta-name portion of the Reporting-MTA field is formatted according to the conventions indicated by the mta-name-type subfield. If an MTA functions as a gateway between dissimilar mail environments and thus is known by multiple names depending on the environment, the mta-name subfield SHOULD contain the name used by the environment from which the message was accepted by the Reporting-MTA.

Because the exact spelling of an MTA name may be significant in a particular environment, MTA names are CASE-SENSITIVE.

2.2.3 The DSN-Gateway field

The DSN-Gateway field indicates the name of the gateway or MTA which translated a foreign (non-Internet) delivery status notification into this DSN. This field MUST appear in any DSN which was translated by a gateway from a foreign system into DSN format, and MUST NOT appear otherwise.

dsn-gateway-field = "DSN-Gateway" ":" mta-name-type ";" mta-name





For gateways into Internet mail, the MTA-name-type will normally be "smtp", and the mta-name will be the Internet domain name of the gateway.

2.2.4 The Received-From-MTA DSN field

The optional Received-From-MTA field indicates the name of the MTA from which the message was received.

received-from-mta-field =
"Received-From-MTA" ":" mta-name-type ";" mta-name

If the message was received from an Internet host via SMTP, the contents of the mta-name subfield SHOULD be the Internet domain name supplied in the HELO or EHLO command, and the network address used by the SMTP client SHOULD be included as a comment enclosed in parentheses. (In this case, the MTA-name-type will be "smtp".)

The mta-name portion of the Received-From-MTA field is formatted according to the conventions indicated by the MTA-name-type subfield.

Since case is significant in some mail systems, the exact spelling, including case, of the MTA name SHOULD be preserved.

2.2.5 The Arrival-Date DSN field

The optional Arrival-Date field indicates the date and time at which the message arrived at the Reporting MTA. If the Last-Attempt-Date field is also provided in a per-recipient field, this can be used to determine the interval between when the message arrived at the Reporting MTA and when the report was issued for that recipient.

arrival-date-field = "Arrival-Date" ":" date-time

The date and time are expressed in RFC 822 'date-time' format, as modified by [8]. Numeric timezones ([+/-]HHMM format) MUST be used.

2.3 Per-Recipient DSN fields

A DSN contains information about attempts to deliver a message to one or more recipients. The delivery information for any particular recipient is contained in a group of contiguous per-recipient fields. Each group of per-recipient fields is preceded by a blank line.

The syntax for the group of per-recipient fields is as follows:

```
per-recipient-fields =
     [ original-recipient-field CRLF ]
     final-recipient-field CRLF
     action-field CRLF
     status-field CRLF
     [ remote-mta-field CRLF ]
     [ diagnostic-code-field CRLF ]
     [ last-attempt-date-field CRLF ]
     [ will-retry-until-field CRLF ]
     *( extension-field CRLF )
```

### 2.3.1 Original-Recipient field

The Original-Recipient field indicates the original recipient address as specified by the sender of the message for which the DSN is being issued.

```
original-recipient-field =
     "Original-Recipient" ":" address-type ";" generic-address

generic-address = *text
```

The address-type field indicates the type of the original recipient address. If the message originated within the Internet, the address-type field field will normally be "rfc822", and the address will be according to the syntax specified in [6]. The value "unknown" should be used if the Reporting MTA cannot determine the type of the original recipient address from the message envelope.

This field is optional. It should be included only if the sender-specified recipient address was present in the message envelope, such as by the SMTP extensions defined in [4]. This address is the same as that provided by the sender and can be used to automatically correlate DSN reports and message transactions.

### 2.3.2 Final-Recipient field

The Final-Recipient field indicates the recipient for which this set of per-recipient fields applies. This field MUST be present in each set of per-recipient data.





The syntax of the field is as follows:

```
final-recipient-field =
     "Final-Recipient" ":" address-type ";" generic-address
```

The generic-address subfield of the Final-Recipient field MUST contain the mailbox address of the recipient (from the transport envelope) as it was when the message was accepted for delivery by the Reporting MTA.

The Final-Recipient address may differ from the address originally provided by the sender, because it may have been transformed during forwarding and gatewaying into an totally unrecognizable mess. However, in the absence of the optional Original-Recipient field, the Final-Recipient field and any returned content may be the only information available with which to correlate the DSN with a particular message submission.

The address-type subfield indicates the type of address expected by the reporting MTA in that context. Recipient addresses obtained via SMTP will normally be of address-type "rfc822".

NOTE: The Reporting MTA is not expected to ensure that the address actually conforms to the syntax conventions of the address-type. Instead, it MUST report exactly the address received in the envelope, unless that address contains characters such as CR or LF which may not appear in a DSN field.

Since mailbox addresses (including those used in the Internet) may be case sensitive, the case of alphabetic characters in the address MUST be preserved.

### 2.3.3 Action field

The Action field indicates the action performed by the Reporting-MTA as a result of its attempt to deliver the message to this recipient address. This field MUST be present for each recipient named in the DSN.

The syntax for the action-field is:

```
action-field = "Action" ":" action-value

action-value =
     "failed" / "delayed" / "delivered" / "relayed" / "expanded"
```

The action-value may be spelled in any combination of upper and lower case characters.

"failed" indicates that the message could not be delivered to the recipient. The Reporting MTA has abandoned any attempts to deliver the message to this recipient. No further notifications should be expected.

"delayed" indicates that the Reporting MTA has so far been unable to deliver or relay the message, but it will continue to attempt to do so. Additional notification messages may be issued as the message is further delayed or successfully delivered, or if delivery attempts are later abandoned.

"delivered" indicates that the message was successfully delivered to the recipient address specified by the sender, which includes "delivery" to a mailing list exploder. It does not indicate that the message has been read. This is a terminal state and no further DSN for this recipient should be expected.

"relayed" indicates that the message has been relayed or gatewayed into an environment that does not accept responsibility for generating DSNs upon successful delivery. This action-value SHOULD NOT be used unless the sender has requested notification of successful delivery for this recipient.

"expanded" indicates that the message has been successfully delivered to the recipient address as specified by the sender, and forwarded by the Reporting-MTA beyond that destination to multiple additional recipient addresses. An action-value of "expanded" differs from "delivered" in that "expanded" is not a terminal state. Further "failed" and/or "delayed" notifications may be provided.

Using the terms "mailing list" and "alias" as defined in [4], section 7.2.7: An action-value of "expanded" is only to be used when the message is delivered to a multiple-recipient "alias". An action-value of "expanded" SHOULD NOT be used with a DSN issued on delivery of a message to a "mailing list".

NOTE ON ACTION VS. STATUS CODES: Although the 'action' field might seem to be redundant with the 'status' field, this is not the case. In particular, a "temporary failure" ("4") status code could be used with an action-value of either "delayed" or "failed". For example, assume that an SMTP client repeatedly tries to relay a message to the mail exchanger for a recipient, but fails because a query to a domain







name server timed out. After a few hours, it might issue a "delayed" DSN to inform the sender that the message had not yet been delivered. After a few days, the MTA might abandon its attempt to deliver the message and return a "failed" DSN. The status code (which would begin with a "4" to indicate "temporary failure") would be the same for both DSNs.

Another example for which the action and status codes may appear contradictory: If an MTA or mail gateway cannot deliver a message because doing so would entail conversions resulting in an unacceptable loss of information, it would issue a DSN with the 'action' field of "failure" and a status code of 'XXX'. If the message had instead been relayed, but with some loss of information, it might generate a DSN with the same XXX status-code, but with an action field of "relayed".

### 2.3.4 Status field

The per-recipient Status field contains a transport-independent status code which indicates the delivery status of the message to that recipient. This field MUST be present for each delivery attempt which is described by a DSN.

The syntax of the status field is:

```
status-field = "Status" ":" status-code

status-code = DIGIT "." 1*3DIGIT "." 1*3DIGIT

     ; White-space characters and comments are NOT allowed within a
     ; status-code, though a comment enclosed in parentheses MAY follow
     ; the last numeric subfield of the status-code. Each numeric
     ; subfield within the status-code MUST be expressed without
     ; leading zero digits.
```

Status codes thus consist of three numerical fields separated by ".". The first sub-field indicates whether the delivery attempt was successful (2 = success, 4 = persistent temporary failure, 5 = permanent failure). The second sub-field indicates the probable source of any delivery anomalies, and the third sub-field denotes a precise error condition, if known.

The initial set of status-codes is defined in [5].

2.3.5 Remote-MTA field

The value associated with the Remote-MTA DSN field is a printable ASCII representation of the name of the "remote" MTA that reported delivery status to the "reporting" MTA.

```
remote-mta-field = "Remote-MTA" ":" mta-name-type ";" mta-name
```

NOTE: The Remote-MTA field preserves the "while talking to" information that was provided in some pre-existing nondelivery reports.

This field is optional. It MUST NOT be included if no remote MTA was involved in the attempted delivery of the message to that recipient.

2.3.6 Diagnostic-Code field

For a "failed" or "delayed" recipient, the Diagnostic-Code DSN field contains the actual diagnostic code issued by the mail transport. Since such codes vary from one mail transport to another, the diagnostic-type subfield is needed to specify which type of diagnostic code is represented.

```
diagnostic-code-field =
     "Diagnostic-Code" ":" diagnostic-type ";" *text
```

NOTE: The information in the Diagnostic-Code field may be somewhat redundant with that from the Status field. The Status field is needed so that any DSN, regardless of origin, may be understood by any user agent or gateway that parses DSNs. Since the Status code will sometimes be less precise than the actual transport diagnostic code, the Diagnostic-Code field is provided to retain the latter information. Such information may be useful in a trouble ticket sent to the administrator of the Reporting MTA, or when tunneling foreign nondelivery reports through DSNs.

If the Diagnostic Code was obtained from a Remote MTA during an attempt to relay the message to that MTA, the Remote-MTA field should be present. When interpreting a DSN, the presence of a Remote-MTA field indicates that the Diagnostic Code was issued by the Remote MTA. The absence of a Remote-MTA indicates that the Diagnostic Code was issued by the Reporting MTA.

In addition to the Diagnostic-Code itself, additional textual description of the diagnostic, MAY appear in a comment enclosed in parentheses.







This field is optional, because some mail systems supply no additional information beyond that which is returned in the 'action' and 'status' fields. However, this field SHOULD be included if transport-specific diagnostic information is available.

2.3.7 Last-Attempt-Date field

The Last-Attempt-Date field gives the date and time of the last attempt to relay, gateway, or deliver the message (whether successful or unsuccessful) by the Reporting MTA. This is not necessarily the same as the value of the Date field from the header of the message used to transmit this delivery status notification: In cases where the DSN was generated by a gateway, the Date field in the message header contains the time the DSN was sent by the gateway and the DSN Last-Attempt-Date field contains the time the last delivery attempt occurred.

```
last-attempt-date-field = "Last-Attempt-Date" ":" date-time
```

This field is optional. It MUST NOT be included if the actual date and time of the last delivery attempt are not available (which might be the case if the DSN were being issued by a gateway).

The date and time are expressed in RFC 822 'date-time' format, as modified by [8]. Numeric timezones ([+/-]HHMM format) MUST be used.

3.2.1.5 final-log-id field

The "final-log-id" field gives the final-log-id of the message that was used by the final-mta. This can be useful as an index to the final-mta's log entry for that delivery attempt.

```
final-log-id-field = "Final-Log-ID" ":" *text
```

This field is optional.

2.3.8 Will-Retry-Until field

For DSNs of type "delayed", the Will-Retry-Until field gives the date after which the Reporting MTA expects to abandon all attempts to deliver the message to that recipient. The Will-Retry-Until field is optional for "delay" DSNs, and MUST NOT appear in other DSNs.

```
will-retry-until-field = "Will-Retry-Until" ":" date-time
```

The date and time are expressed in RFC 822 'date-time' format, as modified by [8]. Numeric timezones ([+/-]HHMM format) MUST be used.

## 2.4 Extension fields

Additional per-message or per-recipient DSN fields may be defined in the future by later revisions or extensions to this specification. Extension-field names beginning with "X-" will never be defined as standard fields; such names are reserved for experimental use. DSN field names NOT beginning with "X-" MUST be registered with the Internet Assigned Numbers Authority (IANA) and published in an RFC.

Extension DSN fields may be defined for the following reasons:

(a) To allow additional information from foreign delivery status reports to be tunneled through Internet DSNs. The names of such DSN fields should begin with an indication of the foreign environment name (e.g. X400-Physical-Forwarding-Address).

(b) To allow the transmission of diagnostic information which is specific to a particular mail transport protocol. The names of such DSN fields should begin with an indication of the mail transport being used (e.g. SMTP-Remote-Recipient-Address). Such fields should be used for diagnostic purposes only and not by user agents or mail gateways.

(c) To allow transmission of diagnostic information which is specific to a particular message transfer agent (MTA). The names of such DSN fields should begin with an indication of the MTA implementation which produced the DSN. (e.g. Foomail-Queue-ID).

MTA implementors are encouraged to provide adequate information, via extension fields if necessary, to allow an MTA maintainer to understand the nature of correctable delivery failures and how to fix them. For example, if message delivery attempts are logged, the DSN might include information which allows the MTA maintainer to easily find the log entry for a failed delivery attempt.

If an MTA developer does not wish to register the meanings of such extension fields, "X-" fields may be used for this purpose. To avoid name collisions, the name of the MTA implementation should follow the "X-", (e.g. "X-Foomail-Log-ID").

## 3. Conformance and Usage Requirements

An MTA or gateway conforms to this specification if it generates DSNs according to the protocol defined in this memo. For MTAs and gateways that do not support requests for positive delivery notification (such as in [4]), it is sufficient that delivery failure reports use this protocol.





A minimal implementation of this specification need generate only the Reporting-MTA per-message field, and the Final-Recipient, Action, and Status fields for each attempt to deliver a message to a recipient described by the DSN. Generation of the other fields, when appropriate, is strongly recommended.

MTAs and gateways MUST NOT generate the Original-Recipient field of a DSN unless the mail transfer protocol provides the address originally specified by the sender at the time of submission. (Ordinary SMTP does not make that guarantee, but the SMTP extension defined in [4] permits such information to be carried in the envelope if it is available.)

Each sender-specified recipient address SHOULD result in at most one "delivered" or "failed" DSN for that recipient. If a positive DSN is requested (e.g. one using NOTIFY=SUCCESS in SMTP) for a recipient that is forwarded to multiple recipients of an "alias" (as defined in [4], section 7.2.7), the forwarding MTA SHOULD normally issue a "expanded" DSN for the originally-specified recipient and not propagate the request for a DSN to the forwarding addresses. Alternatively, the forwarding MTA MAY relay the request for a DSN to exactly one of the forwarding addresses and not propagate the request to the others.

By contrast, successful submission of a message to a mailing list exploder is considered final delivery of the message. Upon delivery of a message to a recipient address corresponding to a mailing list exploder, the Reporting MTA SHOULD issue an appropriate DSN exactly as if the recipient address were that of an ordinary mailbox.

NOTE: This is actually intended to make DSNs usable by mailing lists themselves. Any message sent to a mailing list subscriber should have its envelope return address pointing to the list maintainer [see RFC 1123, section 5.3.7(E)]. Since DSNs are sent to the envelope return address, all DSNs resulting from delivery to the recipients of a mailing list will be sent to the list maintainer. The list maintainer may elect to mechanically process DSNs upon receipt, and thus automatically delete invalid addresses from the list. (See section 7 of this memo.)

This specification places no restrictions on the processing of DSNs received by user agents or distribution lists.

## 4. Security Considerations

The following security considerations apply when using DSNs:

### 4.1 Forgery

DSNs may be forged as easily as ordinary Internet electronic mail. User agents and automatic mail handling facilities (such as mail distribution list exploders) that wish to make automatic use of DSNs should take appropriate precautions to minimize the potential damage from denial-of-service attacks.

Security threats related to forged DSNs include the sending of:

(a) A falsified delivery notification when the message is not delivered to the indicated recipient,
(b) A falsified non-delivery notification when the message was in fact delivered to the indicated recipient,
(c) A falsified Final-Recipient address,
(d) A falsified Remote-MTA identification,
(e) A falsified relay notification when the message is "dead ended".
(f) Unsolicited DSNs

### 4.2 Confidentiality

Another dimension of security is confidentiality. There may be cases in which a message recipient is autoforwarding messages but does not wish to divulge the address to which the messages are autoforwarded. The desire for such confidentiality will probably be heightened as "wireless mailboxes", such as pagers, become more widely used as autoforward addresses.

MTA authors are encouraged to provide a mechanism which enables the end user to preserve the confidentiality of a forwarding address. Depending on the degree of confidentiality required, and the nature of the environment to which a message were being forwarded, this might be accomplished by one or more of:

(a) issuing a "relayed" DSN (if a positive DSN was requested) when a message is forwarded to a confidential forwarding address, and disabling requests for positive DSNs for the forwarded message,

(b) declaring the message to be delivered, issuing a "delivered" DSN, re-sending the message to the confidential forwarding address, and arranging for no DSNs to be issued for the re-sent message,

(c) omitting "Remote-*" or extension fields of a DSN whenever they would otherwise contain confidential information (such as a confidential forwarding address),

(d) for messages forwarded to a confidential address, setting the envelope return address (e.g. SMTP MAIL FROM address) to the NULL reverse-path ("<>") (so that no DSNs would be sent from a downstream MTA to the original sender),

(e) for messages forwarded to a confidential address, disabling delivery notifications for the forwarded message (e.g. if the "next-hop" MTA uses ESMTP and supports the DSN extension, by using the NOTIFY=NEVER parameter to the RCPT command), or

(f) when forwarding mail to a confidential address, having the forwarding MTA rewrite the envelope return address for the forwarded message and attempt delivery of that message as if the forwarding MTA were the originator. On its receipt of final delivery status, the forwarding MTA would issue a DSN to the original sender.

In general, any optional DSN field may be omitted if the Reporting MTA site determines that inclusion of the field would impose too great a compromise of site confidentiality. The need for such confidentiality must be balanced against the utility of the omitted information in trouble reports and DSNs gatewayed to foreign environments.

Implementors are cautioned that many existing MTAs will send nondelivery notifications to a return address in the message header (rather than to the one in the envelope), in violation of SMTP and other protocols. If a message is forwarded through such an MTA, no reasonable action on the part of the forwarding MTA will prevent the downstream MTA from compromising the forwarding address. Likewise, if the recipient's MTA automatically responds to messages based on a request in the message header (such as the nonstandard, but widely used, Return-Receipt-To extension header), it will also compromise the forwarding address.

### 4.3 Non-Repudiation

Within the framework of today's internet mail, the DSNs defined in this memo provide valuable information to the mail user; however, even a "failed" DSN can not be relied upon as a guarantee that a message was not received by the recipient. Even if DSNs are not actively forged, conditions exist under which a message can be delivered despite the fact that a failure DSN was issued.

For example, a race condition in the SMTP protocol allows for the duplication of messages if the connection is dropped following a completed DATA command, but before a response is seen by the SMTP client. This will cause the SMTP client to retransmit the message, even though the SMTP server has already accepted it.[9] If one of those delivery attempts succeeds and the other one fails, a "failed" DSN could be issued even though the message actually reached the recipient.







## 5. Appendix - collected grammar

NOTE: The following lexical tokens are defined in RFC 822: atom, CHAR, comment, CR, CRLF, DIGIT, LF, linear-white-space, SPACE, text. The date-time lexical token is defined in [8].

```
action-field = "Action" ":" action-value

action-value =
     "failed" / "delayed" / "delivered" / "relayed" / "expanded"

address-type = atom

arrival-date-field = "Arrival-Date" ":" date-time

delivery-status-content =
     per-message-fields 1*( CRLF per-recipient-fields )

diagnostic-code-field =
     "Diagnostic-Code" ":" diagnostic-type ";" *text

diagnostic-type = atom

dsn-gateway-field = "DSN-Gateway" ":" mta-name-type ";" mta-name

envelope-id = *text

extension-field = extension-field-name ":" *text

extension-field-name = atom

final-recipient-field =
     "Final-Recipient" ":" address-type ";" generic-address

generic-address = *text

last-attempt-date-field = "Last-Attempt-Date" ":" date-time

mta-name = *text

mta-name-type = atom

original-envelope-id-field =
     "Original-Envelope-Id" ":" envelope-id

original-recipient-field =
     "Original-Recipient" ":" address-type ";" generic-address
```

```
per-message-fields =
     [ original-envelope-id-field CRLF ]
     reporting-mta-field CRLF
     [ dsn-gateway-field CRLF ]
     [ received-from-mta-field CRLF ]
     [ arrival-date-field CRLF ]
     *( extension-field CRLF )

per-recipient-fields =
     [ original-recipient-field CRLF ]
     final-recipient-field CRLF
     action-field CRLF
     status-field CRLF
     [ remote-mta-field CRLF ]
     [ diagnostic-code-field CRLF ]
     [ last-attempt-date-field CRLF ]
     [ will-retry-until-field CRLF ]
     *( extension-field CRLF )

received-from-mta-field =
     "Received-From-MTA" ":" mta-name-type ";" mta-name

remote-mta-field = "Remote-MTA" ":" mta-name-type ";" mta-name

reporting-mta-field =
     "Reporting-MTA" ":" mta-name-type ";" mta-name

status-code = DIGIT "." 1*3DIGIT "." 1*3DIGIT

     ; White-space characters and comments are NOT allowed within a
     ; status-code, though a comment enclosed in parentheses MAY follow
     ; the last numeric subfield of the status-code.  Each numeric
     ; subfield within the status-code MUST be expressed without
     ; leading zero digits.

status-field = "Status" ":" status-code

will-retry-until-field = "Will-Retry-Until" ":" date-time
```







## 6. Appendix - Guidelines for gatewaying DSNs

NOTE: This section provides non-binding recommendations for the construction of mail gateways that wish to provide semi-transparent delivery reports between the Internet and another electronic mail system. Specific DSN gateway requirements for a particular pair of mail systems may be defined by other documents.

### 6.1 Gatewaying from other mail systems to DSNs

A mail gateway may issue a DSN to convey the contents of a "foreign" delivery or non-delivery notification over Internet mail. When there are appropriate mappings from the foreign notification elements to DSN fields, the information may be transmitted in those DSN fields. Additional information (such as might be useful in a trouble ticket or needed to tunnel the foreign notification through the Internet) may be defined in extension DSN fields. (Such fields should be given names that identify the foreign mail protocol, e.g. X400-* for X.400 NDN or DN protocol elements)

The gateway must attempt to supply reasonable values for the Reporting-MTA, Final-Recipient, Action, and Status fields. These will normally be obtained by translating the values from the remote delivery or non-delivery notification into their Internet-style equivalents. However, some loss of information is to be expected. For example, the set of status-codes defined for DSNs may not be adequate to fully convey the delivery diagnostic code from the foreign system. The gateway should assign the most precise code which describes the failure condition, falling back on "generic" codes such as 2.0.0 (success), 4.0.0 (temporary failure), and 5.0.0 (permanent failure) when necessary. The actual foreign diagnostic code should be retained in the Diagnostic-Code field (with an appropriate diagnostic-type value) for use in trouble tickets or tunneling.

The sender-specified recipient address, and the original envelope-id, if present in the foreign transport envelope, should be preserved in the Original-Recipient and Original-Envelope-ID fields.

The gateway should also attempt to preserve the "final" recipient addresses and MTA names from the foreign system. Whenever possible, foreign protocol elements should be encoded as meaningful printable ASCII strings.

For DSNs produced from foreign delivery or nondelivery notifications, the name of the gateway MUST appear in the DSN-Gateway field of the DSN.

### 6.2 Gatewaying from DSNs to other mail systems

It may be possible to gateway DSNs from the Internet into a foreign mail system. The primary purpose of such gatewaying is to convey delivery status information in a form that is usable by the destination system. A secondary purpose is to allow "tunneling" of DSNs through foreign mail systems, in case the DSN may be gatewayed back into the Internet.

In general, the recipient of the DSN (i.e., the sender of the original message) will want to know, for each recipient: the closest available approximation to the original recipient address, the delivery status (success, failure, or temporary failure), and for failed deliveries, a diagnostic code that describes the reason for the failure.

If possible, the gateway should attempt to preserve the Original-Recipient address and Original-Envelope-ID (if present), in the resulting foreign delivery status report.

When reporting delivery failures, if the diagnostic-type subfield of the Diagnostic-Code field indicates that the original diagnostic code is understood by the destination environment, the information from the Diagnostic-Code field should be used. Failing that, the information in the Status field should be mapped into the closest available diagnostic code used in the destination environment.

If it is possible to tunnel a DSN through the destination environment, the gateway specification may define a means of preserving the DSN information in the delivery status reports used by that environment.







## 7. Appendix - Guidelines for use of DSNs by mailing list exploders

NOTE: This section pertains only to the use of DSNs by "mailing lists" as defined in [4], section 7.2.7.

DSNs are designed to be used by mailing list exploders to allow them to detect and automatically delete recipients for whom mail delivery fails repeatedly.

When forwarding a message to list subscribers, the mailing list exploder should always set the envelope return address (e.g. SMTP MAIL FROM address) to point to a special address which is set up to received nondelivery reports. A "smart" mailing list exploder can therefore intercept such nondelivery reports, and if they are in the DSN format, automatically examine them to determine for which recipients a message delivery failed or was delayed.

The Original-Recipient field should be used if available, since it should exactly match the subscriber address known to the list. If the Original-Recipient field is not available, the recipient field may resemble the list subscriber address. Often, however, the list subscriber will have forwarded his mail to a different address, or the address may be subject to some re-writing, so heuristics may be required to successfully match an address from the recipient field. Care is needed in this case to minimize the possibility of false matches.

The reason for delivery failure can be obtained from the Status and Action fields, and from the Diagnostic-Code field (if the status-type is recognized). Reports for recipients with action values other than "failed" can generally be ignored; in particular, subscribers should not be removed from a list due to "delayed" reports.

In general, almost any failure status code (even a "permanent" one) can result from a temporary condition. It is therefore recommended that a list exploder not delete a subscriber based on any single failure DSN (regardless of the status code), but only on the persistence of delivery failure over a period of time.

However, some kinds of failures are less likely than others to have been caused by temporary conditions, and some kinds of failures are more likely to be noticed and corrected quickly than others. Once more precise status codes are defined, it may be useful to differentiate between the status codes when deciding whether to delete a subscriber. For example, on a list with a high message volume, it might be desirable to temporarily suspend delivery to a recipient address which causes repeated "temporary" failures, rather than simply deleting the recipient. The duration of the suspension

might depend on the type of error. On the other hand, a "user unknown" error which persisted for several days could be considered a reliable indication that address were no longer valid.

## 8. Appendix - IANA registration forms for DSN types

The forms below are for use when registering a new address-type, diagnostic-type, or MTA-name-type with the Internet Assigned Numbers Authority (IANA). Each piece of information requested by a registration form may be satisfied either by providing the information on the form itself, or by including a reference to a published, publicly available specification which includes the necessary information. IANA MAY reject DSN type registrations because of incomplete registration forms, imprecise specifications, or inappropriate type names.

To register a DSN type, complete the applicable form below and send it via Internet electronic mail to <IANA@IANA.ORG>.

### 8.1 IANA registration form for address-type

A registration for a DSN address-type MUST include the following information:

(a) The proposed address-type name.

(b) The syntax for mailbox addresses of this type, specified using BNF, regular expressions, ASN.1, or other non-ambiguous language.

(c) If addresses of this type are not composed entirely of graphic characters from the US-ASCII repertoire, a specification for how they are to be encoded as graphic US-ASCII characters in a DSN Original-Recipient or Final-Recipient DSN field.

(d) [optional] A specification for how addresses of this type are to be translated to and from Internet electronic mail addresses.

### 8.2 IANA registration form for diagnostic-type

A registration for a DSN address-type MUST include the following information:

(a) The proposed diagnostic-type name.

(b) A description of the syntax to be used for expressing diagnostic codes of this type as graphic characters from the US-ASCII repertoire.





(c) A list of valid diagnostic codes of this type and the meaning of each code.

(d) [optional] A specification for mapping from diagnostic codes of this type to DSN status codes (as defined in [5]).

### 8.3 IANA registration form for MTA-name-type

A registration for a DSN MTA-name-type must include the following information:

(a) The proposed MTA-name-type name.

(b) A description of the syntax of MTA names of this type, using BNF, regular expressions, ASN.1, or other non-ambiguous language.

(c) If MTA names of this type do not consist entirely of graphic characters from the US-ASCII repertoire, a specification for how an MTA name of this type should be expressed as a sequence of graphic US-ASCII characters.

## 9. Appendix - Examples

NOTE: These examples are provided as illustration only, and are not considered part of the DSN protocol specification. If an example conflicts with the protocol definition above, the example is wrong.

Likewise, the use of *-type subfield names or extension fields in these examples is not to be construed as a definition for those type names or extension fields.

These examples were manually translated from bounced messages using whatever information was available.







### 9.1 This is a simple DSN issued after repeated attempts to deliver a message failed. In this case, the DSN is issued by the same MTA from which the message was originated.

```
Date: Thu, 7 Jul 1994 17:16:05 -0400
From: Mail Delivery Subsystem <MAILER-DAEMON@CS.UTK.EDU>
Message-Id: <199407072116.RAA14128@CS.UTK.EDU>
Subject: Returned mail: Cannot send message for 5 days
To: <owner-info-mime@cs.utk.edu>
MIME-Version: 1.0
Content-Type: multipart/report; report-type=delivery-status;
      boundary="RAA14128.773615765/CS.UTK.EDU"

--RAA14128.773615765/CS.UTK.EDU

The original message was received at Sat, 2 Jul 1994 17:10:28 -0400
from root@localhost

   ----- The following addresses had delivery problems -----
<louisl@larry.slip.umd.edu>  (unrecoverable error)

   ----- Transcript of session follows -----
<louisl@larry.slip.umd.edu>... Deferred: Connection timed out
      with larry.slip.umd.edu.
Message could not be delivered for 5 days
Message will be deleted from queue

--RAA14128.773615765/CS.UTK.EDU
content-type: message/delivery-status

Reporting-MTA: dns; cs.utk.edu

Original-Recipient: rfc822;louisl@larry.slip.umd.edu
Final-Recipient: rfc822;louisl@larry.slip.umd.edu
Action: failed
Status: 4.0.0
Diagnostic-Code: smtp; 426 connection timed out
Last-Attempt-Date: Thu, 7 Jul 1994 17:15:49 -0400

--RAA14128.773615765/CS.UTK.EDU
content-type: message/rfc822

[original message goes here]
--RAA14128.773615765/CS.UTK.EDU--
```

9.2 This is another DSN issued by the sender's MTA, which contains details of multiple delivery attempts. Some of these were detected locally, and others by a remote MTA.

```
Date: Fri, 8 Jul 1994 09:21:47 -0400
From: Mail Delivery Subsystem <MAILER-DAEMON@CS.UTK.EDU>
Subject: Returned mail: User unknown
To: <owner-ups-mib@CS.UTK.EDU>
MIME-Version: 1.0
Content-Type: multipart/report; report-type=delivery-status;
      boundary="JAA13167.773673707/CS.UTK.EDU"

--JAA13167.773673707/CS.UTK.EDU
content-type: text/plain; charset=us-ascii

   ----- The following addresses had delivery problems -----
<arathib@vnet.ibm.com>  (unrecoverable error)
<wsnell@sdcc13.ucsd.edu>  (unrecoverable error)

--JAA13167.773673707/CS.UTK.EDU
content-type: message/delivery-status

Reporting-MTA: dns; cs.utk.edu

Original-Recipient: rfc822;arathib@vnet.ibm.com
Final-Recipient: rfc822;arathib@vnet.ibm.com
Action: failed
Status: 5.0.0 (permanent failure)
Diagnostic-Code: smtp;
 550 'arathib@vnet.IBM.COM' is not a registered gateway user
Remote-MTA: dns; vnet.ibm.com

Original-Recipient: rfc822;johnh@hpnjld.njd.hp.com
Final-Recipient: rfc822;johnh@hpnjld.njd.hp.com
Action: delayed
Status: 4.0.0 (hpnjld.njd.jp.com: host name lookup failure)

Original-Recipient: rfc822;wsnell@sdcc13.ucsd.edu
Final-Recipient: rfc822;wsnell@sdcc13.ucsd.edu
Action: failed
Status: 5.0.0
Diagnostic-Code: smtp; 550 user unknown
Remote-MTA: dns; sdcc13.ucsd.edu

--JAA13167.773673707/CS.UTK.EDU
content-type: message/rfc822

[original message goes here]
--JAA13167.773673707/CS.UTK.EDU--
```

9.3 A delivery report generated by Message Router (MAILBUS) and gatewayed by PMDF_MR to a DSN. In this case the gateway did not have sufficient information to supply an original-recipient address.

```
Disclose-recipients: prohibited
Date: Fri, 08 Jul 1994 09:21:25 -0400 (EDT)
From: Message Router Submission Agent <AMMGR@corp.timeplex.com>
Subject: Status of : Re: Battery current sense
To: owner-ups-mib@CS.UTK.EDU
Message-id: <01HEGJ0WNBY28Y95LN@mr.timeplex.com>
MIME-version: 1.0
content-type: multipart/report; report-type=delivery-status;
      boundary="84229080704991.122306.SYS30"

--84229080704991.122306.SYS30
content-type: text/plain

Invalid address - nair_s
%DIR-E-NODIRMTCH, No matching Directory Entry found

--84229080704991.122306.SYS30
content-type: message/delivery-status

Reporting-MTA: mailbus; SYS30

Final-Recipient: unknown; nair_s
Status: 5.0.0 (unknown permanent failure)
Action: failed

--84229080704991.122306.SYS30--
```

9.4 A delay report from a multiprotocol MTA. Note that there is no returned content, so no third body part appears in the DSN.

```
From: <postmaster@nsfnet-relay.ac.uk>
Message-Id: <199407092338.TAA23293@CS.UTK.EDU>
Received: from nsfnet-relay.ac.uk by sun2.nsfnet-relay.ac.uk
          id <g.12954-0@sun2.nsfnet-relay.ac.uk>;
Sun, 10 Jul 1994 00:36:51 +0100
To: owner-info-mime@cs.utk.edu
Date: Sun, 10 Jul 1994 00:36:51 +0100
Subject: WARNING: message delayed at "nsfnet-relay.ac.uk"
content-type: multipart/report; report-type=delivery-status;
      boundary=foobar

--foobar
content-type: text/plain

The following message:

UA-ID:  Reliable PC (...
Q-ID:   sun2.nsf:77/msg.11820-0

has not been delivered to the intended recipient:

thomas@de-montfort.ac.uk

despite repeated delivery attempts over the past 24 hours.

The  usual cause of this problem is that the remote system is
temporarily unavailable.

Delivery will continue to be attempted up to a total elapsed
time of  168 hours, ie 7 days.

You  will  be  informed  if  delivery proves to be impossible
within this time.

Please quote the Q-ID in any queries regarding this mail.

--foobar
content-type: message/delivery-status

Reporting-MTA: dns; sun2.nsfnet-relay.ac.uk

Final-Recipient: rfc822;thomas@de-montfort.ac.uk
Status: 4.0.0 (unknown temporary failure)
Action: delayed
```





```
--foobar--
```

## 10. Acknowledgments

The authors wish to thank the following people for their reviews of earlier drafts of this document and their suggestions for improvement: Eric Allman, Harald Alvestrand, Allan Cargille, Jim Conklin, Peter Cowen, Dave Crocker, Roger Fajman, Ned Freed, Marko Kaittola, Steve Kille, John Klensin, John Gardiner Myers, Mark Nahabedian, Julian Onions, Jacob Palme, Jean Charles Roy, and Gregory Sheehan.

## 11. References

[1] Borenstein, N., Freed, N. "Multipurpose Internet Mail Extensions", RFC 1521, Bellcore, Innosoft, September 1993.

[2] Vaudreuil, G., "The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages", RFC 1892, Octal Network Services, January 1996.

[3] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[4] Moore, K., "SMTP Service Extension for Delivery Status Notifications", RFC 1891, University of Tennessee, January 1996.

[5] Vaudreuil, G., "Enhanced Mail System Status Codes", RFC 1893, Octal Network Services, January 1996.

[6] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[7] Moore, K. "MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-Ascii Text", RFC 1522, University of Tennessee, September 1993.

[8] Braden, R. (ed.) "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, USC/Information Sciences Institute, October 1989.

[9] Partridge, C., "Duplicate Messages and SMTP", RFC 1047, BBN, February 1988.

## 11. Authors' Addresses

Keith Moore
University of Tennessee
107 Ayres Hall
Knoxville, TN 37996-1301
USA

EMail: moore@cs.utk.edu
Phone: +1 615 974 3126
Fax: +1 615 974 8296

Gregory M. Vaudreuil
Octel Network Services
17080 Dallas Parkway
Dallas, TX 75248-1905
USA

EMail: Greg.Vaudreuil@Octel.Com

Network Working Group J. Myers
Request for Comments: 1939 Carnegie Mellon
STD: 53 M. Rose
Obsoletes: 1725 Dover Beach Consulting, Inc.
Category: Standards Track May 1996

# Post Office Protocol - Version 3

## Status of this Memo



## Table of Contents









## 1. Introduction

On certain types of smaller nodes in the Internet it is often impractical to maintain a message transport system (MTS). For example, a workstation may not have sufficient resources (cycles, disk space) in order to permit a SMTP server [RFC821] and associated local mail delivery system to be kept resident and continuously running. Similarly, it may be expensive (or impossible) to keep a personal computer interconnected to an IP-style network for long amounts of time (the node is lacking the resource known as "connectivity").

Despite this, it is often very useful to be able to manage mail on these smaller nodes, and they often support a user agent (UA) to aid the tasks of mail handling. To solve this problem, a node which can support an MTS entity offers a maildrop service to these less endowed nodes. The Post Office Protocol - Version 3 (POP3) is intended to permit a workstation to dynamically access a maildrop on a server host in a useful fashion. Usually, this means that the POP3 protocol is used to allow a workstation to retrieve mail that the server is holding for it.

POP3 is not intended to provide extensive manipulation operations of mail on the server; normally, mail is downloaded and then deleted. A more advanced (and complex) protocol, IMAP4, is discussed in [RFC1730].

For the remainder of this memo, the term "client host" refers to a host making use of the POP3 service, while the term "server host" refers to a host which offers the POP3 service.

## 2. A Short Digression

This memo does not specify how a client host enters mail into the transport system, although a method consistent with the philosophy of this memo is presented here:

> When the user agent on a client host wishes to enter a message into the transport system, it establishes an SMTP connection to its relay host and sends all mail to it. This relay host could be, but need not be, the POP3 server host for the client host. Of course, the relay host must accept mail for delivery to arbitrary recipient addresses, that functionality is not required of all SMTP servers.

## 3. Basic Operation

Initially, the server host starts the POP3 service by listening on TCP port 110. When a client host wishes to make use of the service, it establishes a TCP connection with the server host. When the connection is established, the POP3 server sends a greeting. The client and POP3 server then exchange commands and responses (respectively) until the connection is closed or aborted.

Commands in the POP3 consist of a case-insensitive keyword, possibly followed by one or more arguments. All commands are terminated by a CRLF pair. Keywords and arguments consist of printable ASCII characters. Keywords and arguments are each separated by a single SPACE character. Keywords are three or four characters long. Each argument may be up to 40 characters long.

Responses in the POP3 consist of a status indicator and a keyword possibly followed by additional information. All responses are terminated by a CRLF pair. Responses may be up to 512 characters long, including the terminating CRLF. There are currently two status indicators: positive ("+OK") and negative ("-ERR"). Servers MUST send the "+OK" and "-ERR" in upper case.

Responses to certain commands are multi-line. In these cases, which are clearly indicated below, after sending the first line of the response and a CRLF, any additional lines are sent, each terminated by a CRLF pair. When all lines of the response have been sent, a final line is sent, consisting of a termination octet (decimal code 046, ".") and a CRLF pair. If any line of the multi-line response begins with the termination octet, the line is "byte-stuffed" by pre-pending the termination octet to that line of the response. Hence a multi-line response is terminated with the five octets "CRLF.CRLF". When examining a multi-line response, the client checks to see if the line begins with the termination octet. If so and if octets other than CRLF follow, the first octet of the line (the termination octet) is stripped away. If so and if CRLF immediately follows the termination character, then the response from the POP server is ended and the line containing ".CRLF" is not considered part of the multi-line response.

A POP3 session progresses through a number of states during its lifetime. Once the TCP connection has been opened and the POP3 server has sent the greeting, the session enters the AUTHORIZATION state. In this state, the client must identify itself to the POP3 server. Once the client has successfully done this, the server acquires resources associated with the client's maildrop, and the session enters the TRANSACTION state. In this state, the client requests actions on the part of the POP3 server. When the client has





issued the QUIT command, the session enters the UPDATE state. In this state, the POP3 server releases any resources acquired during the TRANSACTION state and says goodbye. The TCP connection is then closed.

A server MUST respond to an unrecognized, unimplemented, or syntactically invalid command by responding with a negative status indicator. A server MUST respond to a command issued when the session is in an incorrect state by responding with a negative status indicator. There is no general method for a client to distinguish between a server which does not implement an optional command and a server which is unwilling or unable to process the command.

A POP3 server MAY have an inactivity autologout timer. Such a timer MUST be of at least 10 minutes' duration. The receipt of any command from the client during that interval should suffice to reset the autologout timer. When the timer expires, the session does NOT enter the UPDATE state--the server should close the TCP connection without removing any messages or sending any response to the client.

## 4. The AUTHORIZATION State

Once the TCP connection has been opened by a POP3 client, the POP3 server issues a one line greeting. This can be any positive response. An example might be:

```
   S:  +OK POP3 server ready
```

The POP3 session is now in the AUTHORIZATION state. The client must now identify and authenticate itself to the POP3 server. Two possible mechanisms for doing this are described in this document, the USER and PASS command combination and the APOP command. Both mechanisms are described later in this document. Additional authentication mechanisms are described in [RFC1734]. While there is no single authentication mechanism that is required of all POP3 servers, a POP3 server must of course support at least one authentication mechanism.

Once the POP3 server has determined through the use of any authentication command that the client should be given access to the appropriate maildrop, the POP3 server then acquires an exclusive-access lock on the maildrop, as necessary to prevent messages from being modified or removed before the session enters the UPDATE state. If the lock is successfully acquired, the POP3 server responds with a positive status indicator. The POP3 session now enters the TRANSACTION state, with no messages marked as deleted. If the maildrop cannot be opened for some reason (for example, a lock can not be acquired, the client is denied access to the appropriate

maildrop, or the maildrop cannot be parsed), the POP3 server responds with a negative status indicator. (If a lock was acquired but the POP3 server intends to respond with a negative status indicator, the POP3 server must release the lock prior to rejecting the command.) After returning a negative status indicator, the server may close the connection. If the server does not close the connection, the client may either issue a new authentication command and start again, or the client may issue the QUIT command.

After the POP3 server has opened the maildrop, it assigns a message-number to each message, and notes the size of each message in octets. The first message in the maildrop is assigned a message-number of "1", the second is assigned "2", and so on, so that the nth message in a maildrop is assigned a message-number of "n". In POP3 commands and responses, all message-numbers and message sizes are expressed in base-10 (i.e., decimal).

Here is the summary for the QUIT command when used in the AUTHORIZATION state:

```
   QUIT

      Arguments: none

      Restrictions: none

      Possible Responses:
          +OK

      Examples:
          C: QUIT
          S: +OK dewey POP3 server signing off
```

## 5. The TRANSACTION State

Once the client has successfully identified itself to the POP3 server and the POP3 server has locked and opened the appropriate maildrop, the POP3 session is now in the TRANSACTION state. The client may now issue any of the following POP3 commands repeatedly. After each command, the POP3 server issues a response. Eventually, the client issues the QUIT command and the POP3 session enters the UPDATE state.





Here are the POP3 commands valid in the TRANSACTION state:

```
   STAT

      Arguments: none

      Restrictions:
          may only be given in the TRANSACTION state

      Discussion:
          The POP3 server issues a positive response with a line
          containing information for the maildrop.  This line is
          called a "drop listing" for that maildrop.

          In order to simplify parsing, all POP3 servers are
          required to use a certain format for drop listings.  The
          positive response consists of "+OK" followed by a single
          space, the number of messages in the maildrop, a single
          space, and the size of the maildrop in octets.  This memo
          makes no requirement on what follows the maildrop size.
          Minimal implementations should just end that line of the
          response with a CRLF pair.  More advanced implementations
          may include other information.

             NOTE: This memo STRONGLY discourages implementations
             from supplying additional information in the drop
             listing.  Other, optional, facilities are discussed
             later on which permit the client to parse the messages
             in the maildrop.

          Note that messages marked as deleted are not counted in
          either total.

      Possible Responses:
          +OK nn mm

      Examples:
          C: STAT
          S: +OK 2 320


   LIST [msg]

      Arguments:
          a message-number (optional), which, if present, may NOT
          refer to a message marked as deleted
```

Restrictions:
may only be given in the TRANSACTION state

Discussion:
If an argument was given and the POP3 server issues a positive response with a line containing information for that message. This line is called a "scan listing" for that message.

If no argument was given and the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, for each message in the maildrop, the POP3 server responds with a line containing information for that message. This line is also called a "scan listing" for that message. If there are no messages in the maildrop, then the POP3 server responds with no scan listings--it issues a positive response followed by a line containing a termination octet and a CRLF pair.

In order to simplify parsing, all POP3 servers are required to use a certain format for scan listings. A scan listing consists of the message-number of the message, followed by a single space and the exact size of the message in octets. Methods for calculating the exact size of the message are described in the "Message Format" section below. This memo makes no requirement on what follows the message size in the scan listing. Minimal implementations should just end that line of the response with a CRLF pair. More advanced implementations may include other information, as parsed from the message.

NOTE: This memo STRONGLY discourages implementations from supplying additional information in the scan listing. Other, optional, facilities are discussed later on which permit the client to parse the messages in the maildrop.

Note that messages marked as deleted are not listed.

Possible Responses:
+OK scan listing follows
-ERR no such message

Examples:
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
...
C: LIST 2
S: +OK 2 200
...
C: LIST 3
S: -ERR no such message, only 2 messages in maildrop





RETR msg

Arguments:
a message-number (required) which may NOT refer to a message marked as deleted

Restrictions:
may only be given in the TRANSACTION state

Discussion:
If the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, the POP3 server sends the message corresponding to the given message-number, being careful to byte-stuff the termination character (as with all multi-line responses).

Possible Responses:
+OK message follows
-ERR no such message

Examples:
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends the entire message here>
S: .

DELE msg

Arguments:
a message-number (required) which may NOT refer to a message marked as deleted

Restrictions:
may only be given in the TRANSACTION state

Discussion:
The POP3 server marks the message as deleted. Any future reference to the message-number associated with the message in a POP3 command generates an error. The POP3 server does not actually delete the message until the POP3 session enters the UPDATE state.

Possible Responses:
+OK message deleted
-ERR no such message

Examples:
C: DELE 1
S: +OK message 1 deleted
...
C: DELE 2
S: -ERR message 2 already deleted

NOOP

Arguments: none

Restrictions:
may only be given in the TRANSACTION state

Discussion:
The POP3 server does nothing, it merely replies with a positive response.

Possible Responses:
+OK

Examples:
C: NOOP
S: +OK

RSET

Arguments: none

Restrictions:
may only be given in the TRANSACTION state

Discussion:
If any messages have been marked as deleted by the POP3 server, they are unmarked. The POP3 server then replies





with a positive response.

Possible Responses:
+OK

Examples:
C: RSET
S: +OK maildrop has 2 messages (320 octets)

## 6. The UPDATE State

When the client issues the QUIT command from the TRANSACTION state, the POP3 session enters the UPDATE state. (Note that if the client issues the QUIT command from the AUTHORIZATION state, the POP3 session terminates but does NOT enter the UPDATE state.)

If a session terminates for some reason other than a client-issued QUIT command, the POP3 session does NOT enter the UPDATE state and MUST not remove any messages from the maildrop.

QUIT

Arguments: none

Restrictions: none

Discussion:
The POP3 server removes all messages marked as deleted from the maildrop and replies as to the status of this operation. If there is an error, such as a resource shortage, encountered while removing messages, the maildrop may result in having some or none of the messages marked as deleted be removed. In no case may the server remove any messages not marked as deleted.

Whether the removal was successful or not, the server then releases any exclusive-access lock on the maildrop and closes the TCP connection.

Possible Responses:
+OK
-ERR some deleted messages not removed

Examples:
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
...
C: QUIT

```
            S: +OK dewey POP3 server signing off (2 messages left)
               ...
```

## 7. Optional POP3 Commands

The POP3 commands discussed above must be supported by all minimal implementations of POP3 servers.

The optional POP3 commands described below permit a POP3 client greater freedom in message handling, while preserving a simple POP3 server implementation.

NOTE: This memo STRONGLY encourages implementations to support these commands in lieu of developing augmented drop and scan listings. In short, the philosophy of this memo is to put intelligence in the part of the POP3 client and not the POP3 server.

TOP msg n

Arguments:
a message-number (required) which may NOT refer to to a message marked as deleted, and a non-negative number of lines (required)

Restrictions:
may only be given in the TRANSACTION state

Discussion:
If the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, the POP3 server sends the headers of the message, the blank line separating the headers from the body, and then the number of lines of the indicated message's body, being careful to byte-stuff the termination character (as with all multi-line responses).

Note that if the number of lines requested by the POP3 client is greater than than the number of lines in the body, then the POP3 server sends the entire message.

Possible Responses:
+OK top of message follows
-ERR no such message

Examples:

```
C: TOP 1 10
S: +OK
   <the POP3 server sends the headers of the
   message, a blank line, and the first 10 lines
   of the body of the message>
S: .
   ...
C: TOP 100 3
S: -ERR no such message
```





UIDL [msg]

Arguments:
a message-number (optional), which, if present, may NOT refer to a message marked as deleted

Restrictions:
may only be given in the TRANSACTION state.

Discussion:
If an argument was given and the POP3 server issues a positive response with a line containing information for that message. This line is called a "unique-id listing" for that message.

If no argument was given and the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, for each message in the maildrop, the POP3 server responds with a line containing information for that message. This line is called a "unique-id listing" for that message.

In order to simplify parsing, all POP3 servers are required to use a certain format for unique-id listings. A unique-id listing consists of the message-number of the message, followed by a single space and the unique-id of the message. No information follows the unique-id in the unique-id listing.

The unique-id of a message is an arbitrary server-determined string, consisting of one to 70 characters in the range 0x21 to 0x7E, which uniquely identifies a message within a maildrop and which persists across sessions. This persistence is required even if a session ends without entering the UPDATE state. The server should never reuse an unique-id in a given maildrop, for as long as the entity using the unique-id exists.

Note that messages marked as deleted are not listed.

While it is generally preferable for server implementations to store arbitrarily assigned unique-ids in the maildrop,

this specification is intended to permit unique-ids to be calculated as a hash of the message. Clients should be able to handle a situation where two identical copies of a message in a maildrop have the same unique-id.

Possible Responses:
+OK unique-id listing follows
-ERR no such message

Examples:
```
C: UIDL
S: +OK
S: 1 whqtswO00WBw418f9t5JxYwZ
S: 2 QhdPYR:00WBw1Ph7x7
S: .
   ...
C: UIDL 2
S: +OK 2 QhdPYR:00WBw1Ph7x7
   ...
C: UIDL 3
S: -ERR no such message, only 2 messages in maildrop
```

USER name

Arguments:
a string identifying a mailbox (required), which is of significance ONLY to the server

Restrictions:
may only be given in the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command

Discussion:
To authenticate using the USER and PASS command combination, the client must first issue the USER command. If the POP3 server responds with a positive status indicator ("+OK"), then the client may issue either the PASS command to complete the authentication, or the QUIT command to terminate the POP3 session. If the POP3 server responds with a negative status indicator ("-ERR") to the USER command, then the client may either issue a new authentication command or may issue the QUIT command.

The server may return a positive response even though no such mailbox exists. The server may return a negative response if mailbox exists, but does not permit plaintext





password authentication.

Possible Responses:
+OK name is a valid mailbox
-ERR never heard of mailbox name

Examples:
```
C: USER frated
S: -ERR sorry, no mailbox for frated here
   ...
C: USER mrose
S: +OK mrose is a real hoopy frood
```

PASS string

Arguments:
a server/mailbox-specific password (required)

Restrictions:
may only be given in the AUTHORIZATION state immediately after a successful USER command

Discussion:
When the client issues the PASS command, the POP3 server uses the argument pair from the USER and PASS commands to determine if the client should be given access to the appropriate maildrop.

Since the PASS command has exactly one argument, a POP3 server may treat spaces in the argument as part of the password, instead of as argument separators.

Possible Responses:
+OK maildrop locked and ready
-ERR invalid password
-ERR unable to lock maildrop

Examples:
```
C: USER mrose
S: +OK mrose is a real hoopy frood
C: PASS secret
S: -ERR maildrop already locked
  ...
C: USER mrose
S: +OK mrose is a real hoopy frood
C: PASS secret
S: +OK mrose's maildrop has 2 messages (320 octets)
```

APOP name digest

Arguments:
a string identifying a mailbox and a MD5 digest string (both required)

Restrictions:
may only be given in the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command

Discussion:
Normally, each POP3 session starts with a USER/PASS exchange. This results in a server/user-id specific password being sent in the clear on the network. For intermittent use of POP3, this may not introduce a sizable risk. However, many POP3 client implementations connect to the POP3 server on a regular basis -- to check for new mail. Further the interval of session initiation may be on the order of five minutes. Hence, the risk of password capture is greatly enhanced.

An alternate method of authentication is required which provides for both origin authentication and replay protection, but which does not involve sending a password in the clear over the network. The APOP command provides this functionality.

A POP3 server which implements the APOP command will include a timestamp in its banner greeting. The syntax of the timestamp corresponds to the `msg-id' in [RFC822], and MUST be different each time the POP3 server issues a banner greeting. For example, on a UNIX implementation in which a separate UNIX process is used for each instance of a POP3 server, the syntax of the timestamp might be:

<process-ID.clock@hostname>

where `process-ID' is the decimal value of the process's PID, clock is the decimal value of the system clock, and hostname is the fully-qualified domain-name corresponding to the host where the POP3 server is running.

The POP3 client makes note of this timestamp, and then issues the APOP command. The `name' parameter has identical semantics to the `name' parameter of the USER command. The `digest' parameter is calculated by applying the MD5 algorithm [RFC1321] to a string consisting of the timestamp (including angle-brackets) followed by a shared







secret. This shared secret is a string known only to the POP3 client and server. Great care should be taken to prevent unauthorized disclosure of the secret, as knowledge of the secret will allow any entity to successfully masquerade as the named user. The `digest' parameter itself is a 16-octet value which is sent in hexadecimal format, using lower-case ASCII characters.

When the POP3 server receives the APOP command, it verifies the digest provided. If the digest is correct, the POP3 server issues a positive response, and the POP3 session enters the TRANSACTION state. Otherwise, a negative response is issued and the POP3 session remains in the AUTHORIZATION state.

Note that as the length of the shared secret increases, so does the difficulty of deriving it. As such, shared secrets should be long strings (considerably longer than the 8-character example shown below).

Possible Responses:
+OK maildrop locked and ready
-ERR permission denied

Examples:
S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK maildrop has 1 message (369 octets)

In this example, the shared secret is the string `tanstaaf'. Hence, the MD5 algorithm is applied to the string

<1896.697170952@dbc.mtview.ca.us>tanstaaf

which produces a digest value of

c4c9334bac560ecc979e58001b3e22fb

## 8. Scaling and Operational Considerations

Since some of the optional features described above were added to the POP3 protocol, experience has accumulated in using them in large-scale commercial post office operations where most of the users are unrelated to each other. In these situations and others, users and vendors of POP3 clients have discovered that the combination of using the UIDL command and not issuing the DELE command can provide a weak version of the "maildrop as semi-permanent repository" functionality normally associated with IMAP. Of course the other capabilities of

IMAP, such as polling an existing connection for newly arrived messages and supporting multiple folders on the server, are not present in POP3.

When these facilities are used in this way by casual users, there has been a tendency for already-read messages to accumulate on the server without bound. This is clearly an undesirable behavior pattern from the standpoint of the server operator. This situation is aggravated by the fact that the limited capabilities of the POP3 do not permit efficient handling of maildrops which have hundreds or thousands of messages.

Consequently, it is recommended that operators of large-scale multi-user servers, especially ones in which the user's only access to the maildrop is via POP3, consider such options as:

* Imposing a per-user maildrop storage quota or the like.

  A disadvantage to this option is that accumulation of messages may result in the user's inability to receive new ones into the maildrop. Sites which choose this option should be sure to inform users of impending or current exhaustion of quota, perhaps by inserting an appropriate message into the user's maildrop.

* Enforce a site policy regarding mail retention on the server.

  Sites are free to establish local policy regarding the storage and retention of messages on the server, both read and unread. For example, a site might delete unread messages from the server after 60 days and delete read messages after 7 days. Such message deletions are outside the scope of the POP3 protocol and are not considered a protocol violation.

  Server operators enforcing message deletion policies should take care to make all users aware of the policies in force.

  Clients must not assume that a site policy will automate message deletions, and should continue to explicitly delete messages using the DELE command when appropriate.

  It should be noted that enforcing site message deletion policies may be confusing to the user community, since their POP3 client may contain configuration options to leave mail on the server which will not in fact be supported by the server.

  One special case of a site policy is that messages may only be downloaded once from the server, and are deleted after this has been accomplished. This could be implemented in POP3 server







software by the following mechanism: "following a POP3 login by a client which was ended by a QUIT, delete all messages downloaded during the session with the RETR command". It is important not to delete messages in the event of abnormal connection termination (ie, if no QUIT was received from the client) because the client may not have successfully received or stored the messages. Servers implementing a download-and-delete policy may also wish to disable or limit the optional TOP command, since it could be used as an alternate mechanism to download entire messages.

## 9. POP3 Command Summary

Minimal POP3 Commands:

```
   USER name               valid in the AUTHORIZATION state
   PASS string
   QUIT

   STAT                    valid in the TRANSACTION state
   LIST [msg]
   RETR msg
   DELE msg
   NOOP
   RSET
   QUIT
```

Optional POP3 Commands:

```
   APOP name digest        valid in the AUTHORIZATION state

   TOP msg n               valid in the TRANSACTION state
   UIDL [msg]
```

POP3 Replies:

```
   +OK
   -ERR
```

Note that with the exception of the STAT, LIST, and UIDL commands, the reply given by the POP3 server to any command is significant only to "+OK" and "-ERR". Any text occurring after this reply may be ignored by the client.

## 10. Example POP3 Session

```
S: <wait for connection on TCP port 110>
C: <open connection>
S:    +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:    APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
C:    RETR 1
S:    +OK 120 octets
S:    <the POP3 server sends message 1>
S:    .
C:    DELE 1
S:    +OK message 1 deleted
C:    RETR 2
S:    +OK 200 octets
S:    <the POP3 server sends message 2>
S:    .
C:    DELE 2
S:    +OK message 2 deleted
C:    QUIT
S:    +OK dewey POP3 server signing off (maildrop empty)
C:  <close connection>
S:  <wait for next connection>
```

## 11. Message Format

All messages transmitted during a POP3 session are assumed to conform to the standard for the format of Internet text messages [RFC822].

It is important to note that the octet count for a message on the server host may differ from the octet count assigned to that message due to local conventions for designating end-of-line. Usually, during the AUTHORIZATION state of the POP3 session, the POP3 server can calculate the size of each message in octets when it opens the maildrop. For example, if the POP3 server host internally represents end-of-line as a single character, then the POP3 server simply counts each occurrence of this character in a message as two octets. Note that lines in the message which start with the termination octet need not (and must not) be counted twice, since the POP3 client will remove all byte-stuffed termination characters when it receives a multi-line response.





## 12. References

[RFC821] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[RFC822] Crocker, D., "Standard for the Format of ARPA-Internet Text Messages", STD 11, RFC 822, University of Delaware, August 1982.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, MIT Laboratory for Computer Science, April 1992.

[RFC1730] Crispin, M., "Internet Message Access Protocol - Version 4", RFC 1730, University of Washington, December 1994.

[RFC1734] Myers, J., "POP3 AUTHentication command", RFC 1734, Carnegie Mellon, December 1994.

## 13. Security Considerations

It is conjectured that use of the APOP command provides origin identification and replay protection for a POP3 session. Accordingly, a POP3 server which implements both the PASS and APOP commands should not allow both methods of access for a given user; that is, for a given mailbox name, either the USER/PASS command sequence or the APOP command is allowed, but not both.

Further, note that as the length of the shared secret increases, so does the difficulty of deriving it.

Servers that answer -ERR to the USER command are giving potential attackers clues about which names are valid.

Use of the PASS command sends passwords in the clear over the network.

Use of the RETR and TOP commands sends mail in the clear over the network.

Otherwise, security issues are not discussed in this memo.

## 14. Acknowledgements

The POP family has a long and checkered history. Although primarily a minor revision to RFC 1460, POP3 is based on the ideas presented in RFCs 918, 937, and 1081.

In addition, Alfred Grimstad, Keith McCloghrie, and Neil Ostroff provided significant comments on the APOP command.

## 15. Authors' Addresses

John G. Myers
Carnegie-Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

EMail: jgm+@cmu.edu

Marshall T. Rose
Dover Beach Consulting, Inc.
420 Whisman Court
Mountain View, CA 94043-2186

EMail: mrose@dbc.mtview.ca.us







## Appendix A. Differences from RFC 1725

This memo is a revision to RFC 1725, a Draft Standard. It makes the following changes from that document:

- clarifies that command keywords are case insensitive.
- specifies that servers must send "+OK" and "-ERR" in upper case.
- specifies that the initial greeting is a positive response, instead of any string which should be a positive response.
- clarifies behavior for unimplemented commands.
- makes the USER and PASS commands optional.
- clarified the set of possible responses to the USER command.
- reverses the order of the examples in the USER and PASS commands, to reduce confusion.
- clarifies that the PASS command may only be given immediately after a successful USER command.
- clarified the persistence requirements of UIDs and added some implementation notes.
- specifies a UID length limitation of one to 70 octets.
- specifies a status indicator length limitation of 512 octets, including the CRLF.
- clarifies that LIST with no arguments on an empty mailbox returns success.
- adds a reference from the LIST command to the Message Format section
- clarifies the behavior of QUIT upon failure
- clarifies the security section to not imply the use of the USER command with the APOP command.
- adds references to RFCs 1730 and 1734
- clarifies the method by which a UA may enter mail into the transport system.

- clarifies that the second argument to the TOP command is a number of lines.

- changes the suggestion in the Security Considerations section for a server to not accept both PASS and APOP for a given user from a "must" to a "should".

- adds a section on scaling and operational considerations

Appendix B. Command Index

Network Working Group
Request for Comments: 2060
Obsoletes: 1730
Category: Standards Track

M. Crispin
University of Washington
December 1996

# INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1

## Status of this Memo



## Abstract

The Internet Message Access Protocol, Version 4rev1 (IMAP4rev1) allows a client to access and manipulate electronic mail messages on a server. IMAP4rev1 permits manipulation of remote message folders, called "mailboxes", in a way that is functionally equivalent to local mailboxes. IMAP4rev1 also provides the capability for an offline client to resynchronize with the server (see also [IMAP-DISC]).

IMAP4rev1 includes operations for creating, deleting, and renaming mailboxes; checking for new messages; permanently removing messages; setting and clearing flags; [RFC-822] and [MIME-IMB] parsing; searching; and selective fetching of message attributes, texts, and portions thereof. Messages in IMAP4rev1 are accessed by the use of numbers. These numbers are either message sequence numbers or unique identifiers.

IMAP4rev1 supports a single server. A mechanism for accessing configuration information to support multiple IMAP4rev1 servers is discussed in [ACAP].

IMAP4rev1 does not specify a means of posting mail; this function is handled by a mail transfer protocol such as [SMTP].

IMAP4rev1 is designed to be upwards compatible from the [IMAP2] and unpublished IMAP2bis protocols. In the course of the evolution of IMAP4rev1, some aspects in the earlier protocol have become obsolete. Obsolete commands, responses, and data formats which an IMAP4rev1 implementation may encounter when used with an earlier implementation are described in [IMAP-OBSOLETE].





Other compatibility issues with IMAP2bis, the most common variant of the earlier protocol, are discussed in [IMAP-COMPAT]. A full discussion of compatibility issues with rare (and presumed extinct) variants of [IMAP2] is in [IMAP-HISTORICAL]; this document is primarily of historical interest.

## Table of Contents

# IMAP4rev1 Protocol Specification

## 1. How to Read This Document

### 1.1. Organization of This Document

This document is written from the point of view of the implementor of an IMAP4rev1 client or server. Beyond the protocol overview in section 2, it is not optimized for someone trying to understand the operation of the protocol. The material in sections 3 through 5 provides the general context and definitions with which IMAP4rev1 operates.

Sections 6, 7, and 9 describe the IMAP commands, responses, and syntax, respectively. The relationships among these are such that it is almost impossible to understand any of them separately. In particular, do not attempt to deduce command syntax from the command section alone; instead refer to the Formal Syntax section.

### 1.2. Conventions Used in This Document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

The following terms are used in this document to signify the requirements of this specification.

1) MUST, or the adjective REQUIRED, means that the definition is an absolute requirement of the specification.

2) MUST NOT that the definition is an absolute prohibition of the specification.

3) SHOULD means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.

4) SHOULD NOT means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications SHOULD be understood and the case carefully weighed before implementing any behavior described with this label.

5) MAY, or the adjective OPTIONAL, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option.

"Can" is used instead of "may" when referring to a possible circumstance or situation, as opposed to an optional facility of the protocol.

"User" is used to refer to a human user, whereas "client" refers to the software being run by the user.

"Connection" refers to the entire sequence of client/server interaction from the initial establishment of the network connection until its termination. "Session" refers to the sequence of client/server interaction from the time that a mailbox is selected (SELECT or EXAMINE command) until the time that selection ends (SELECT or EXAMINE of another mailbox, CLOSE command, or connection termination).

Characters are 7-bit US-ASCII unless otherwise specified. Other character sets are indicated using a "CHARSET", as described in [MIME-IMT] and defined in [CHARSET]. CHARSETs have important additional semantics in addition to defining character set; refer to these documents for more detail.

## 2. Protocol Overview

### 2.1. Link Level

The IMAP4rev1 protocol assumes a reliable data stream such as provided by TCP. When TCP is used, an IMAP4rev1 server listens on port 143.





### 2.2. Commands and Responses

An IMAP4rev1 connection consists of the establishment of a client/server network connection, an initial greeting from the server, and client/server interactions. These client/server interactions consist of a client command, server data, and a server completion result response.

All interactions transmitted by client and server are in the form of lines; that is, strings that end with a CRLF. The protocol receiver of an IMAP4rev1 client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line.

#### 2.2.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with an identifier (typically a short alphanumeric string, e.g. A0001, A0002, etc.) called a "tag". A different tag is generated by the client for each command.

There are two cases in which a line from the client does not represent a complete command. In one case, a command argument is quoted with an octet count (see the description of literal in String under Data Formats); in the other case, the command arguments require server feedback (see the AUTHENTICATE command). In either case, the server sends a command continuation request response if it is ready for the octets (if appropriate) and the remainder of the command. This response is prefixed with the token "+".

Note: If, instead, the server detected an error in the command, it sends a BAD completion response with tag matching the command (as described below) to reject the command and prevent the client from sending any more of the command.

It is also possible for the server to send a completion response for some other command (if multiple commands are in progress), or untagged data. In either case, the command continuation request is still pending; the client takes the appropriate action for the response, and reads another response from the server. In all cases, the client MUST send a complete command (including receiving all command continuation request responses and command continuations for the command) before initiating a new command.

The protocol receiver of an IMAP4rev1 server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result response.

### 2.2.2. Server Protocol Sender and Client Protocol Receiver

Data transmitted by the server to the client and status responses that do not indicate command completion are prefixed with the token "*", and are called untagged responses.

Server data MAY be sent as a result of a client command, or MAY be sent unilaterally by the server. There is no syntactic difference between server data that resulted from a specific command and server data that were sent unilaterally.

The server completion result response indicates the success or failure of the operation. It is tagged with the same tag as the client command which began the operation. Thus, if more than one command is in progress, the tag in a server completion response identifies the command to which the response applies. There are three possible server completion responses: OK (indicating success), NO (indicating failure), or BAD (indicating protocol error such as unrecognized command or command syntax error).

The protocol receiver of an IMAP4rev1 client reads a response line from the server. It then takes action on the response based upon the first token of the response, which can be a tag, a "*", or a "+".

A client MUST be prepared to accept any server response at all times. This includes server data that was not requested. Server data SHOULD be recorded, so that the client can reference its recorded copy rather than sending a command to the server to request the data. In the case of certain server data, the data MUST be recorded.

This topic is discussed in greater detail in the Server Responses section.

## 2.3. Message Attributes

In addition to message text, each message has several attributes associated with it. These attributes may be retrieved individually or in conjunction with other attributes or message texts.

### 2.3.1. Message Numbers

Messages in IMAP4rev1 are accessed by one of two numbers; the unique identifier and the message sequence number.

#### 2.3.1.1. Unique Identifier (UID) Message Attribute

A 32-bit value assigned to each message, which when used with the unique identifier validity value (see below) forms a 64-bit value





that is permanently guaranteed not to refer to any other message in the mailbox. Unique identifiers are assigned in a strictly ascending fashion in the mailbox; as each message is added to the mailbox it is assigned a higher UID than the message(s) which were added previously.

Unlike message sequence numbers, unique identifiers are not necessarily contiguous. Unique identifiers also persist across sessions. This permits a client to resynchronize its state from a previous session with the server (e.g. disconnected or offline access clients); this is discussed further in [IMAP-DISC].

Associated with every mailbox is a unique identifier validity value, which is sent in an UIDVALIDITY response code in an OK untagged response at mailbox selection time. If unique identifiers from an earlier session fail to persist to this session, the unique identifier validity value MUST be greater than the one used in the earlier session.

> Note: Unique identifiers MUST be strictly ascending in the mailbox at all times. If the physical message store is re-ordered by a non-IMAP agent, this requires that the unique identifiers in the mailbox be regenerated, since the former unique identifers are no longer strictly ascending as a result of the re-ordering. Another instance in which unique identifiers are regenerated is if the message store has no mechanism to store unique identifiers. Although this specification recognizes that this may be unavoidable in certain server environments, it STRONGLY ENCOURAGES message store implementation techniques that avoid this problem.
>
> Another cause of non-persistance is if the mailbox is deleted and a new mailbox with the same name is created at a later date, Since the name is the same, a client may not know that this is a new mailbox unless the unique identifier validity is different. A good value to use for the unique identifier validity value is a 32-bit representation of the creation date/time of the mailbox. It is alright to use a constant such as 1, but only if it guaranteed that unique identifiers will never be reused, even in the case of a mailbox being deleted (or renamed) and a new mailbox by the same name created at some future time.

The unique identifier of a message MUST NOT change during the session, and SHOULD NOT change between sessions. However, if it is not possible to preserve the unique identifier of a message in a subsequent session, each subsequent session MUST have a new unique identifier validity value that is larger than any that was used previously.

#### 2.3.1.2. Message Sequence Number Message Attribute

A relative position from 1 to the number of messages in the mailbox. This position MUST be ordered by ascending unique identifier. As each new message is added, it is assigned a message sequence number that is 1 higher than the number of messages in the mailbox before that new message was added.

Message sequence numbers can be reassigned during the session. For example, when a message is permanently removed (expunged) from the mailbox, the message sequence number for all subsequent messages is decremented. Similarly, a new message can be assigned a message sequence number that was once held by some other message prior to an expunge.

In addition to accessing messages by relative position in the mailbox, message sequence numbers can be used in mathematical calculations. For example, if an untagged "EXISTS 11" is received, and previously an untagged "8 EXISTS" was received, three new messages have arrived with message sequence numbers of 9, 10, and 11. Another example; if message 287 in a 523 message mailbox has UID 12345, there are exactly 286 messages which have lesser UIDs and 236 messages which have greater UIDs.

### 2.3.2. Flags Message Attribute

A list of zero or more named tokens associated with the message. A flag is set by its addition to this list, and is cleared by its removal. There are two types of flags in IMAP4rev1. A flag of either type may be permanent or session-only.

A system flag is a flag name that is pre-defined in this specification. All system flags begin with "\". Certain system flags (\Deleted and \Seen) have special semantics described elsewhere. The currently-defined system flags are:

\Seen Message has been read

\Answered Message has been answered

\Flagged Message is "flagged" for urgent/special attention

\Deleted Message is "deleted" for removal by later EXPUNGE

\Draft Message has not completed composition (marked as a draft).





\Recent Message is "recently" arrived in this mailbox. This session is the first session to have been notified about this message; subsequent sessions will not see \Recent set for this message. This flag can not be altered by the client.

If it is not possible to determine whether or not this session is the first session to be notified about a message, then that message SHOULD be considered recent.

If multiple connections have the same mailbox selected simultaneously, it is undefined which of these connections will see newly-arrives messages with \Recent set and which will see it without \Recent set.

A keyword is defined by the server implementation. Keywords do not begin with "\". Servers MAY permit the client to define new keywords in the mailbox (see the description of the PERMANENTFLAGS response code for more information).

A flag may be permanent or session-only on a per-flag basis. Permanent flags are those which the client can add or remove from the message flags permanently; that is, subsequent sessions will see any change in permanent flags. Changes to session flags are valid only in that session.

Note: The \Recent system flag is a special case of a session flag. \Recent can not be used as an argument in a STORE command, and thus can not be changed at all.

### 2.3.3. Internal Date Message Attribute

The internal date and time of the message on the server. This is not the date and time in the [RFC-822] header, but rather a date and time which reflects when the message was received. In the case of messages delivered via [SMTP], this SHOULD be the date and time of final delivery of the message as defined by [SMTP]. In the case of messages delivered by the IMAP4rev1 COPY command, this SHOULD be the internal date and time of the source message. In the case of messages delivered by the IMAP4rev1 APPEND command, this SHOULD be the date and time as specified in the APPEND command description. All other cases are implementation defined.

#### 2.3.4. [RFC-822] Size Message Attribute

The number of octets in the message, as expressed in [RFC-822] format.

#### 2.3.5. Envelope Structure Message Attribute

A parsed representation of the [RFC-822] envelope information (not to be confused with an [SMTP] envelope) of the message.

#### 2.3.6. Body Structure Message Attribute

A parsed representation of the [MIME-IMB] body structure information of the message.

### 2.4. Message Texts

In addition to being able to fetch the full [RFC-822] text of a message, IMAP4rev1 permits the fetching of portions of the full message text. Specifically, it is possible to fetch the [RFC-822] message header, [RFC-822] message body, a [MIME-IMB] body part, or a [MIME-IMB] header.

## 3. State and Flow Diagram

An IMAP4rev1 server is in one of four states. Most commands are valid in only certain states. It is a protocol error for the client to attempt a command while the command is in an inappropriate state. In this case, a server will respond with a BAD or NO (depending upon server implementation) command completion result.

### 3.1. Non-Authenticated State

In non-authenticated state, the client MUST supply authentication credentials before most commands will be permitted. This state is entered when a connection starts unless the connection has been pre-authenticated.

### 3.2. Authenticated State

In authenticated state, the client is authenticated and MUST select a mailbox to access before commands that affect messages will be permitted. This state is entered when a pre-authenticated connection starts, when acceptable authentication credentials have been provided, or after an error in selecting a mailbox.



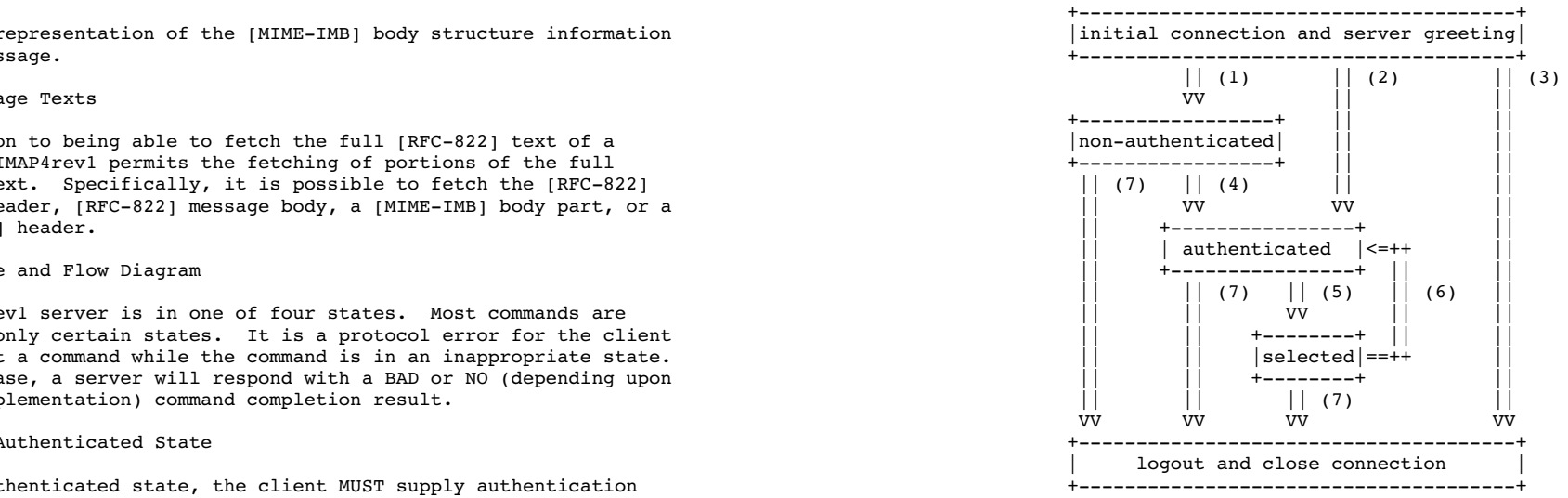


### 3.3. Selected State

In selected state, a mailbox has been selected to access. This state is entered when a mailbox has been successfully selected.

### 3.4. Logout State

In logout state, the connection is being terminated, and the server will close the connection. This state can be entered as a result of a client request or by unilateral server decision.




(1) connection without pre-authentication (OK greeting)
(2) pre-authenticated connection (PREAUTH greeting)
(3) rejected connection (BYE greeting)
(4) successful LOGIN or AUTHENTICATE command
(5) successful SELECT or EXAMINE command
(6) CLOSE command, or failed SELECT or EXAMINE command
(7) LOGOUT command, server shutdown, or connection closed

## 4. Data Formats

IMAP4rev1 uses textual commands and responses. Data in IMAP4rev1 can be in one of several forms: atom, number, string, parenthesized list, or NIL.

### 4.1. Atom

An atom consists of one or more non-special characters.

### 4.2. Number

A number consists of one or more digit characters, and represents a numeric value.

### 4.3. String

A string is in one of two forms: literal and quoted string. The literal form is the general form of string. The quoted string form is an alternative that avoids the overhead of processing a literal at the cost of limitations of characters that can be used in a quoted string.

A literal is a sequence of zero or more octets (including CR and LF), prefix-quoted with an octet count in the form of an open brace ("{"), the number of octets, close brace ("}"), and CRLF. In the case of literals transmitted from server to client, the CRLF is immediately followed by the octet data. In the case of literals transmitted from client to server, the client MUST wait to receive a command continuation request (described later in this document) before sending the octet data (and the remainder of the command).

A quoted string is a sequence of zero or more 7-bit characters, excluding CR and LF, with double quote (<">) characters at each end.

The empty string is represented as either "" (a quoted string with zero characters between double quotes) or as {0} followed by CRLF (a literal with an octet count of 0).

> Note: Even if the octet count is 0, a client transmitting a literal MUST wait to receive a command continuation request.

#### 4.3.1. 8-bit and Binary Strings

8-bit textual and binary mail is supported through the use of a [MIME-IMB] content transfer encoding. IMAP4rev1 implementations MAY transmit 8-bit or multi-octet characters in literals, but SHOULD do so only when the [CHARSET] is identified.





Although a BINARY body encoding is defined, unencoded binary strings are not permitted. A "binary string" is any string with NUL characters. Implementations MUST encode binary data into a textual form such as BASE64 before transmitting the data. A string with an excessive amount of CTL characters MAY also be considered to be binary.

### 4.4. Parenthesized List

Data structures are represented as a "parenthesized list"; a sequence of data items, delimited by space, and bounded at each end by parentheses. A parenthesized list can contain other parenthesized lists, using multiple levels of parentheses to indicate nesting.

The empty list is represented as () -- a parenthesized list with no members.

### 4.5. NIL

The special atom "NIL" represents the non-existence of a particular data item that is represented as a string or parenthesized list, as distinct from the empty string "" or the empty parenthesized list ().

## 5. Operational Considerations

### 5.1. Mailbox Naming

The interpretation of mailbox names is implementation-dependent. However, the case-insensitive mailbox name INBOX is a special name reserved to mean "the primary mailbox for this user on this server".

#### 5.1.1. Mailbox Hierarchy Naming

If it is desired to export hierarchical mailbox names, mailbox names MUST be left-to-right hierarchical using a single character to separate levels of hierarchy. The same hierarchy separator character is used for all levels of hierarchy within a single name.

#### 5.1.2. Mailbox Namespace Naming Convention

By convention, the first hierarchical element of any mailbox name which begins with "#" identifies the "namespace" of the remainder of the name. This makes it possible to disambiguate between different types of mailbox stores, each of which have their own namespaces.

For example, implementations which offer access to USENET newsgroups MAY use the "#news" namespace to partition the USENET newsgroup namespace from that of other mailboxes. Thus, the comp.mail.misc newsgroup would have an mailbox name of "#news.comp.mail.misc", and the name "comp.mail.misc" could refer to a different object (e.g. a user's private mailbox).

#### 5.1.3. Mailbox International Naming Convention

By convention, international mailbox names are specified using a modified version of the UTF-7 encoding described in [UTF-7]. The purpose of these modifications is to correct the following problems with UTF-7:

1) UTF-7 uses the "+" character for shifting; this conflicts with the common use of "+" in mailbox names, in particular USENET newsgroup names.

2) UTF-7's encoding is BASE64 which uses the "/" character; this conflicts with the use of "/" as a popular hierarchy delimiter.

3) UTF-7 prohibits the unencoded usage of "\"; this conflicts with the use of "\" as a popular hierarchy delimiter.

4) UTF-7 prohibits the unencoded usage of "~"; this conflicts with the use of "~" in some servers as a home directory indicator.

5) UTF-7 permits multiple alternate forms to represent the same string; in particular, printable US-ASCII chararacters can be represented in encoded form.

In modified UTF-7, printable US-ASCII characters except for "&" represent themselves; that is, characters with octet values 0x20-0x25 and 0x27-0x7e. The character "&" (0x26) is represented by the two-octet sequence "&-".

All other characters (octet values 0x00-0x1f, 0x7f-0xff, and all Unicode 16-bit octets) are represented in modified BASE64, with a further modification from [UTF-7] that "," is used instead of "/". Modified BASE64 MUST NOT be used to represent any printing US-ASCII character which can represent itself.

"&" is used to shift to modified BASE64 and "-" to shift back to US-ASCII. All names start in US-ASCII, and MUST end in US-ASCII (that is, a name that ends with a Unicode 16-bit octet MUST end with a "-").







For example, here is a mailbox name which mixes English, Japanese, and Chinese text: ~peter/mail/&ZeVnLIqe-/&U,BTFw-

### 5.2. Mailbox Size and Message Status Updates

At any time, a server can send data that the client did not request. Sometimes, such behavior is REQUIRED. For example, agents other than the server MAY add messages to the mailbox (e.g. new mail delivery), change the flags of message in the mailbox (e.g. simultaneous access to the same mailbox by multiple agents), or even remove messages from the mailbox. A server MUST send mailbox size updates automatically if a mailbox size change is observed during the processing of a command. A server SHOULD send message flag updates automatically, without requiring the client to request such updates explicitly. Special rules exist for server notification of a client about the removal of messages to prevent synchronization errors; see the description of the EXPUNGE response for more detail.

Regardless of what implementation decisions a client makes on remembering data from the server, a client implementation MUST record mailbox size updates. It MUST NOT assume that any command after initial mailbox selection will return the size of the mailbox.

### 5.3. Response when no Command in Progress

Server implementations are permitted to send an untagged response (except for EXPUNGE) while there is no command in progress. Server implementations that send such responses MUST deal with flow control considerations. Specifically, they MUST either (1) verify that the size of the data does not exceed the underlying transport's available window size, or (2) use non-blocking writes.

### 5.4. Autologout Timer

If a server has an inactivity autologout timer, that timer MUST be of at least 30 minutes' duration. The receipt of ANY command from the client during that interval SHOULD suffice to reset the autologout timer.

### 5.5. Multiple Commands in Progress

The client MAY send another command without waiting for the completion result response of a command, subject to ambiguity rules (see below) and flow control constraints on the underlying data stream. Similarly, a server MAY begin processing another command before processing the current command to completion, subject to ambiguity rules. However, any command continuation request responses and command continuations MUST be negotiated before any subsequent command is initiated.

The exception is if an ambiguity would result because of a command that would affect the results of other commands. Clients MUST NOT send multiple commands without waiting if an ambiguity would result. If the server detects a possible ambiguity, it MUST execute commands to completion in the order given by the client.

The most obvious example of ambiguity is when a command would affect the results of another command; for example, a FETCH of a message's flags and a STORE of that same message's flags.

A non-obvious ambiguity occurs with commands that permit an untagged EXPUNGE response (commands other than FETCH, STORE, and SEARCH), since an untagged EXPUNGE response can invalidate sequence numbers in a subsequent command. This is not a problem for FETCH, STORE, or SEARCH commands because servers are prohibited from sending EXPUNGE responses while any of those commands are in progress. Therefore, if the client sends any command other than FETCH, STORE, or SEARCH, it MUST wait for a response before sending a command with message sequence numbers.

For example, the following non-waiting command sequences are invalid:

```
FETCH + NOOP + STORE
STORE + COPY + FETCH
COPY + COPY
CHECK + FETCH
```

The following are examples of valid non-waiting command sequences:

```
FETCH + STORE + SEARCH + CHECK
STORE + COPY + EXPUNGE
```

## 6. Client Commands

IMAP4rev1 commands are described in this section. Commands are organized by the state in which the command is permitted. Commands which are permitted in multiple states are listed in the minimum





permitted state (for example, commands valid in authenticated and selected state are listed in the authenticated state commands).

Command arguments, identified by "Arguments:" in the command descriptions below, are described by function, not by syntax. The precise syntax of command arguments is described in the Formal Syntax section.

Some commands cause specific server responses to be returned; these are identified by "Responses:" in the command descriptions below. See the response descriptions in the Responses section for information on these responses, and the Formal Syntax section for the precise syntax of these responses. It is possible for server data to be transmitted as a result of any command; thus, commands that do not specifically require server data specify "no specific responses for this command" instead of "none".

The "Result:" in the command description refers to the possible tagged status responses to a command, and any special interpretation of these status responses.

### 6.1. Client Commands - Any State

The following commands are valid in any state: CAPABILITY, NOOP, and LOGOUT.

#### 6.1.1. CAPABILITY Command

Arguments: none

Responses: REQUIRED untagged response: CAPABILITY

Result: OK - capability completed
BAD - command unknown or arguments invalid

The CAPABILITY command requests a listing of capabilities that the server supports. The server MUST send a single untagged CAPABILITY response with "IMAP4rev1" as one of the listed capabilities before the (tagged) OK response. This listing of capabilities is not dependent upon connection state or user. It is therefore not necessary to issue a CAPABILITY command more than once in a connection.

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism. All such names are, by definition, part of this specification. For example, the authorization capability for an experimental "blurdybloop" authenticator would be "AUTH=XBLURDYBLOOP" and not "XAUTH=BLURDYBLOOP" or "XAUTH=XBLURDYBLOOP".

Other capability names refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response for additional information. No capabilities, beyond the base IMAP4rev1 set defined in this specification, are enabled without explicit client action to invoke the capability.

See the section entitled "Client Commands - Experimental/Expansion" for information about the form of site or implementation-specific capabilities.

Example: C: abcd CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=KERBEROS_V4
S: abcd OK CAPABILITY completed

### 6.1.2. NOOP Command

Arguments: none

Responses: no specific responses for this command (but see below)

Result: OK - noop completed
BAD - command unknown or arguments invalid

The NOOP command always succeeds. It does nothing.

Since any command can return a status update as untagged data, the NOOP command can be used as a periodic poll for new messages or message status updates during a period of inactivity. The NOOP command can also be used to reset any inactivity autologout timer on the server.

Example: C: a002 NOOP
S: a002 OK NOOP completed
. . .
C: a047 NOOP
S: * 22 EXPUNGE
S: * 23 EXISTS
S: * 3 RECENT
S: * 14 FETCH (FLAGS (\Seen \Deleted))
S: a047 OK NOOP completed





### 6.1.3. LOGOUT Command

Arguments: none

Responses: REQUIRED untagged response: BYE

Result: OK - logout completed
BAD - command unknown or arguments invalid

The LOGOUT command informs the server that the client is done with the connection. The server MUST send a BYE untagged response before the (tagged) OK response, and then close the network connection.

Example: C: A023 LOGOUT
S: * BYE IMAP4rev1 Server logging out
S: A023 OK LOGOUT completed
(Server and client then close the connection)

## 6.2. Client Commands - Non-Authenticated State

In non-authenticated state, the AUTHENTICATE or LOGIN command establishes authentication and enter authenticated state. The AUTHENTICATE command provides a general mechanism for a variety of authentication techniques, whereas the LOGIN command uses the traditional user name and plaintext password pair.

Server implementations MAY allow non-authenticated access to certain mailboxes. The convention is to use a LOGIN command with the userid "anonymous". A password is REQUIRED. It is implementation-dependent what requirements, if any, are placed on the password and what access restrictions are placed on anonymous users.

Once authenticated (including as anonymous), it is not possible to re-enter non-authenticated state.

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in non-authenticated state: AUTHENTICATE and LOGIN.

### 6.2.1. AUTHENTICATE Command

Arguments: authentication mechanism name

Responses: continuation data can be requested

Result: OK - authenticate completed, now in authenticated state
NO - authenticate failure: unsupported authentication mechanism, credentials rejected
BAD - command unknown or arguments invalid, authentication exchange cancelled

The AUTHENTICATE command indicates an authentication mechanism, such as described in [IMAP-AUTH], to the server. If the server supports the requested authentication mechanism, it performs an authentication protocol exchange to authenticate and identify the client. It MAY also negotiate an OPTIONAL protection mechanism for subsequent protocol interactions. If the requested authentication mechanism is not supported, the server SHOULD reject the AUTHENTICATE command by sending a tagged NO response.

The authentication protocol exchange consists of a series of server challenges and client answers that are specific to the authentication mechanism. A server challenge consists of a command continuation request response with the "+" token followed by a BASE64 encoded string. The client answer consists of a line consisting of a BASE64 encoded string. If the client wishes to cancel an authentication exchange, it issues a line with a single "*". If the server receives such an answer, it MUST reject the AUTHENTICATE command by sending a tagged BAD response.

A protection mechanism provides integrity and privacy protection to the connection. If a protection mechanism is negotiated, it is applied to all subsequent data sent over the connection. The protection mechanism takes effect immediately following the CRLF that concludes the authentication exchange for the client, and the CRLF of the tagged OK response for the server. Once the protection mechanism is in effect, the stream of command and response octets is processed into buffers of ciphertext. Each buffer is transferred over the connection as a stream of octets prepended with a four octet field in network byte order that represents the length of the following data. The maximum ciphertext buffer length is defined by the protection mechanism.

Authentication mechanisms are OPTIONAL. Protection mechanisms are also OPTIONAL; an authentication mechanism MAY be implemented without any protection mechanism. If an AUTHENTICATE command fails with a NO response, the client MAY try another







authentication mechanism by issuing another AUTHENTICATE command, or MAY attempt to authenticate by using the LOGIN command. In other words, the client MAY request authentication types in decreasing order of preference, with the LOGIN command as a last resort.

Example:
```
S: * OK KerberosV4 IMAP4rev1 Server
C: A001 AUTHENTICATE KERBEROS_V4
S: + AmFYig==
C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
   +nZImJjnTNHJUtxAA+o0KPKfHEcAFs9a3CL5Oebe/ydHJUwYFd
   WwuQ1MWiy6IesKvjL5rL9WjXUb9MwT9bpObYLGOKi1Qh
S: + or//EoAADZI=
C: DiAF5A4gA+oOIALuBkAAmw==
S: A001 OK Kerberos V4 authentication successful
```

Note: the line breaks in the first client answer are for editorial clarity and are not in real authenticators.

### 6.2.2. LOGIN Command

Arguments: user name
password

Responses: no specific responses for this command

Result: OK - login completed, now in authenticated state
NO - login failure: user name or password rejected
BAD - command unknown or arguments invalid

The LOGIN command identifies the client to the server and carries the plaintext password authenticating this user.

Example:
```
C: a001 LOGIN SMITH SESAME
S: a001 OK LOGIN completed
```

## 6.3. Client Commands - Authenticated State

In authenticated state, commands that manipulate mailboxes as atomic entities are permitted. Of these commands, the SELECT and EXAMINE commands will select a mailbox for access and enter selected state.

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), the following commands are valid in authenticated state: SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, STATUS, and APPEND.

### 6.3.1. SELECT Command

Arguments: mailbox name

Responses: REQUIRED untagged responses: FLAGS, EXISTS, RECENT
OPTIONAL OK untagged responses: UNSEEN, PERMANENTFLAGS

Result: OK - select completed, now in selected state
NO - select failure, now in authenticated state: no such mailbox, can't access mailbox
BAD - command unknown or arguments invalid

The SELECT command selects a mailbox so that messages in the mailbox can be accessed. Before returning an OK to the client, the server MUST send the following untagged data to the client:

FLAGS Defined flags in the mailbox. See the description of the FLAGS response for more detail.

<n> EXISTS The number of messages in the mailbox. See the description of the EXISTS response for more detail.

<n> RECENT The number of messages with the \Recent flag set. See the description of the RECENT response for more detail.

OK [UIDVALIDITY <n>]
The unique identifier validity value. See the description of the UID command for more detail.

to define the initial state of the mailbox at the client.

The server SHOULD also send an UNSEEN response code in an OK untagged response, indicating the message sequence number of the first unseen message in the mailbox.

If the client can not change the permanent state of one or more of the flags listed in the FLAGS untagged response, the server SHOULD send a PERMANENTFLAGS response code in an OK untagged response, listing the flags that the client can change permanently.

Only one mailbox can be selected at a time in a connection; simultaneous access to multiple mailboxes requires multiple connections. The SELECT command automatically deselects any currently selected mailbox before attempting the new selection. Consequently, if a mailbox is selected and a SELECT command that fails is attempted, no mailbox is selected.







If the client is permitted to modify the mailbox, the server SHOULD prefix the text of the tagged OK response with the "[READ-WRITE]" response code.

If the client is not permitted to modify the mailbox but is permitted read access, the mailbox is selected as read-only, and the server MUST prefix the text of the tagged OK response to SELECT with the "[READ-ONLY]" response code. Read-only access through SELECT differs from the EXAMINE command in that certain read-only mailboxes MAY permit the change of permanent state on a per-user (as opposed to global) basis. Netnews messages marked in a server-based .newsrc file are an example of such per-user permanent state that can be modified with read-only mailboxes.

Example: C: A142 SELECT INBOX
S: * 172 EXISTS
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen *)] Limited
S: A142 OK [READ-WRITE] SELECT completed

### 6.3.2. EXAMINE Command

Arguments: mailbox name

Responses: REQUIRED untagged responses: FLAGS, EXISTS, RECENT
OPTIONAL OK untagged responses: UNSEEN, PERMANENTFLAGS

Result: OK - examine completed, now in selected state
NO - examine failure, now in authenticated state: no such mailbox, can't access mailbox
BAD - command unknown or arguments invalid

The EXAMINE command is identical to SELECT and returns the same output; however, the selected mailbox is identified as read-only. No changes to the permanent state of the mailbox, including per-user state, are permitted.

The text of the tagged OK response to the EXAMINE command MUST begin with the "[READ-ONLY]" response code.

```
Example:    C: A932 EXAMINE blurdybloop
            S: * 17 EXISTS
            S: * 2 RECENT
            S: * OK [UNSEEN 8] Message 8 is first unseen
            S: * OK [UIDVALIDITY 3857529045] UIDs valid
            S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
            S: * OK [PERMANENTFLAGS ()] No permanent flags permitted
            S: A932 OK [READ-ONLY] EXAMINE completed
```

### 6.3.3. CREATE Command

```
Arguments:  mailbox name

Responses:  no specific responses for this command

Result:     OK - create completed
            NO - create failure: can't create mailbox with that name
            BAD - command unknown or arguments invalid
```

The CREATE command creates a mailbox with the given name. An OK response is returned only if a new mailbox with that name has been created. It is an error to attempt to create INBOX or a mailbox with a name that refers to an extant mailbox. Any error in creation will return a tagged NO response.

If the mailbox name is suffixed with the server's hierarchy separator character (as returned from the server by a LIST command), this is a declaration that the client intends to create mailbox names under this name in the hierarchy. Server implementations that do not require this declaration MUST ignore it.

If the server's hierarchy separator character appears elsewhere in the name, the server SHOULD create any superior hierarchical names that are needed for the CREATE command to complete successfully. In other words, an attempt to create "foo/bar/zap" on a server in which "/" is the hierarchy separator character SHOULD create foo/ and foo/bar/ if they do not already exist.

If a new mailbox is created with the same name as a mailbox which was deleted, its unique identifiers MUST be greater than any unique identifiers used in the previous incarnation of the mailbox UNLESS the new incarnation has a different unique identifier validity value. See the description of the UID command for more detail.





```
Example:    C: A003 CREATE owatagusiam/
            S: A003 OK CREATE completed
            C: A004 CREATE owatagusiam/blurdybloop
            S: A004 OK CREATE completed
```

Note: the interpretation of this example depends on whether "/" was returned as the hierarchy separator from LIST. If "/" is the hierarchy separator, a new level of hierarchy named "owatagusiam" with a member called "blurdybloop" is created. Otherwise, two mailboxes at the same hierarchy level are created.

### 6.3.4. DELETE Command

```
Arguments:  mailbox name

Responses:  no specific responses for this command

Result:     OK - delete completed
            NO - delete failure: can't delete mailbox with that name
            BAD - command unknown or arguments invalid
```

The DELETE command permanently removes the mailbox with the given name. A tagged OK response is returned only if the mailbox has been deleted. It is an error to attempt to delete INBOX or a mailbox name that does not exist.

The DELETE command MUST NOT remove inferior hierarchical names. For example, if a mailbox "foo" has an inferior "foo.bar" (assuming "." is the hierarchy delimiter character), removing "foo" MUST NOT remove "foo.bar". It is an error to attempt to delete a name that has inferior hierarchical names and also has the \Noselect mailbox name attribute (see the description of the LIST response for more details).

It is permitted to delete a name that has inferior hierarchical names and does not have the \Noselect mailbox name attribute. In this case, all messages in that mailbox are removed, and the name will acquire the \Noselect mailbox name attribute.

The value of the highest-used unique identifier of the deleted mailbox MUST be preserved so that a new mailbox created with the same name will not reuse the identifiers of the former incarnation, UNLESS the new incarnation has a different unique identifier validity value. See the description of the UID command for more detail.

```
   Examples:   C: A682 LIST "" *
               S: * LIST () "/" blurdybloop
               S: * LIST (\Noselect) "/" foo
               S: * LIST () "/" foo/bar
               S: A682 OK LIST completed
               C: A683 DELETE blurdybloop
               S: A683 OK DELETE completed
               C: A684 DELETE foo
               S: A684 NO Name "foo" has inferior hierarchical names
               C: A685 DELETE foo/bar
               S: A685 OK DELETE Completed
               C: A686 LIST "" *
               S: * LIST (\Noselect) "/" foo
               S: A686 OK LIST completed
               C: A687 DELETE foo
               S: A687 OK DELETE Completed


               C: A82 LIST "" *
               S: * LIST () "." blurdybloop
               S: * LIST () "." foo
               S: * LIST () "." foo.bar
               S: A82 OK LIST completed
               C: A83 DELETE blurdybloop
               S: A83 OK DELETE completed
               C: A84 DELETE foo
               S: A84 OK DELETE Completed
               C: A85 LIST "" *
               S: * LIST () "." foo.bar
               S: A85 OK LIST completed
               C: A86 LIST "" %
               S: * LIST (\Noselect) "." foo
               S: A86 OK LIST completed
```

### 6.3.5. RENAME Command

```
   Arguments:  existing mailbox name
               new mailbox name

   Responses:  no specific responses for this command

   Result:     OK - rename completed
               NO - rename failure: can't rename mailbox with that name,
                    can't rename to mailbox with that name
               BAD - command unknown or arguments invalid
```

The RENAME command changes the name of a mailbox. A tagged OK response is returned only if the mailbox has been renamed. It is





an error to attempt to rename from a mailbox name that does not exist or to a mailbox name that already exists. Any error in renaming will return a tagged NO response.

If the name has inferior hierarchical names, then the inferior hierarchical names MUST also be renamed. For example, a rename of "foo" to "zap" will rename "foo/bar" (assuming "/" is the hierarchy delimiter character) to "zap/bar".

The value of the highest-used unique identifier of the old mailbox name MUST be preserved so that a new mailbox created with the same name will not reuse the identifiers of the former incarnation, UNLESS the new incarnation has a different unique identifier validity value. See the description of the UID command for more detail.

Renaming INBOX is permitted, and has special behavior. It moves all messages in INBOX to a new mailbox with the given name, leaving INBOX empty. If the server implementation supports inferior hierarchical names of INBOX, these are unaffected by a rename of INBOX.

```
   Examples:   C: A682 LIST "" *
               S: * LIST () "/" blurdybloop
               S: * LIST (\Noselect) "/" foo
               S: * LIST () "/" foo/bar
               S: A682 OK LIST completed
               C: A683 RENAME blurdybloop sarasoop
               S: A683 OK RENAME completed
               C: A684 RENAME foo zowie
               S: A684 OK RENAME Completed
               C: A685 LIST "" *
               S: * LIST () "/" sarasoop
               S: * LIST (\Noselect) "/" zowie
               S: * LIST () "/" zowie/bar
               S: A685 OK LIST completed
```

```
C: Z432 LIST "" *
S: * LIST () "." INBOX
S: * LIST () "." INBOX.bar
S: Z432 OK LIST completed
C: Z433 RENAME INBOX old-mail
S: Z433 OK RENAME completed
C: Z434 LIST "" *
S: * LIST () "." INBOX
S: * LIST () "." INBOX.bar
S: * LIST () "." old-mail
S: Z434 OK LIST completed
```

### 6.3.6. SUBSCRIBE Command

Arguments: mailbox

Responses: no specific responses for this command

Result: OK - subscribe completed
NO - subscribe failure: can't subscribe to that name
BAD - command unknown or arguments invalid

The SUBSCRIBE command adds the specified mailbox name to the server's set of "active" or "subscribed" mailboxes as returned by the LSUB command. This command returns a tagged OK response only if the subscription is successful.

A server MAY validate the mailbox argument to SUBSCRIBE to verify that it exists. However, it MUST NOT unilaterally remove an existing mailbox name from the subscription list even if a mailbox by that name no longer exists.

Note: this requirement is because some server sites may routinely remove a mailbox with a well-known name (e.g. "system-alerts") after its contents expire, with the intention of recreating it when new contents are appropriate.

Example: C: A002 SUBSCRIBE #news.comp.mail.mime
S: A002 OK SUBSCRIBE completed







### 6.3.7. UNSUBSCRIBE Command

Arguments: mailbox name

Responses: no specific responses for this command

Result: OK - unsubscribe completed
NO - unsubscribe failure: can't unsubscribe that name
BAD - command unknown or arguments invalid

The UNSUBSCRIBE command removes the specified mailbox name from the server's set of "active" or "subscribed" mailboxes as returned by the LSUB command. This command returns a tagged OK response only if the unsubscription is successful.

Example: C: A002 UNSUBSCRIBE #news.comp.mail.mime
S: A002 OK UNSUBSCRIBE completed

### 6.3..8. LIST Command

Arguments: reference name
mailbox name with possible wildcards

Responses: untagged responses: LIST

Result: OK - list completed
NO - list failure: can't list that reference or name
BAD - command unknown or arguments invalid

The LIST command returns a subset of names from the complete set of all names available to the client. Zero or more untagged LIST replies are returned, containing the name attributes, hierarchy delimiter, and name; see the description of the LIST reply for more detail.

The LIST command SHOULD return its data quickly, without undue delay. For example, it SHOULD NOT go to excess trouble to calculate \Marked or \Unmarked status or perform other processing; if each name requires 1 second of processing, then a list of 1200 names would take 20 minutes!

An empty ("" string) reference name argument indicates that the mailbox name is interpreted as by SELECT. The returned mailbox names MUST match the supplied mailbox name pattern. A non-empty reference name argument is the name of a mailbox or a level of mailbox hierarchy, and indicates a context in which the mailbox name is interpreted in an implementation-defined manner.

An empty ("" string) mailbox name argument is a special request to return the hierarchy delimiter and the root name of the name given in the reference. The value returned as the root MAY be null if the reference is non-rooted or is null. In all cases, the hierarchy delimiter is returned. This permits a client to get the hierarchy delimiter even when no mailboxes by that name currently exist.

The reference and mailbox name arguments are interpreted, in an implementation-dependent fashion, into a canonical form that represents an unambiguous left-to-right hierarchy. The returned mailbox names will be in the interpreted form.

Any part of the reference argument that is included in the interpreted form SHOULD prefix the interpreted form. It SHOULD also be in the same form as the reference name argument. This rule permits the client to determine if the returned mailbox name is in the context of the reference argument, or if something about the mailbox argument overrode the reference argument. Without this rule, the client would have to have knowledge of the server's naming semantics including what characters are "breakouts" that override a naming context.

For example, here are some examples of how references and mailbox names might be interpreted on a UNIX-based server:

| Reference | Mailbox Name | Interpretation |
|---|---|---|
| ~smith/Mail/ | foo.* | ~smith/Mail/foo.* |
| archive/ | % | archive/% |
| #news. | comp.mail.* | #news.comp.mail.* |
| ~smith/Mail/ | /usr/doc/foo | /usr/doc/foo |
| archive/ | ~fred/Mail/* | ~fred/Mail/* |

The first three examples demonstrate interpretations in the context of the reference argument. Note that "~smith/Mail" SHOULD NOT be transformed into something like "/u2/users/smith/Mail", or it would be impossible for the client to determine that the interpretation was in the context of the reference.

The character "*" is a wildcard, and matches zero or more characters at this position. The character "%" is similar to "*", but it does not match a hierarchy delimiter. If the "%" wildcard is the last character of a mailbox name argument, matching levels of hierarchy are also returned. If these levels of hierarchy are not also selectable mailboxes, they are returned with the \Noselect mailbox name attribute (see the description of the LIST response for more details).







Server implementations are permitted to "hide" otherwise accessible mailboxes from the wildcard characters, by preventing certain characters or names from matching a wildcard in certain situations. For example, a UNIX-based server might restrict the interpretation of "*" so that an initial "/" character does not match.

The special name INBOX is included in the output from LIST, if INBOX is supported by this server for this user and if the uppercase string "INBOX" matches the interpreted reference and mailbox name arguments with wildcards as described above. The criteria for omitting INBOX is whether SELECT INBOX will return failure; it is not relevant whether the user's real INBOX resides on this or some other server.

Example:
```
C: A101 LIST "" ""
S: * LIST (\Noselect) "/" ""
S: A101 OK LIST Completed
C: A102 LIST #news.comp.mail.misc ""
S: * LIST (\Noselect) "." #news.
S: A102 OK LIST Completed
C: A103 LIST /usr/staff/jones ""
S: * LIST (\Noselect) "/" /
S: A103 OK LIST Completed
C: A202 LIST ~/Mail/ %
S: * LIST (\Noselect) "/" ~/Mail/foo
S: * LIST () "/" ~/Mail/meetings
S: A202 OK LIST completed
```

### 6.3.9. LSUB Command

Arguments: reference name
mailbox name with possible wildcards

Responses: untagged responses: LSUB

Result: OK - lsub completed
NO - lsub failure: can't list that reference or name
BAD - command unknown or arguments invalid

The LSUB command returns a subset of names from the set of names that the user has declared as being "active" or "subscribed". Zero or more untagged LSUB replies are returned. The arguments to LSUB are in the same form as those for LIST.

A server MAY validate the subscribed names to see if they still exist. If a name does not exist, it SHOULD be flagged with the \Noselect attribute in the LSUB response. The server MUST NOT

unilaterally remove an existing mailbox name from the subscription list even if a mailbox by that name no longer exists.

Example: C: A002 LSUB "#news." "comp.mail.*"
S: * LSUB () "." #news.comp.mail.mime
S: * LSUB () "." #news.comp.mail.misc
S: A002 OK LSUB completed

### 6.3.10. STATUS Command

Arguments: mailbox name
status data item names

Responses: untagged responses: STATUS

Result: OK - status completed
NO - status failure: no status for that name
BAD - command unknown or arguments invalid

The STATUS command requests the status of the indicated mailbox. It does not change the currently selected mailbox, nor does it affect the state of any messages in the queried mailbox (in particular, STATUS MUST NOT cause messages to lose the \Recent flag).

The STATUS command provides an alternative to opening a second IMAP4rev1 connection and doing an EXAMINE command on a mailbox to query that mailbox's status without deselecting the current mailbox in the first IMAP4rev1 connection.

Unlike the LIST command, the STATUS command is not guaranteed to be fast in its response. In some implementations, the server is obliged to open the mailbox read-only internally to obtain certain status information. Also unlike the LIST command, the STATUS command does not accept wildcards.

The currently defined status data items that can be requested are:

MESSAGES The number of messages in the mailbox.

RECENT The number of messages with the \Recent flag set.

UIDNEXT The next UID value that will be assigned to a new message in the mailbox. It is guaranteed that this value will not change unless new messages are added to the mailbox; and that it will change when new messages are added even if those new messages are subsequently expunged.





UIDVALIDITY The unique identifier validity value of the mailbox.

UNSEEN The number of messages which do not have the \Seen flag set.

Example: C: A042 STATUS blurdybloop (UIDNEXT MESSAGES)
S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292)
S: A042 OK STATUS completed

### 6.3.11. APPEND Command

Arguments: mailbox name
OPTIONAL flag parenthesized list
OPTIONAL date/time string
message literal

Responses: no specific responses for this command

Result: OK - append completed
NO - append error: can't append to that mailbox, error in flags or date/time or message text
BAD - command unknown or arguments invalid

The APPEND command appends the literal argument as a new message to the end of the specified destination mailbox. This argument SHOULD be in the format of an [RFC-822] message. 8-bit characters are permitted in the message. A server implementation that is unable to preserve 8-bit data properly MUST be able to reversibly convert 8-bit APPEND data to 7-bit using a [MIME-IMB] content transfer encoding.

Note: There MAY be exceptions, e.g. draft messages, in which required [RFC-822] header lines are omitted in the message literal argument to APPEND. The full implications of doing so MUST be understood and carefully weighed.

If a flag parenthesized list is specified, the flags SHOULD be set in the resulting message; otherwise, the flag list of the resulting message is set empty by default.

If a date_time is specified, the internal date SHOULD be set in the resulting message; otherwise, the internal date of the resulting message is set to the current date and time by default.

If the append is unsuccessful for any reason, the mailbox MUST be restored to its state before the APPEND attempt; no partial appending is permitted.

If the destination mailbox does not exist, a server MUST return an error, and MUST NOT automatically create the mailbox. Unless it is certain that the destination mailbox can not be created, the server MUST send the response code "[TRYCREATE]" as the prefix of the text of the tagged NO response. This gives a hint to the client that it can attempt a CREATE command and retry the APPEND if the CREATE is successful.

If the mailbox is currently selected, the normal new mail actions SHOULD occur. Specifically, the server SHOULD notify the client immediately via an untagged EXISTS response. If the server does not do so, the client MAY issue a NOOP command (or failing that, a CHECK command) after one or more APPEND commands.

```
Example:    C: A003 APPEND saved-messages (\Seen) {310}
            C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
            C: From: Fred Foobar <foobar@Blurdybloop.COM>
            C: Subject: afternoon meeting
            C: To: mooch@owatagu.siam.edu
            C: Message-Id: <B27397-0100000@Blurdybloop.COM>
            C: MIME-Version: 1.0
            C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
            C:
            C: Hello Joe, do you think we can meet at 3:30 tomorrow?
            C:
            S: A003 OK APPEND completed
```

Note: the APPEND command is not used for message delivery, because it does not provide a mechanism to transfer [SMTP] envelope information.

## 6.4. Client Commands - Selected State

In selected state, commands that manipulate messages in a mailbox are permitted.

In addition to the universal commands (CAPABILITY, NOOP, and LOGOUT), and the authenticated state commands (SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, STATUS, and APPEND), the following commands are valid in the selected state: CHECK, CLOSE, EXPUNGE, SEARCH, FETCH, STORE, COPY, and UID.







### 6.4.1. CHECK Command

```
Arguments:  none

Responses:  no specific responses for this command

Result:     OK - check completed
            BAD - command unknown or arguments invalid
```

The CHECK command requests a checkpoint of the currently selected mailbox. A checkpoint refers to any implementation-dependent housekeeping associated with the mailbox (e.g. resolving the server's in-memory state of the mailbox with the state on its disk) that is not normally executed as part of each command. A checkpoint MAY take a non-instantaneous amount of real time to complete. If a server implementation has no such housekeeping considerations, CHECK is equivalent to NOOP.

There is no guarantee that an EXISTS untagged response will happen as a result of CHECK. NOOP, not CHECK, SHOULD be used for new mail polling.

```
Example:    C: FXXZ CHECK
            S: FXXZ OK CHECK Completed
```

### 6.4.2. CLOSE Command

```
Arguments:  none

Responses:  no specific responses for this command

Result:     OK - close completed, now in authenticated state
            NO - close failure: no mailbox selected
            BAD - command unknown or arguments invalid
```

The CLOSE command permanently removes from the currently selected mailbox all messages that have the \Deleted flag set, and returns to authenticated state from selected state. No untagged EXPUNGE responses are sent.

No messages are removed, and no error is given, if the mailbox is selected by an EXAMINE command or is otherwise selected read-only.

Even if a mailbox is selected, a SELECT, EXAMINE, or LOGOUT command MAY be issued without previously issuing a CLOSE command. The SELECT, EXAMINE, and LOGOUT commands implicitly close the currently selected mailbox without doing an expunge. However, when many messages are deleted, a CLOSE-LOGOUT or CLOSE-SELECT

sequence is considerably faster than an EXPUNGE-LOGOUT or EXPUNGE-SELECT because no untagged EXPUNGE responses (which the client would probably ignore) are sent.

Example: C: A341 CLOSE
S: A341 OK CLOSE completed

### 6.4.3. EXPUNGE Command

Arguments: none

Responses: untagged responses: EXPUNGE

Result: OK - expunge completed
NO - expunge failure: can't expunge (e.g. permission denied)
BAD - command unknown or arguments invalid

The EXPUNGE command permanently removes from the currently selected mailbox all messages that have the \Deleted flag set. Before returning an OK to the client, an untagged EXPUNGE response is sent for each message that is removed.

Example: C: A202 EXPUNGE
S: * 3 EXPUNGE
S: * 3 EXPUNGE
S: * 5 EXPUNGE
S: * 8 EXPUNGE
S: A202 OK EXPUNGE completed

Note: in this example, messages 3, 4, 7, and 11 had the \Deleted flag set. See the description of the EXPUNGE response for further explanation.

### 6.4.4. SEARCH Command

Arguments: OPTIONAL [CHARSET] specification
searching criteria (one or more)

Responses: REQUIRED untagged response: SEARCH

Result: OK - search completed
NO - search error: can't search that [CHARSET] or criteria
BAD - command unknown or arguments invalid





The SEARCH command searches the mailbox for messages that match the given searching criteria. Searching criteria consist of one or more search keys. The untagged SEARCH response from the server contains a listing of message sequence numbers corresponding to those messages that match the searching criteria.

When multiple keys are specified, the result is the intersection (AND function) of all the messages that match those keys. For example, the criteria DELETED FROM "SMITH" SINCE 1-Feb-1994 refers to all deleted messages from Smith that were placed in the mailbox since February 1, 1994. A search key can also be a parenthesized list of one or more search keys (e.g. for use with the OR and NOT keys).

Server implementations MAY exclude [MIME-IMB] body parts with terminal content media types other than TEXT and MESSAGE from consideration in SEARCH matching.

The OPTIONAL [CHARSET] specification consists of the word "CHARSET" followed by a registered [CHARSET]. It indicates the [CHARSET] of the strings that appear in the search criteria. [MIME-IMB] content transfer encodings, and [MIME-HDRS] strings in [RFC-822]/[MIME-IMB] headers, MUST be decoded before comparing text in a [CHARSET] other than US-ASCII. US-ASCII MUST be supported; other [CHARSET]s MAY be supported. If the server does not support the specified [CHARSET], it MUST return a tagged NO response (not a BAD).

In all search keys that use strings, a message matches the key if the string is a substring of the field. The matching is case-insensitive.

The defined search keys are as follows. Refer to the Formal Syntax section for the precise syntactic definitions of the arguments.

<message set> Messages with message sequence numbers corresponding to the specified message sequence number set

ALL All messages in the mailbox; the default initial key for ANDing.

ANSWERED Messages with the \Answered flag set.

BCC <string> Messages that contain the specified string in the envelope structure's BCC field.

BEFORE <date> Messages whose internal date is earlier than the specified date.

BODY <string> Messages that contain the specified string in the body of the message.

CC <string> Messages that contain the specified string in the envelope structure's CC field.

DELETED Messages with the \Deleted flag set.

DRAFT Messages with the \Draft flag set.

FLAGGED Messages with the \Flagged flag set.

FROM <string> Messages that contain the specified string in the envelope structure's FROM field.

HEADER <field-name> <string>
Messages that have a header with the specified field-name (as defined in [RFC-822]) and that contains the specified string in the [RFC-822] field-body.

KEYWORD <flag> Messages with the specified keyword set.

LARGER <n> Messages with an [RFC-822] size larger than the specified number of octets.

NEW Messages that have the \Recent flag set but not the \Seen flag. This is functionally equivalent to "(RECENT UNSEEN)".

NOT <search-key>
Messages that do not match the specified search key.

OLD Messages that do not have the \Recent flag set. This is functionally equivalent to "NOT RECENT" (as opposed to "NOT NEW").

ON <date> Messages whose internal date is within the specified date.

OR <search-key1> <search-key2>
Messages that match either search key.

RECENT Messages that have the \Recent flag set.





SEEN Messages that have the \Seen flag set.

SENTBEFORE <date>
Messages whose [RFC-822] Date: header is earlier than the specified date.

SENTON <date> Messages whose [RFC-822] Date: header is within the specified date.

SENTSINCE <date>
Messages whose [RFC-822] Date: header is within or later than the specified date.

SINCE <date> Messages whose internal date is within or later than the specified date.

SMALLER <n> Messages with an [RFC-822] size smaller than the specified number of octets.

SUBJECT <string>
Messages that contain the specified string in the envelope structure's SUBJECT field.

TEXT <string> Messages that contain the specified string in the header or body of the message.

TO <string> Messages that contain the specified string in the envelope structure's TO field.

UID <message set>
Messages with unique identifiers corresponding to the specified unique identifier set.

UNANSWERED Messages that do not have the \Answered flag set.

UNDELETED Messages that do not have the \Deleted flag set.

UNDRAFT Messages that do not have the \Draft flag set.

UNFLAGGED Messages that do not have the \Flagged flag set.

UNKEYWORD <flag>
Messages that do not have the specified keyword set.

UNSEEN Messages that do not have the \Seen flag set.

Example: C: A282 SEARCH FLAGGED SINCE 1-Feb-1994 NOT FROM "Smith"
S: * SEARCH 2 84 882
S: A282 OK SEARCH completed

### 6.4.5. FETCH Command

Arguments: message set
message data item names

Responses: untagged responses: FETCH

Result: OK - fetch completed
NO - fetch error: can't fetch that data
BAD - command unknown or arguments invalid

The FETCH command retrieves data associated with a message in the mailbox. The data items to be fetched can be either a single atom or a parenthesized list.

The currently defined data items that can be fetched are:

ALL Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)

BODY Non-extensible form of BODYSTRUCTURE.

BODY[<section>]<<partial>>
The text of a particular body section. The section specification is a set of zero or more part specifiers delimited by periods. A part specifier is either a part number or one of the following: HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, MIME, and TEXT. An empty section specification refers to the entire message, including the header.

Every message has at least one part number. Non-[MIME-IMB] messages, and non-multipart [MIME-IMB] messages with no encapsulated message, only have a part 1.

Multipart messages are assigned consecutive part numbers, as they occur in the message. If a particular part is of type message or multipart, its parts MUST be indicated by a period followed by the part number within that nested multipart part.





A part of type MESSAGE/RFC822 also has nested part numbers, referring to parts of the MESSAGE part's body.

The HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, and TEXT part specifiers can be the sole part specifier or can be prefixed by one or more numeric part specifiers, provided that the numeric part specifier refers to a part of type MESSAGE/RFC822. The MIME part specifier MUST be prefixed by one or more numeric part specifiers.

The HEADER, HEADER.FIELDS, and HEADER.FIELDS.NOT part specifiers refer to the [RFC-822] header of the message or of an encapsulated [MIME-IMT] MESSAGE/RFC822 message. HEADER.FIELDS and HEADER.FIELDS.NOT are followed by a list of field-name (as defined in [RFC-822]) names, and return a subset of the header. The subset returned by HEADER.FIELDS contains only those header fields with a field-name that matches one of the names in the list; similarly, the subset returned by HEADER.FIELDS.NOT contains only the header fields with a non-matching field-name. The field-matching is case-insensitive but otherwise exact. In all cases, the delimiting blank line between the header and the body is always included.

The MIME part specifier refers to the [MIME-IMB] header for this part.

The TEXT part specifier refers to the text body of the message, omitting the [RFC-822] header.

Here is an example of a complex message with some of its part specifiers:

```
HEADER     ([RFC-822] header of the message)
TEXT       MULTIPART/MIXED
1          TEXT/PLAIN
2          APPLICATION/OCTET-STREAM
3          MESSAGE/RFC822
3.HEADER   ([RFC-822] header of the message)
3.TEXT     ([RFC-822] text body of the message)
3.1        TEXT/PLAIN
3.2        APPLICATION/OCTET-STREAM
4          MULTIPART/MIXED
4.1        IMAGE/GIF
4.1.MIME   ([MIME-IMB] header for the IMAGE/GIF)
4.2        MESSAGE/RFC822
4.2.HEADER ([RFC-822] header of the message)
4.2.TEXT   ([RFC-822] text body of the message)
4.2.1      TEXT/PLAIN
4.2.2      MULTIPART/ALTERNATIVE
4.2.2.1    TEXT/PLAIN
4.2.2.2    TEXT/RICHTEXT
```

It is possible to fetch a substring of the designated text. This is done by appending an open angle bracket ("<"), the octet position of the first desired octet, a period, the maximum number of octets desired, and a close angle bracket (">") to the part specifier. If the starting octet is beyond the end of the text, an empty string is returned.

Any partial fetch that attempts to read beyond the end of the text is truncated as appropriate. A partial fetch that starts at octet 0 is returned as a partial fetch, even if this truncation happened.

> Note: this means that BODY[]<0.2048> of a 1500-octet message will return BODY[]<0> with a literal of size 1500, not BODY[].

> Note: a substring fetch of a HEADER.FIELDS or HEADER.FIELDS.NOT part specifier is calculated after subsetting the header.





The \Seen flag is implicitly set; if this causes the flags to change they SHOULD be included as part of the FETCH responses.

BODY.PEEK[<section>]<<partial>>
: An alternate form of BODY[<section>] that does not implicitly set the \Seen flag.

BODYSTRUCTURE
: The [MIME-IMB] body structure of the message. This is computed by the server by parsing the [MIME-IMB] header fields in the [RFC-822] header and [MIME-IMB] headers.

ENVELOPE
: The envelope structure of the message. This is computed by the server by parsing the [RFC-822] header into the component parts, defaulting various fields as necessary.

FAST
: Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE)

FLAGS
: The flags that are set for this message.

FULL
: Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE ENVELOPE BODY)

INTERNALDATE
: The internal date of the message.

RFC822
: Functionally equivalent to BODY[], differing in the syntax of the resulting untagged FETCH data (RFC822 is returned).

RFC822.HEADER
: Functionally equivalent to BODY.PEEK[HEADER], differing in the syntax of the resulting untagged FETCH data (RFC822.HEADER is returned).

RFC822.SIZE
: The [RFC-822] size of the message.

RFC822.TEXT
: Functionally equivalent to BODY[TEXT], differing in the syntax of the resulting untagged FETCH data (RFC822.TEXT is returned).

UID
: The unique identifier for the message.

Example: C: A654 FETCH 2:4 (FLAGS BODY[HEADER.FIELDS (DATE FROM)])
S: * 2 FETCH ....
S: * 3 FETCH ....
S: * 4 FETCH ....
S: A654 OK FETCH completed

### 6.4.6. STORE Command

Arguments: message set
message data item name
value for message data item

Responses: untagged responses: FETCH

Result: OK - store completed
NO - store error: can't store that data
BAD - command unknown or arguments invalid

The STORE command alters data associated with a message in the mailbox. Normally, STORE will return the updated value of the data with an untagged FETCH response. A suffix of ".SILENT" in the data item name prevents the untagged FETCH, and the server SHOULD assume that the client has determined the updated value itself or does not care about the updated value.

Note: regardless of whether or not the ".SILENT" suffix was used, the server SHOULD send an untagged FETCH response if a change to a message's flags from an external source is observed. The intent is that the status of the flags is determinate without a race condition.

The currently defined data items that can be stored are:

FLAGS <flag list>
Replace the flags for the message with the argument. The new value of the flags are returned as if a FETCH of those flags was done.

FLAGS.SILENT <flag list>
Equivalent to FLAGS, but without returning a new value.

+FLAGS <flag list>
Add the argument to the flags for the message. The new value of the flags are returned as if a FETCH of those flags was done.





+FLAGS.SILENT <flag list>
Equivalent to +FLAGS, but without returning a new value.

-FLAGS <flag list>
Remove the argument from the flags for the message. The new value of the flags are returned as if a FETCH of those flags was done.

-FLAGS.SILENT <flag list>
Equivalent to -FLAGS, but without returning a new value.

Example: C: A003 STORE 2:4 +FLAGS (\Deleted)
S: * 2 FETCH FLAGS (\Deleted \Seen)
S: * 3 FETCH FLAGS (\Deleted)
S: * 4 FETCH FLAGS (\Deleted \Flagged \Seen)
S: A003 OK STORE completed

### 6.4.7. COPY Command

Arguments: message set
mailbox name

Responses: no specific responses for this command

Result: OK - copy completed
NO - copy error: can't copy those messages or to that name
BAD - command unknown or arguments invalid

The COPY command copies the specified message(s) to the end of the specified destination mailbox. The flags and internal date of the message(s) SHOULD be preserved in the copy.

If the destination mailbox does not exist, a server SHOULD return an error. It SHOULD NOT automatically create the mailbox. Unless it is certain that the destination mailbox can not be created, the server MUST send the response code "[TRYCREATE]" as the prefix of the text of the tagged NO response. This gives a hint to the client that it can attempt a CREATE command and retry the COPY if the CREATE is successful.

If the COPY command is unsuccessful for any reason, server implementations MUST restore the destination mailbox to its state before the COPY attempt.

Example: C: A003 COPY 2:4 MEETING
S: A003 OK COPY completed

### 6.4.8. UID Command

Arguments: command name
command arguments

Responses: untagged responses: FETCH, SEARCH

Result: OK - UID command completed
NO - UID command error
BAD - command unknown or arguments invalid

The UID command has two forms. In the first form, it takes as its arguments a COPY, FETCH, or STORE command with arguments appropriate for the associated command. However, the numbers in the message set argument are unique identifiers instead of message sequence numbers.

In the second form, the UID command takes a SEARCH command with SEARCH command arguments. The interpretation of the arguments is the same as with SEARCH; however, the numbers returned in a SEARCH response for a UID SEARCH command are unique identifiers instead of message sequence numbers. For example, the command UID SEARCH 1:100 UID 443:557 returns the unique identifiers corresponding to the intersection of the message sequence number set 1:100 and the UID set 443:557.

Message set ranges are permitted; however, there is no guarantee that unique identifiers be contiguous. A non-existent unique identifier within a message set range is ignored without any error message generated.

The number after the "*" in an untagged FETCH response is always a message sequence number, not a unique identifier, even for a UID command response. However, server implementations MUST implicitly include the UID message data item as part of any FETCH response caused by a UID command, regardless of whether a UID was specified as a message data item to the FETCH.







Example: C: A999 UID FETCH 4827313:4828442 FLAGS
S: * 23 FETCH (FLAGS (\Seen) UID 4827313)
S: * 24 FETCH (FLAGS (\Seen) UID 4827943)
S: * 25 FETCH (FLAGS (\Seen) UID 4828442)
S: A999 UID FETCH completed

## 6.5. Client Commands - Experimental/Expansion

### 6.5.1. X<atom> Command

Arguments: implementation defined

Responses: implementation defined

Result: OK - command completed
NO - failure
BAD - command unknown or arguments invalid

Any command prefixed with an X is an experimental command. Commands which are not part of this specification, a standard or standards-track revision of this specification, or an IESG-approved experimental protocol, MUST use the X prefix.

Any added untagged responses issued by an experimental command MUST also be prefixed with an X. Server implementations MUST NOT send any such untagged responses, unless the client requested it by issuing the associated experimental command.

Example: C: a441 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=KERBEROS_V4 XPIG-LATIN
S: a441 OK CAPABILITY completed
C: A442 XPIG-LATIN
S: * XPIG-LATIN ow-nay eaking-spay ig-pay atin-lay
S: A442 OK XPIG-LATIN ompleted-cay

# 7. Server Responses

Server responses are in three forms: status responses, server data, and command continuation request. The information contained in a server response, identified by "Contents:" in the response descriptions below, is described by function, not by syntax. The precise syntax of server responses is described in the Formal Syntax section.

The client MUST be prepared to accept any response at all times.

Status responses can be tagged or untagged. Tagged status responses indicate the completion result (OK, NO, or BAD status) of a client command, and have a tag matching the command.

Some status responses, and all server data, are untagged. An untagged response is indicated by the token "*" instead of a tag. Untagged status responses indicate server greeting, or server status that does not indicate the completion of a command (for example, an impending system shutdown alert). For historical reasons, untagged server data responses are also called "unsolicited data", although strictly speaking only unilateral server data is truly "unsolicited".

Certain server data MUST be recorded by the client when it is received; this is noted in the description of that data. Such data conveys critical information which affects the interpretation of all subsequent commands and responses (e.g. updates reflecting the creation or destruction of messages).

Other server data SHOULD be recorded for later reference; if the client does not need to record the data, or if recording the data has no obvious purpose (e.g. a SEARCH response when no SEARCH command is in progress), the data SHOULD be ignored.

An example of unilateral untagged server data occurs when the IMAP connection is in selected state. In selected state, the server checks the mailbox for new messages as part of command execution. Normally, this is part of the execution of every command; hence, a NOOP command suffices to check for new messages. If new messages are found, the server sends untagged EXISTS and RECENT responses reflecting the new size of the mailbox. Server implementations that offer multiple simultaneous access to the same mailbox SHOULD also send appropriate unilateral untagged FETCH and EXPUNGE responses if another agent changes the state of any message flags or expunges any messages.

Command continuation request responses use the token "+" instead of a tag. These responses are sent by the server to indicate acceptance of an incomplete client command and readiness for the remainder of the command.

### 7.1. Server Responses - Status Responses

Status responses are OK, NO, BAD, PREAUTH and BYE. OK, NO, and BAD may be tagged or untagged. PREAUTH and BYE are always untagged.

Status responses MAY include an OPTIONAL "response code". A response code consists of data inside square brackets in the form of an atom, possibly followed by a space and arguments. The response code





contains additional information or status codes for client software beyond the OK/NO/BAD condition, and are defined when there is a specific action that a client can take based upon the additional information.

The currently defined response codes are:

| | |
|---|---|
| ALERT | The human-readable text contains a special alert that MUST be presented to the user in a fashion that calls the user's attention to the message. |
| NEWNAME | Followed by a mailbox name and a new mailbox name. A SELECT or EXAMINE is failing because the target mailbox name no longer exists because it was renamed to the new mailbox name. This is a hint to the client that the operation can succeed if the SELECT or EXAMINE is reissued with the new mailbox name. |
| PARSE | The human-readable text represents an error in parsing the [RFC-822] header or [MIME-IMB] headers of a message in the mailbox. |
| PERMANENTFLAGS | Followed by a parenthesized list of flags, indicates which of the known flags that the client can change permanently. Any flags that are in the FLAGS untagged response, but not the PERMANENTFLAGS list, can not be set permanently. If the client attempts to STORE a flag that is not in the PERMANENTFLAGS list, the server will either reject it with a NO reply or store the state for the remainder of the current session only. The PERMANENTFLAGS list can also include the special flag *, which indicates that it is possible to create new keywords by attempting to store those flags in the mailbox. |
| READ-ONLY | The mailbox is selected read-only, or its access while selected has changed from read-write to read-only. |
| READ-WRITE | The mailbox is selected read-write, or its access while selected has changed from read-only to read-write. |

TRYCREATE An APPEND or COPY attempt is failing because the target mailbox does not exist (as opposed to some other reason). This is a hint to the client that the operation can succeed if the mailbox is first created by the CREATE command.

UIDVALIDITY Followed by a decimal number, indicates the unique identifier validity value.

UNSEEN Followed by a decimal number, indicates the number of the first message without the \Seen flag set.

Additional response codes defined by particular client or server implementations SHOULD be prefixed with an "X" until they are added to a revision of this protocol. Client implementations SHOULD ignore response codes that they do not recognize.

#### 7.1.1. OK Response

Contents: OPTIONAL response code
human-readable text

The OK response indicates an information message from the server. When tagged, it indicates successful completion of the associated command. The human-readable text MAY be presented to the user as an information message. The untagged form indicates an information-only message; the nature of the information MAY be indicated by a response code.

The untagged form is also used as one of three possible greetings at connection startup. It indicates that the connection is not yet authenticated and that a LOGIN command is needed.

Example: S: * OK IMAP4rev1 server ready
C: A001 LOGIN fred blurdybloop
S: * OK [ALERT] System shutdown in 10 minutes
S: A001 OK LOGIN Completed

#### 7.1.2. NO Response

Contents: OPTIONAL response code
human-readable text

The NO response indicates an operational error message from the server. When tagged, it indicates unsuccessful completion of the associated command. The untagged form indicates a warning; the command can still complete successfully. The human-readable text describes the condition.





Example: C: A222 COPY 1:2 owatagusiam
S: * NO Disk is 98% full, please delete unnecessary data
S: A222 OK COPY completed
C: A223 COPY 3:200 blurdybloop
S: * NO Disk is 98% full, please delete unnecessary data
S: * NO Disk is 99% full, please delete unnecessary data
S: A223 NO COPY failed: disk is full

#### 7.1.3. BAD Response

Contents: OPTIONAL response code
human-readable text

The BAD response indicates an error message from the server. When tagged, it reports a protocol-level error in the client's command; the tag indicates the command that caused the error. The untagged form indicates a protocol-level error for which the associated command can not be determined; it can also indicate an internal server failure. The human-readable text describes the condition.

Example: C: ...very long command line...
S: * BAD Command line too long
C: ...empty line...
S: * BAD Empty command line
C: A443 EXPUNGE
S: * BAD Disk crash, attempting salvage to a new disk!
S: * OK Salvage successful, no data lost
S: A443 OK Expunge completed

#### 7.1.4. PREAUTH Response

Contents: OPTIONAL response code
human-readable text

The PREAUTH response is always untagged, and is one of three possible greetings at connection startup. It indicates that the connection has already been authenticated by external means and thus no LOGIN command is needed.

Example: S: * PREAUTH IMAP4rev1 server logged in as Smith

#### 7.1.5. BYE Response

Contents: OPTIONAL response code
human-readable text

The BYE response is always untagged, and indicates that the server is about to close the connection. The human-readable text MAY be displayed to the user in a status report by the client. The BYE response is sent under one of four conditions:

1) as part of a normal logout sequence. The server will close the connection after sending the tagged OK response to the LOGOUT command.

2) as a panic shutdown announcement. The server closes the connection immediately.

3) as an announcement of an inactivity autologout. The server closes the connection immediately.

4) as one of three possible greetings at connection startup, indicating that the server is not willing to accept a connection from this client. The server closes the connection immediately.

The difference between a BYE that occurs as part of a normal LOGOUT sequence (the first case) and a BYE that occurs because of a failure (the other three cases) is that the connection closes immediately in the failure case.

Example: S: * BYE Autologout; idle for too long

### 7.2. Server Responses - Server and Mailbox Status

These responses are always untagged. This is how server and mailbox status data are transmitted from the server to the client. Many of these responses typically result from a command with the same name.

#### 7.2.1. CAPABILITY Response

Contents: capability listing

The CAPABILITY response occurs as a result of a CAPABILITY command. The capability listing contains a space-separated listing of capability names that the server supports. The capability listing MUST include the atom "IMAP4rev1".

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism.





Other capability names indicate that the server supports an extension, revision, or amendment to the IMAP4rev1 protocol. Server responses MUST conform to this document until the client issues a command that uses the associated capability.

Capability names MUST either begin with "X" or be standard or standards-track IMAP4rev1 extensions, revisions, or amendments registered with IANA. A server MUST NOT offer unregistered or non-standard capability names, unless such names are prefixed with an "X".

Client implementations SHOULD NOT require any capability name other than "IMAP4rev1", and MUST ignore any unknown capability names.

Example: S: * CAPABILITY IMAP4rev1 AUTH=KERBEROS_V4 XPIG-LATIN

#### 7.2.2. LIST Response

Contents: name attributes
hierarchy delimiter
name

The LIST response occurs as a result of a LIST command. It returns a single name that matches the LIST specification. There can be multiple LIST responses for a single LIST command.

Four name attributes are defined:

\Noinferiors It is not possible for any child levels of hierarchy to exist under this name; no child levels exist now and none can be created in the future.

\Noselect It is not possible to use this name as a selectable mailbox.

\Marked The mailbox has been marked "interesting" by the server; the mailbox probably contains messages that have been added since the last time the mailbox was selected.

\Unmarked The mailbox does not contain any additional messages since the last time the mailbox was selected.

If it is not feasible for the server to determine whether the mailbox is "interesting" or not, or if the name is a \Noselect name, the server SHOULD NOT send either \Marked or \Unmarked.

The hierarchy delimiter is a character used to delimit levels of hierarchy in a mailbox name. A client can use it to create child mailboxes, and to search higher or lower levels of naming hierarchy. All children of a top-level hierarchy node MUST use the same separator character. A NIL hierarchy delimiter means that no hierarchy exists; the name is a "flat" name.

The name represents an unambiguous left-to-right hierarchy, and MUST be valid for use as a reference in LIST and LSUB commands. Unless \Noselect is indicated, the name MUST also be valid as an argument for commands, such as SELECT, that accept mailbox names.

Example: S: * LIST (\Noselect) "/" ~/Mail/foo

### 7.2.3. LSUB Response

Contents: name attributes
hierarchy delimiter
name

The LSUB response occurs as a result of an LSUB command. It returns a single name that matches the LSUB specification. There can be multiple LSUB responses for a single LSUB command. The data is identical in format to the LIST response.

Example: S: * LSUB () "." #news.comp.mail.misc

### 7.2.4 STATUS Response

Contents: name
status parenthesized list

The STATUS response occurs as a result of an STATUS command. It returns the mailbox name that matches the STATUS specification and the requested mailbox status information.

Example: S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292)

### 7.2.5. SEARCH Response

Contents: zero or more numbers





The SEARCH response occurs as a result of a SEARCH or UID SEARCH command. The number(s) refer to those messages that match the search criteria. For SEARCH, these are message sequence numbers; for UID SEARCH, these are unique identifiers. Each number is delimited by a space.

Example: S: * SEARCH 2 3 6

### 7.2.6. FLAGS Response

Contents: flag parenthesized list

The FLAGS response occurs as a result of a SELECT or EXAMINE command. The flag parenthesized list identifies the flags (at a minimum, the system-defined flags) that are applicable for this mailbox. Flags other than the system flags can also exist, depending on server implementation.

The update from the FLAGS response MUST be recorded by the client.

Example: S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)

## 7.3. Server Responses - Mailbox Size

These responses are always untagged. This is how changes in the size of the mailbox are trasnmitted from the server to the client. Immediately following the "*" token is a number that represents a message count.

### 7.3.1. EXISTS Response

Contents: none

The EXISTS response reports the number of messages in the mailbox. This response occurs as a result of a SELECT or EXAMINE command, and if the size of the mailbox changes (e.g. new mail).

The update from the EXISTS response MUST be recorded by the client.

Example: S: * 23 EXISTS

### 7.3.2. RECENT Response

Contents: none

The RECENT response reports the number of messages with the \Recent flag set. This response occurs as a result of a SELECT or EXAMINE command, and if the size of the mailbox changes (e.g. new mail).

Note: It is not guaranteed that the message sequence numbers of recent messages will be a contiguous range of the highest n messages in the mailbox (where n is the value reported by the RECENT response). Examples of situations in which this is not the case are: multiple clients having the same mailbox open (the first session to be notified will see it as recent, others will probably see it as non-recent), and when the mailbox is re-ordered by a non-IMAP agent.

The only reliable way to identify recent messages is to look at message flags to see which have the \Recent flag set, or to do a SEARCH RECENT.

The update from the RECENT response MUST be recorded by the client.

Example: S: * 5 RECENT

## 7.4. Server Responses - Message Status

These responses are always untagged. This is how message data are transmitted from the server to the client, often as a result of a command with the same name. Immediately following the "*" token is a number that represents a message sequence number.

### 7.4.1. EXPUNGE Response

Contents: none

The EXPUNGE response reports that the specified message sequence number has been permanently removed from the mailbox. The message sequence number for each successive message in the mailbox is immediately decremented by 1, and this decrement is reflected in message sequence numbers in subsequent responses (including other untagged EXPUNGE responses).

As a result of the immediate decrement rule, message sequence numbers that appear in a set of successive EXPUNGE responses depend upon whether the messages are removed starting from lower





numbers to higher numbers, or from higher numbers to lower numbers. For example, if the last 5 messages in a 9-message mailbox are expunged; a "lower to higher" server will send five untagged EXPUNGE responses for message sequence number 5, whereas a "higher to lower server" will send successive untagged EXPUNGE responses for message sequence numbers 9, 8, 7, 6, and 5.

An EXPUNGE response MUST NOT be sent when no command is in progress; nor while responding to a FETCH, STORE, or SEARCH command. This rule is necessary to prevent a loss of synchronization of message sequence numbers between client and server.

The update from the EXPUNGE response MUST be recorded by the client.

Example: S: * 44 EXPUNGE

### 7.4.2. FETCH Response

Contents: message data

The FETCH response returns data about a message to the client. The data are pairs of data item names and their values in parentheses. This response occurs as the result of a FETCH or STORE command, as well as by unilateral server decision (e.g. flag updates).

The current data items are:

BODY A form of BODYSTRUCTURE without extension data.

BODY[<section>]<<origin_octet>>
A string expressing the body contents of the specified section. The string SHOULD be interpreted by the client according to the content transfer encoding, body type, and subtype.

If the origin octet is specified, this string is a substring of the entire body contents, starting at that origin octet. This means that BODY[]<0> MAY be truncated, but BODY[] is NEVER truncated.

8-bit textual data is permitted if a [CHARSET] identifier is part of the body parameter parenthesized list for this section. Note that headers (part specifiers HEADER or MIME, or the header portion of a MESSAGE/RFC822 part), MUST be

7-bit; 8-bit characters are not permitted in headers. Note also that the blank line at the end of the header is always included in header data.

Non-textual data such as binary data MUST be transfer encoded into a textual form such as BASE64 prior to being sent to the client. To derive the original binary data, the client MUST decode the transfer encoded string.

BODYSTRUCTURE A parenthesized list that describes the [MIME-IMB] body structure of a message. This is computed by the server by parsing the [MIME-IMB] header fields, defaulting various fields as necessary.

For example, a simple text message of 48 lines and 2279 octets can have a body structure of: ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 2279 48)

Multiple parts are indicated by parenthesis nesting. Instead of a body type as the first element of the parenthesized list there is a nested body. The second element of the parenthesized list is the multipart subtype (mixed, digest, parallel, alternative, etc.).

For example, a two part message consisting of a text and a BASE645-encoded text attachment can have a body structure of: (("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 1152 23)("TEXT" "PLAIN" ("CHARSET" "US-ASCII" "NAME" "cc.diff") "<960723163407.20117h@cac.washington.edu>" "Compiler diff" "BASE64" 4554 73) "MIXED"))

Extension data follows the multipart subtype. Extension data is never returned with the BODY fetch, but can be returned with a BODYSTRUCTURE fetch. Extension data, if present, MUST be in the defined order.

The extension data of a multipart body part are in the following order:

body parameter parenthesized list
   A parenthesized list of attribute/value pairs [e.g. ("foo" "bar" "baz" "rag") where "bar" is the value of "foo" and "rag" is the value of "baz"] as defined in [MIME-IMB].

body disposition
   A parenthesized list, consisting of a disposition type string followed by a parenthesized list of disposition attribute/value pairs. The disposition type and attribute names will be defined in a future standards-track revision to [DISPOSITION].

body language
   A string or parenthesized list giving the body language value as defined in [LANGUAGE-TAGS].

Any following extension data are not yet defined in this version of the protocol. Such extension data can consist of zero or more NILs, strings, numbers, or potentially nested parenthesized lists of such data. Client implementations that do a BODYSTRUCTURE fetch MUST be prepared to accept such extension data. Server implementations MUST NOT send such extension data until it has been defined by a revision of this protocol.

The basic fields of a non-multipart body part are in the following order:

body type
   A string giving the content media type name as defined in [MIME-IMB].

body subtype
   A string giving the content subtype name as defined in [MIME-IMB].

body parameter parenthesized list
   A parenthesized list of attribute/value pairs [e.g. ("foo" "bar" "baz" "rag") where "bar" is the value of "foo" and "rag" is the value of "baz"] as defined in [MIME-IMB].

body id
   A string giving the content id as defined in [MIME-IMB].

body description
   A string giving the content description as defined in [MIME-IMB].

body encoding
A string giving the content transfer encoding as defined in [MIME-IMB].

body size
A number giving the size of the body in octets. Note that this size is the size in its transfer encoding and not the resulting size after any decoding.

A body type of type MESSAGE and subtype RFC822 contains, immediately after the basic fields, the envelope structure, body structure, and size in text lines of the encapsulated message.

A body type of type TEXT contains, immediately after the basic fields, the size of the body in text lines. Note that this size is the size in its content transfer encoding and not the resulting size after any decoding.

Extension data follows the basic fields and the type-specific fields listed above. Extension data is never returned with the BODY fetch, but can be returned with a BODYSTRUCTURE fetch. Extension data, if present, MUST be in the defined order.

The extension data of a non-multipart body part are in the following order:

body MD5
A string giving the body MD5 value as defined in [MD5].

body disposition
A parenthesized list with the same content and function as the body disposition for a multipart body part.

body language
A string or parenthesized list giving the body language value as defined in [LANGUAGE-TAGS].

Any following extension data are not yet defined in this version of the protocol, and would be as described above under multipart extension data.







ENVELOPE — A parenthesized list that describes the envelope structure of a message. This is computed by the server by parsing the [RFC-822] header into the component parts, defaulting various fields as necessary.

The fields of the envelope structure are in the following order: date, subject, from, sender, reply-to, to, cc, bcc, in-reply-to, and message-id. The date, subject, in-reply-to, and message-id fields are strings. The from, sender, reply-to, to, cc, and bcc fields are parenthesized lists of address structures.

An address structure is a parenthesized list that describes an electronic mail address. The fields of an address structure are in the following order: personal name, [SMTP] at-domain-list (source route), mailbox name, and host name.

[RFC-822] group syntax is indicated by a special form of address structure in which the host name field is NIL. If the mailbox name field is also NIL, this is an end of group marker (semi-colon in RFC 822 syntax). If the mailbox name field is non-NIL, this is a start of group marker, and the mailbox name field holds the group name phrase.

Any field of an envelope or address structure that is not applicable is presented as NIL. Note that the server MUST default the reply-to and sender fields from the from field; a client is not expected to know to do this.

FLAGS — A parenthesized list of flags that are set for this message.

INTERNALDATE — A string representing the internal date of the message.

RFC822 — Equivalent to BODY[].

RFC822.HEADER — Equivalent to BODY.PEEK[HEADER].

RFC822.SIZE — A number expressing the [RFC-822] size of the message.

RFC822.TEXT — Equivalent to BODY[TEXT].

UID A number expressing the unique identifier of the message.

Example: S: * 23 FETCH (FLAGS (\Seen) RFC822.SIZE 44827)

### 7.5. Server Responses - Command Continuation Request

The command continuation request response is indicated by a "+" token instead of a tag. This form of response indicates that the server is ready to accept the continuation of a command from the client. The remainder of this response is a line of text.

This response is used in the AUTHORIZATION command to transmit server data to the client, and request additional client data. This response is also used if an argument to any command is a literal.

The client is not permitted to send the octets of the literal unless the server indicates that it expects it. This permits the server to process commands and reject errors on a line-by-line basis. The remainder of the command, including the CRLF that terminates a command, follows the octets of the literal. If there are any additional command arguments the literal octets are followed by a space and those arguments.

Example:
```
C: A001 LOGIN {11}
S: + Ready for additional command text
C: FRED FOOBAR {7}
S: + Ready for additional command text
C: fat man
S: A001 OK LOGIN completed
C: A044 BLURDYBLOOP {102856}
S: A044 BAD No such command as "BLURDYBLOOP"
```

## 8. Sample IMAP4rev1 connection

The following is a transcript of an IMAP4rev1 connection. A long line in this sample is broken for editorial clarity.

```
S:   * OK IMAP4rev1 Service Ready
C:   a001 login mrc secret
S:   a001 OK LOGIN completed
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * 2 RECENT
S:   * OK [UNSEEN 17] Message 17 is the first unseen message
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
```





```
S:   a002 OK [READ-WRITE] SELECT completed
C:   a003 fetch 12 full
S:   * 12 FETCH (FLAGS (\Seen) INTERNALDATE "17-Jul-1996 02:44:25 -0700"
      RFC822.SIZE 4286 ENVELOPE ("Wed, 17 Jul 1996 02:23:25 -0700 (PDT)"
      "IMAP4rev1 WG mtg summary and minutes"
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      ((NIL NIL "imap" "cac.washington.edu"))
      ((NIL NIL "minutes" "CNRI.Reston.VA.US")
      ("John Klensin" NIL "KLENSIN" "INFOODS.MIT.EDU")) NIL NIL
      "<B27397-0100000@cac.washington.edu>")
       BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028 92))
S:   a003 OK FETCH completed
C:   a004 fetch 12 body[header]
S:   * 12 FETCH (BODY[HEADER] {350}
S:   Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:   From: Terry Gray <gray@cac.washington.edu>
S:   Subject: IMAP4rev1 WG mtg summary and minutes
S:   To: imap@cac.washington.edu
S:   cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@INFOODS.MIT.EDU>
S:   Message-Id: <B27397-0100000@cac.washington.edu>
S:   MIME-Version: 1.0
S:   Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
S:
S:   )
S:   a004 OK FETCH completed
C:   a005 store 12 +flags \deleted
S:   * 12 FETCH (FLAGS (\Seen \Deleted))
S:   a005 OK +FLAGS completed
C:   a006 logout
S:   * BYE IMAP4rev1 server terminating connection
S:   a006 OK LOGOUT completed
```

## 9. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) notation as specified in [RFC-822] with one exception; the delimiter used with the "#" construct is a single space (SPACE) and not one or more commas.

In the case of alternative or optional rules in which a later rule overlaps an earlier rule, the rule which is listed earlier MUST take priority. For example, "\Seen" when parsed as a flag is the \Seen flag name and not a flag_extension, even though "\Seen" could be parsed as a flag_extension. Some, but not all, instances of this rule are noted below.

Except as noted otherwise, all alphabetic characters are case-insensitive. The use of upper or lower case characters to define token strings is for editorial clarity only. Implementations MUST accept these strings in a case-insensitive fashion.

```
address         ::= "(" addr_name SPACE addr_adl SPACE addr_mailbox
                    SPACE addr_host ")"

addr_adl        ::= nstring
                    ;; Holds route from [RFC-822] route-addr if
                    ;; non-NIL

addr_host       ::= nstring
                    ;; NIL indicates [RFC-822] group syntax.
                    ;; Otherwise, holds [RFC-822] domain name

addr_mailbox    ::= nstring
                    ;; NIL indicates end of [RFC-822] group; if
                    ;; non-NIL and addr_host is NIL, holds
                    ;; [RFC-822] group name.
                    ;; Otherwise, holds [RFC-822] local-part

addr_name       ::= nstring
                    ;; Holds phrase from [RFC-822] mailbox if
                    ;; non-NIL

alpha           ::= "A" / "B" / "C" / "D" / "E" / "F" / "G" / "H" /
                    "I" / "J" / "K" / "L" / "M" / "N" / "O" / "P" /
                    "Q" / "R" / "S" / "T" / "U" / "V" / "W" / "X" /
                    "Y" / "Z" /
                    "a" / "b" / "c" / "d" / "e" / "f" / "g" / "h" /
                    "i" / "j" / "k" / "l" / "m" / "n" / "o" / "p" /
                    "q" / "r" / "s" / "t" / "u" / "v" / "w" / "x" /
                    "y" / "z"
                    ;; Case-sensitive

append          ::= "APPEND" SPACE mailbox [SPACE flag_list]
                    [SPACE date_time] SPACE literal

astring         ::= atom / string

atom            ::= 1*ATOM_CHAR

ATOM_CHAR       ::= <any CHAR except atom_specials>

atom_specials   ::= "(" / ")" / "{" / SPACE / CTL / list_wildcards /
                    quoted_specials
```





```
authenticate    ::= "AUTHENTICATE" SPACE auth_type *(CRLF base64)

auth_type       ::= atom
                    ;; Defined by [IMAP-AUTH]

base64          ::= *(4base64_char) [base64_terminal]

base64_char     ::= alpha / digit / "+" / "/"

base64_terminal ::= (2base64_char "==") / (3base64_char "=")

body            ::= "(" body_type_1part / body_type_mpart ")"

body_extension  ::= nstring / number / "(" 1#body_extension ")"
                    ;; Future expansion. Client implementations
                    ;; MUST accept body_extension fields. Server
                    ;; implementations MUST NOT generate
                    ;; body_extension fields except as defined by
                    ;; future standard or standards-track
                    ;; revisions of this specification.

body_ext_1part  ::= body_fld_md5 [SPACE body_fld_dsp
                    [SPACE body_fld_lang
                    [SPACE 1#body_extension]]]
                    ;; MUST NOT be returned on non-extensible
                    ;; "BODY" fetch

body_ext_mpart  ::= body_fld_param
                    [SPACE body_fld_dsp SPACE body_fld_lang
                    [SPACE 1#body_extension]]
                    ;; MUST NOT be returned on non-extensible
                    ;; "BODY" fetch

body_fields     ::= body_fld_param SPACE body_fld_id SPACE
                    body_fld_desc SPACE body_fld_enc SPACE
                    body_fld_octets

body_fld_desc   ::= nstring

body_fld_dsp    ::= "(" string SPACE body_fld_param ")" / nil

body_fld_enc    ::= (<"> ("7BIT" / "8BIT" / "BINARY" / "BASE64"/
                    "QUOTED-PRINTABLE") <">) / string

body_fld_id     ::= nstring

body_fld_lang   ::= nstring / "(" 1#string ")"
```

```
body_fld_lines  ::= number

body_fld_md5    ::= nstring

body_fld_octets ::= number

body_fld_param  ::= "(" 1#(string SPACE string) ")" / nil

body_type_1part ::= (body_type_basic / body_type_msg / body_type_text)
                    [SPACE body_ext_1part]

body_type_basic ::= media_basic SPACE body_fields
                    ;; MESSAGE subtype MUST NOT be "RFC822"

body_type_mpart ::= 1*body SPACE media_subtype
                    [SPACE body_ext_mpart]

body_type_msg   ::= media_message SPACE body_fields SPACE envelope
                    SPACE body SPACE body_fld_lines

body_type_text  ::= media_text SPACE body_fields SPACE body_fld_lines

capability      ::= "AUTH=" auth_type / atom
                    ;; New capabilities MUST begin with "X" or be
                    ;; registered with IANA as standard or
                    ;; standards-track

capability_data ::= "CAPABILITY" SPACE [1#capability SPACE] "IMAP4rev1"
                    [SPACE 1#capability]
                    ;; IMAP4rev1 servers which offer RFC 1730
                    ;; compatibility MUST list "IMAP4" as the first
                    ;; capability.

CHAR            ::= <any 7-bit US-ASCII character except NUL,
                     0x01 - 0x7f>

CHAR8           ::= <any 8-bit octet except NUL, 0x01 - 0xff>

command         ::= tag SPACE (command_any / command_auth /
                    command_nonauth / command_select) CRLF
                    ;; Modal based on state

command_any     ::= "CAPABILITY" / "LOGOUT" / "NOOP" / x_command
                    ;; Valid in all states

command_auth    ::= append / create / delete / examine / list / lsub /
                    rename / select / status / subscribe / unsubscribe
                    ;; Valid only in Authenticated or Selected state
```





```
command_nonauth ::= login / authenticate
                    ;; Valid only when in Non-Authenticated state

command_select  ::= "CHECK" / "CLOSE" / "EXPUNGE" /
                     copy / fetch / store / uid / search
                    ;; Valid only when in Selected state

continue_req    ::= "+" SPACE (resp_text / base64)

copy            ::= "COPY" SPACE set SPACE mailbox

CR              ::= <ASCII CR, carriage return, 0x0D>

create          ::= "CREATE" SPACE mailbox
                    ;; Use of INBOX gives a NO error

CRLF            ::= CR LF

CTL             ::= <any ASCII control character and DEL,
                         0x00 - 0x1f, 0x7f>

date            ::= date_text / <"> date_text <">

date_day        ::= 1*2digit
                    ;; Day of month

date_day_fixed  ::= (SPACE digit) / 2digit
                    ;; Fixed-format version of date_day

date_month      ::= "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
                    "Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"

date_text       ::= date_day "-" date_month "-" date_year

date_year       ::= 4digit

date_time       ::= <"> date_day_fixed "-" date_month "-" date_year
                    SPACE time SPACE zone <">

delete          ::= "DELETE" SPACE mailbox
                    ;; Use of INBOX gives a NO error

digit           ::= "0" / digit_nz

digit_nz        ::= "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" /
                    "9"
```

```
envelope        ::= "(" env_date SPACE env_subject SPACE env_from
                    SPACE env_sender SPACE env_reply_to SPACE env_to
                    SPACE env_cc SPACE env_bcc SPACE env_in_reply_to
                    SPACE env_message_id ")"

env_bcc         ::= "(" 1*address ")" / nil

env_cc          ::= "(" 1*address ")" / nil

env_date        ::= nstring

env_from        ::= "(" 1*address ")" / nil

env_in_reply_to ::= nstring

env_message_id  ::= nstring

env_reply_to    ::= "(" 1*address ")" / nil

env_sender      ::= "(" 1*address ")" / nil

env_subject     ::= nstring

env_to          ::= "(" 1*address ")" / nil

examine         ::= "EXAMINE" SPACE mailbox

fetch           ::= "FETCH" SPACE set SPACE ("ALL" / "FULL" /
                    "FAST" / fetch_att / "(" 1#fetch_att ")")

fetch_att       ::= "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
                    "RFC822" [".HEADER" / ".SIZE" / ".TEXT"] /
                    "BODY" ["STRUCTURE"] / "UID" /
                    "BODY" [".PEEK"] section
                    ["<" number "." nz_number ">"]

flag            ::= "\Answered" / "\Flagged" / "\Deleted" /
                    "\Seen" / "\Draft" / flag_keyword / flag_extension

flag_extension  ::= "\" atom
                    ;; Future expansion.  Client implementations
                    ;; MUST accept flag_extension flags.  Server
                    ;; implementations MUST NOT generate
                    ;; flag_extension flags except as defined by
                    ;; future standard or standards-track
                    ;; revisions of this specification.

flag_keyword    ::= atom
```







```
flag_list       ::= "(" #flag ")"

greeting        ::= "*" SPACE (resp_cond_auth / resp_cond_bye) CRLF

header_fld_name ::= astring

header_list     ::= "(" 1#header_fld_name ")"

LF              ::= <ASCII LF, line feed, 0x0A>

list            ::= "LIST" SPACE mailbox SPACE list_mailbox

list_mailbox    ::= 1*(ATOM_CHAR / list_wildcards) / string

list_wildcards  ::= "%" / "*"

literal         ::= "{" number "}" CRLF *CHAR8
                    ;; Number represents the number of CHAR8 octets

login           ::= "LOGIN" SPACE userid SPACE password

lsub            ::= "LSUB" SPACE mailbox SPACE list_mailbox

mailbox         ::= "INBOX" / astring
                    ;; INBOX is case-insensitive.  All case variants of
                    ;; INBOX (e.g. "iNbOx") MUST be interpreted as INBOX
                    ;; not as an astring.  Refer to section 5.1 for
                    ;; further semantic details of mailbox names.

mailbox_data    ::=  "FLAGS" SPACE flag_list /
                     "LIST" SPACE mailbox_list /
                     "LSUB" SPACE mailbox_list /
                     "MAILBOX" SPACE text /
                     "SEARCH" [SPACE 1#nz_number] /
                     "STATUS" SPACE mailbox SPACE
                     "(" #<status_att number ")" /
                     number SPACE "EXISTS" / number SPACE "RECENT"

mailbox_list    ::= "(" #("\Marked" / "\Noinferiors" /
                    "\Noselect" / "\Unmarked" / flag_extension) ")"
                    SPACE (<"> QUOTED_CHAR <"> / nil) SPACE mailbox

media_basic     ::= (<"> ("APPLICATION" / "AUDIO" / "IMAGE" /
                    "MESSAGE" / "VIDEO") <">) / string)
                    SPACE media_subtype
                    ;; Defined in [MIME-IMT]

media_message   ::= <"> "MESSAGE" <"> SPACE <"> "RFC822" <">
```

```
                    ;; Defined in [MIME-IMT]

media_subtype   ::= string
                    ;; Defined in [MIME-IMT]

media_text      ::= <"> "TEXT" <"> SPACE media_subtype
                    ;; Defined in [MIME-IMT]

message_data    ::= nz_number SPACE ("EXPUNGE" /
                                     ("FETCH" SPACE msg_att))

msg_att         ::= "(" 1#("ENVELOPE" SPACE envelope /
                    "FLAGS" SPACE "(" #(flag / "\Recent") ")" /
                    "INTERNALDATE" SPACE date_time /
                    "RFC822" [".HEADER" / ".TEXT"] SPACE nstring /
                    "RFC822.SIZE" SPACE number /
                    "BODY" ["STRUCTURE"] SPACE body /
                    "BODY" section ["<" number ">"] SPACE nstring /
                    "UID" SPACE uniqueid) ")"

nil             ::= "NIL"

nstring         ::= string / nil

number          ::= 1*digit
                    ;; Unsigned 32-bit integer
                    ;; (0 <= n < 4,294,967,296)

nz_number       ::= digit_nz *digit
                    ;; Non-zero unsigned 32-bit integer
                    ;; (0 < n < 4,294,967,296)

password        ::= astring

quoted          ::= <"> *QUOTED_CHAR <">

QUOTED_CHAR     ::= <any TEXT_CHAR except quoted_specials> /
                    "\" quoted_specials

quoted_specials ::= <"> / "\"

rename          ::= "RENAME" SPACE mailbox SPACE mailbox
                    ;; Use of INBOX as a destination gives a NO error

response        ::= *(continue_req / response_data) response_done

response_data   ::= "*" SPACE (resp_cond_state / resp_cond_bye /
                    mailbox_data / message_data / capability_data)
```





```
                    CRLF

response_done   ::= response_tagged / response_fatal

response_fatal  ::= "*" SPACE resp_cond_bye CRLF
                    ;; Server closes connection immediately

response_tagged ::= tag SPACE resp_cond_state CRLF

resp_cond_auth  ::= ("OK" / "PREAUTH") SPACE resp_text
                    ;; Authentication condition

resp_cond_bye   ::= "BYE" SPACE resp_text

resp_cond_state ::= ("OK" / "NO" / "BAD") SPACE resp_text
                    ;; Status condition

resp_text       ::= ["[" resp_text_code "]" SPACE] (text_mime2 / text)
                    ;; text SHOULD NOT begin with "[" or "="

resp_text_code  ::= "ALERT" / "PARSE" /
                    "PERMANENTFLAGS" SPACE "(" #(flag / "\*") ")" /
                    "READ-ONLY" / "READ-WRITE" / "TRYCREATE" /
                    "UIDVALIDITY" SPACE nz_number /
                    "UNSEEN" SPACE nz_number /
                    atom [SPACE 1*<any TEXT_CHAR except "]">]

search          ::= "SEARCH" SPACE ["CHARSET" SPACE astring SPACE]
                    1#search_key
                    ;; [CHARSET] MUST be registered with IANA

search_key      ::= "ALL" / "ANSWERED" / "BCC" SPACE astring /
                    "BEFORE" SPACE date / "BODY" SPACE astring /
                    "CC" SPACE astring / "DELETED" / "FLAGGED" /
                    "FROM" SPACE astring /
                    "KEYWORD" SPACE flag_keyword / "NEW" / "OLD" /
                    "ON" SPACE date / "RECENT" / "SEEN" /
                    "SINCE" SPACE date / "SUBJECT" SPACE astring /
                    "TEXT" SPACE astring / "TO" SPACE astring /
                    "UNANSWERED" / "UNDELETED" / "UNFLAGGED" /
                    "UNKEYWORD" SPACE flag_keyword / "UNSEEN" /
                    ;; Above this line were in [IMAP2]
                    "DRAFT" /
                    "HEADER" SPACE header_fld_name SPACE astring /
                    "LARGER" SPACE number / "NOT" SPACE search_key /
                    "OR" SPACE search_key SPACE search_key /
                    "SENTBEFORE" SPACE date / "SENTON" SPACE date /
                    "SENTSINCE" SPACE date / "SMALLER" SPACE number /
```

```
                    "UID" SPACE set / "UNDRAFT" / set /
                    "(" 1#search_key ")"

section         ::= "[" [section_text / (nz_number *["." nz_number]
                    ["." (section_text / "MIME")])] "]"

section_text    ::= "HEADER" / "HEADER.FIELDS" [".NOT"]
                    SPACE header_list / "TEXT"

select          ::= "SELECT" SPACE mailbox

sequence_num    ::= nz_number / "*"
                    ;; * is the largest number in use.  For message
                    ;; sequence numbers, it is the number of messages
                    ;; in the mailbox.  For unique identifiers, it is
                    ;; the unique identifier of the last message in
                    ;; the mailbox.

set             ::= sequence_num / (sequence_num ":" sequence_num) /
                    (set "," set)
                    ;; Identifies a set of messages.  For message
                    ;; sequence numbers, these are consecutive
                    ;; numbers from 1 to the number of messages in
                    ;; the mailbox
                    ;; Comma delimits individual numbers, colon
                    ;; delimits between two numbers inclusive.
                    ;; Example: 2,4:7,9,12:* is 2,4,5,6,7,9,12,13,
                    ;; 14,15 for a mailbox with 15 messages.

SPACE           ::= <ASCII SP, space, 0x20>

status          ::= "STATUS" SPACE mailbox SPACE "(" 1#status_att ")"

status_att      ::= "MESSAGES" / "RECENT" / "UIDNEXT" / "UIDVALIDITY" /
                    "UNSEEN"

store           ::= "STORE" SPACE set SPACE store_att_flags

store_att_flags ::= (["+" / "-"] "FLAGS" [".SILENT"]) SPACE
                    (flag_list / #flag)

string          ::= quoted / literal

subscribe       ::= "SUBSCRIBE" SPACE mailbox

tag             ::= 1*<any ATOM_CHAR except "+">

text            ::= 1*TEXT_CHAR
```





```
text_mime2      ::= "=?" <charset> "?" <encoding> "?"
                    <encoded-text> "?="
                    ;; Syntax defined in [MIME-HDRS]

TEXT_CHAR       ::= <any CHAR except CR and LF>

time            ::= 2digit ":" 2digit ":" 2digit
                    ;; Hours minutes seconds

uid             ::= "UID" SPACE (copy / fetch / search / store)
                    ;; Unique identifiers used instead of message
                    ;; sequence numbers

uniqueid        ::= nz_number
                    ;; Strictly ascending

unsubscribe     ::= "UNSUBSCRIBE" SPACE mailbox

userid          ::= astring

x_command       ::= "X" atom <experimental command arguments>

zone            ::= ("+" / "-") 4digit
                    ;; Signed four-digit value of hhmm representing
                    ;; hours and minutes west of Greenwich (that is,
                    ;; (the amount that the given time differs from
                    ;; Universal Time).  Subtracting the timezone
                    ;; from the given time will give the UT form.
                    ;; The Universal Time zone is "+0000".
```

## 10. Author's Note

This document is a revision or rewrite of earlier documents, and supercedes the protocol specification in those documents: RFC 1730, unpublished IMAP2bis.TXT document, RFC 1176, and RFC 1064.

## 11. Security Considerations

IMAP4rev1 protocol transactions, including electronic mail data, are sent in the clear over the network unless privacy protection is negotiated in the AUTHENTICATE command.

A server error message for an AUTHENTICATE command which fails due to invalid credentials SHOULD NOT detail why the credentials are invalid.

Use of the LOGIN command sends passwords in the clear. This can be avoided by using the AUTHENTICATE command instead.

A server error message for a failing LOGIN command SHOULD NOT specify that the user name, as opposed to the password, is invalid.

Additional security considerations are discussed in the section discussing the AUTHENTICATE and LOGIN commands.

## 12. Author's Address

Mark R. Crispin
Networks and Distributed Computing
University of Washington
4545 15th Aveneue NE
Seattle, WA 98105-4527

Phone: (206) 543-5762

EMail: MRC@CAC.Washington.EDU







Appendices

## A. References

[ACAP] Myers, J. "ACAP -- Application Configuration Access Protocol", Work in Progress.

[CHARSET] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[DISPOSITION] Troost, R., and Dorner, S., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header", RFC 1806, June 1995.

[IMAP-AUTH] Myers, J., "IMAP4 Authentication Mechanism", RFC 1731. Carnegie-Mellon University, December 1994.

[IMAP-COMPAT] Crispin, M., "IMAP4 Compatibility with IMAP2bis", RFC 2061, University of Washington, November 1996.

[IMAP-DISC] Austein, R., "Synchronization Operations for Disconnected IMAP4 Clients", Work in Progress.

[IMAP-HISTORICAL] Crispin, M. "IMAP4 Compatibility with IMAP2 and IMAP2bis", RFC 1732, University of Washington, December 1994.

[IMAP-MODEL] Crispin, M., "Distributed Electronic Mail Models in IMAP4", RFC 1733, University of Washington, December 1994.

[IMAP-OBSOLETE] Crispin, M., "Internet Message Access Protocol - Obsolete Syntax", RFC 2062, University of Washington, November 1996.

[IMAP2] Crispin, M., "Interactive Mail Access Protocol - Version 2", RFC 1176, University of Washington, August 1990.

[LANGUAGE-TAGS] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[MD5] Myers, J., and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995.

[MIME-IMB] Freed, N., and N. Borenstein, "MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[MIME-IMT] Freed, N., and N. Borenstein, "MIME (Multipurpose Internet Mail Extensions) Part Two: Media Types", RFC 2046, November 1996.

[MIME-HDRS] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[RFC-822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, University of Delaware, August 1982.

[SMTP] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[UTF-7] Goldsmith, D., and Davis, M., "UTF-7: A Mail-Safe Transformation Format of Unicode", RFC 1642, July 1994.

## B. Changes from RFC 1730

1) The STATUS command has been added.

2) Clarify in the formal syntax that the "#" construct can never refer to multiple spaces.

3) Obsolete syntax has been moved to a separate document.

4) The PARTIAL command has been obsoleted.

5) The RFC822.HEADER.LINES, RFC822.HEADER.LINES.NOT, RFC822.PEEK, and RFC822.TEXT.PEEK fetch attributes have been obsoleted.

6) The "<" origin "." size ">" suffix for BODY text attributes has been added.

7) The HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT, MIME, and TEXT part specifiers have been added.

8) Support for Content-Disposition and Content-Language has been added.

9) The restriction on fetching nested MULTIPART parts has been removed.

10) Body part number 0 has been obsoleted.

11) Server-supported authenticators are now identified by capabilities.





12) The capability that identifies this protocol is now called "IMAP4rev1". A server that provides backwards support for RFC 1730 SHOULD emit the "IMAP4" capability in addition to "IMAP4rev1" in its CAPABILITY response. Because RFC-1730 required "IMAP4" to appear as the first capability, it MUST listed first in the response.

13) A description of the mailbox name namespace convention has been added.

14) A description of the international mailbox name convention has been added.

15) The UID-NEXT and UID-VALIDITY status items are now called UIDNEXT and UIDVALIDITY. This is a change from the IMAP STATUS Work in Progress and not from RFC-1730

16) Add a clarification that a null mailbox name argument to the LIST command returns an untagged LIST response with the hierarchy delimiter and root of the reference argument.

17) Define terms such as "MUST", "SHOULD", and "MUST NOT".

18) Add a section which defines message attributes and more thoroughly details the semantics of message sequence numbers, UIDs, and flags.

19) Add a clarification detailing the circumstances when a client may send multiple commands without waiting for a response, and the circumstances in which ambiguities may result.

20) Add a recommendation on server behavior for DELETE and RENAME when inferior hierarchical names of the given name exist.

21) Add a clarification that a mailbox name may not be unilaterally unsubscribed by the server, even if that mailbox name no longer exists.

22) Add a clarification that LIST should return its results quickly without undue delay.

23) Add a clarification that the date_time argument to APPEND sets the internal date of the message.

24) Add a clarification on APPEND behavior when the target mailbox is the currently selected mailbox.

25) Add a clarification that external changes to flags should be always announced via an untagged FETCH even if the current command is a STORE with the ".SILENT" suffix.

26) Add a clarification that COPY appends to the target mailbox.

27) Add the NEWNAME response code.

28) Rewrite the description of the untagged BYE response to clarify its semantics.

29) Change the reference for the body MD5 to refer to the proper RFC.

30) Clarify that the formal syntax contains rules which may overlap, and that in the event of such an overlap the rule which occurs first takes precedence.

31) Correct the definition of body_fld_param.

32) More formal syntax for capability_data.

33) Clarify that any case variant of "INBOX" must be interpreted as INBOX.

34) Clarify that the human-readable text in resp_text should not begin with "[" or "=".

35) Change MIME references to Draft Standard documents.

36) Clarify \Recent semantics.

37) Additional examples.

## C. Key Word Index

Network Working Group
Request for Comments: 2251
Category: Standards Track

M. Wahl
Critical Angle Inc.
T. Howes
Netscape Communications Corp.
S. Kille
Isode Limited
December 1997

# Lightweight Directory Access Protocol (v3)

## 1. Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## Copyright Notice



## IESG Note

This document describes a directory access protocol that provides both read and update access. Update access requires secure authentication, but this document does not mandate implementation of any satisfactory authentication mechanisms.

In accordance with RFC 2026, section 4.4.1, this specification is being approved by IESG as a Proposed Standard despite this limitation, for the following reasons:

a. to encourage implementation and interoperability testing of these protocols (with or without update access) before they are deployed, and

b. to encourage deployment and use of these protocols in read-only applications. (e.g. applications where LDAPv3 is used as a query language for directories which are updated by some secure mechanism other than LDAP), and

c. to avoid delaying the advancement and deployment of other Internet standards-track protocols which require the ability to query, but not update, LDAPv3 directory servers.





Readers are hereby warned that until mandatory authentication mechanisms are standardized, clients and servers written according to this specification which make use of update functionality are UNLIKELY TO INTEROPERATE, or MAY INTEROPERATE ONLY IF AUTHENTICATION IS REDUCED TO AN UNACCEPTABLY WEAK LEVEL.

Implementors are hereby discouraged from deploying LDAPv3 clients or servers which implement the update functionality, until a Proposed Standard for mandatory authentication in LDAPv3 has been approved and published as an RFC.

Table of Contents

## 2. Abstract

The protocol described in this document is designed to provide access to directories supporting the X.500 models, while not incurring the resource requirements of the X.500 Directory Access Protocol (DAP). This protocol is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. When used with a directory supporting the X.500 protocols, it is intended to be a complement to the X.500 DAP.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119 [10].

Key aspects of this version of LDAP are:

- All protocol elements of LDAPv2 (RFC 1777) are supported. The protocol is carried directly over TCP or other transport, bypassing much of the session/presentation overhead of X.500 DAP.

- Most protocol data elements can be encoded as ordinary strings (e.g., Distinguished Names).





- Referrals to other servers may be returned.

- SASL mechanisms may be used with LDAP to provide association security services.

- Attribute values and Distinguished Names have been internationalized through the use of the ISO 10646 character set.

- The protocol can be extended to support new operations, and controls may be used to extend existing operations.

- Schema is published in the directory for use by clients.

## 3. Models

Interest in X.500 [1] directory technologies in the Internet has led to efforts to reduce the high cost of entry associated with use of these technologies. This document continues the efforts to define directory protocol alternatives, updating the LDAP [2] protocol specification.

### 3.1. Protocol Model

The general model adopted by this protocol is one of clients performing protocol operations against servers. In this model, a client transmits a protocol request describing the operation to be performed to a server. The server is then responsible for performing the necessary operation(s) in the directory. Upon completion of the operation(s), the server returns a response containing any results or errors to the requesting client.

In keeping with the goal of easing the costs associated with use of the directory, it is an objective of this protocol to minimize the complexity of clients so as to facilitate widespread deployment of applications capable of using the directory.

Note that although servers are required to return responses whenever such responses are defined in the protocol, there is no requirement for synchronous behavior on the part of either clients or servers. Requests and responses for multiple operations may be exchanged between a client and server in any order, provided the client eventually receives a response for every request that requires one.

In LDAP versions 1 and 2, no provision was made for protocol servers returning referrals to clients. However, for improved performance and distribution this version of the protocol permits servers to return to clients referrals to other servers. This allows servers to offload the work of contacting other servers to progress operations.

Note that the core protocol operations defined in this document can be mapped to a strict subset of the X.500(1997) directory abstract service, so it can be cleanly provided by the DAP. However there is not a one-to-one mapping between LDAP protocol operations and DAP operations: server implementations acting as a gateway to X.500 directories may need to make multiple DAP requests.

### 3.2. Data Model

This section provides a brief introduction to the X.500 data model, as used by LDAP.

The LDAP protocol assumes there are one or more servers which jointly provide access to a Directory Information Tree (DIT). The tree is made up of entries. Entries have names: one or more attribute values from the entry form its relative distinguished name (RDN), which MUST be unique among all its siblings. The concatenation of the relative distinguished names of the sequence of entries from a particular entry to an immediate subordinate of the root of the tree forms that entry's Distinguished Name (DN), which is unique in the tree. An example of a Distinguished Name is

CN=Steve Kille, O=Isode Limited, C=GB

Some servers may hold cache or shadow copies of entries, which can be used to answer search and comparison queries, but will return referrals or contact other servers if modification operations are requested.

Servers which perform caching or shadowing MUST ensure that they do not violate any access control constraints placed on the data by the originating server.

The largest collection of entries, starting at an entry that is mastered by a particular server, and including all its subordinates and their subordinates, down to the entries which are mastered by different servers, is termed a naming context. The root of the DIT is a DSA-specific Entry (DSE) and not part of any naming context: each server has different attribute values in the root DSE. (DSA is an X.500 term for the directory server).

#### 3.2.1. Attributes of Entries

Entries consist of a set of attributes. An attribute is a type with one or more associated values. The attribute type is identified by a short descriptive name and an OID (object identifier). The attribute





type governs whether there can be more than one value of an attribute of that type in an entry, the syntax to which the values must conform, the kinds of matching which can be performed on values of that attribute, and other functions.

An example of an attribute is "mail". There may be one or more values of this attribute, they must be IA5 (ASCII) strings, and they are case insensitive (e.g. "foo@bar.com" will match "FOO@BAR.COM").

Schema is the collection of attribute type definitions, object class definitions and other information which a server uses to determine how to match a filter or attribute value assertion (in a compare operation) against the attributes of an entry, and whether to permit add and modify operations. The definition of schema for use with LDAP is given in [5] and [6]. Additional schema elements may be defined in other documents.

Each entry MUST have an objectClass attribute. The objectClass attribute specifies the object classes of an entry, which along with the system and user schema determine the permitted attributes of an entry. Values of this attribute may be modified by clients, but the objectClass attribute cannot be removed. Servers may restrict the modifications of this attribute to prevent the basic structural class of the entry from being changed (e.g. one cannot change a person into a country). When creating an entry or adding an objectClass value to an entry, all superclasses of the named classes are implicitly added as well if not already present, and the client must supply values for any mandatory attributes of new superclasses.

Some attributes, termed operational attributes, are used by servers for administering the directory system itself. They are not returned in search results unless explicitly requested by name. Attributes which are not operational, such as "mail", will have their schema and syntax constraints enforced by servers, but servers will generally not make use of their values.

Servers MUST NOT permit clients to add attributes to an entry unless those attributes are permitted by the object class definitions, the schema controlling that entry (specified in the subschema - see below), or are operational attributes known to that server and used for administrative purposes. Note that there is a particular objectClass 'extensibleObject' defined in [5] which permits all user attributes to be present in an entry.

Entries MAY contain, among others, the following operational attributes, defined in [5]. These attributes are maintained automatically by the server and are not modifiable by clients:

- creatorsName: the Distinguished Name of the user who added this entry to the directory.

- createTimestamp: the time this entry was added to the directory.

- modifiersName: the Distinguished Name of the user who last modified this entry.

- modifyTimestamp: the time this entry was last modified.

- subschemaSubentry: the Distinguished Name of the subschema entry (or subentry) which controls the schema for this entry.

#### 3.2.2. Subschema Entries and Subentries

Subschema entries are used for administering information about the directory schema, in particular the object classes and attribute types supported by directory servers. A single subschema entry contains all schema definitions used by entries in a particular part of the directory tree.

Servers which follow X.500(93) models SHOULD implement subschema using the X.500 subschema mechanisms, and so these subschemas are not ordinary entries. LDAP clients SHOULD NOT assume that servers implement any of the other aspects of X.500 subschema. A server which masters entries and permits clients to modify these entries MUST implement and provide access to these subschema entries, so that its clients may discover the attributes and object classes which are permitted to be present. It is strongly recommended that all other servers implement this as well.

The following four attributes MUST be present in all subschema entries:

- cn: this attribute MUST be used to form the RDN of the subschema entry.

- objectClass: the attribute MUST have at least the values "top" and "subschema".

- objectClasses: each value of this attribute specifies an object class known to the server.

- attributeTypes: each value of this attribute specifies an attribute type known to the server.

These are defined in [5]. Other attributes MAY be present in subschema entries, to reflect additional supported capabilities.







These include matchingRules, matchingRuleUse, dITStructureRules, dITContentRules, nameForms and ldapSyntaxes.

Servers SHOULD provide the attributes createTimestamp and modifyTimestamp in subschema entries, in order to allow clients to maintain their caches of schema information.

Clients MUST only retrieve attributes from a subschema entry by requesting a base object search of the entry, where the search filter is "(objectClass=subschema)". (This will allow LDAPv3 servers which gateway to X.500(93) to detect that subentry information is being requested.)

### 3.3. Relationship to X.500

This document defines LDAP in terms of X.500 as an X.500 access mechanism. An LDAP server MUST act in accordance with the X.500(1993) series of ITU recommendations when providing the service. However, it is not required that an LDAP server make use of any X.500 protocols in providing this service, e.g. LDAP can be mapped onto any other directory system so long as the X.500 data and service model as used in LDAP is not violated in the LDAP interface.

### 3.4. Server-specific Data Requirements

An LDAP server MUST provide information about itself and other information that is specific to each server. This is represented as a group of attributes located in the root DSE (DSA-Specific Entry), which is named with the zero-length LDAPDN. These attributes are retrievable if a client performs a base object search of the root with filter "(objectClass=*)", however they are subject to access control restrictions. The root DSE MUST NOT be included if the client performs a subtree search starting from the root.

Servers may allow clients to modify these attributes.

The following attributes of the root DSE are defined in section 5 of [5]. Additional attributes may be defined in other documents.

- namingContexts: naming contexts held in the server. Naming contexts are defined in section 17 of X.501 [6].

- subschemaSubentry: subschema entries (or subentries) known by this server.

- altServer: alternative servers in case this one is later unavailable.

- supportedExtension: list of supported extended operations.

- supportedControl: list of supported controls.

- supportedSASLMechanisms: list of supported SASL security features.

- supportedLDAPVersion: LDAP versions implemented by the server.

If the server does not master entries and does not know the locations of schema information, the subschemaSubentry attribute is not present in the root DSE. If the server masters directory entries under one or more schema rules, there may be any number of values of the subschemaSubentry attribute in the root DSE.

## 4. Elements of Protocol

The LDAP protocol is described using Abstract Syntax Notation 1 (ASN.1) [3], and is typically transferred using a subset of ASN.1 Basic Encoding Rules [11]. In order to support future extensions to this protocol, clients and servers MUST ignore elements of SEQUENCE encodings whose tags they do not recognize.

Note that unlike X.500, each change to the LDAP protocol other than through the extension mechanisms will have a different version number. A client will indicate the version it supports as part of the bind request, described in section 4.2. If a client has not sent a bind, the server MUST assume that version 3 is supported in the client (since version 2 required that the client bind first).

Clients may determine the protocol version a server supports by reading the supportedLDAPVersion attribute from the root DSE. Servers which implement version 3 or later versions MUST provide this attribute. Servers which only implement version 2 may not provide this attribute.

### 4.1. Common Elements

This section describes the LDAPMessage envelope PDU (Protocol Data Unit) format, as well as data type definitions which are used in the protocol operations.

#### 4.1.1. Message Envelope

For the purposes of protocol exchanges, all protocol operations are encapsulated in a common envelope, the LDAPMessage, which is defined as follows:

```
     LDAPMessage ::= SEQUENCE {
             messageID       MessageID,
             protocolOp      CHOICE {
                     bindRequest     BindRequest,
                     bindResponse    BindResponse,
                     unbindRequest   UnbindRequest,
                     searchRequest   SearchRequest,
                     searchResEntry  SearchResultEntry,
                     searchResDone   SearchResultDone,
                     searchResRef    SearchResultReference,
                     modifyRequest   ModifyRequest,
                     modifyResponse  ModifyResponse,
                     addRequest      AddRequest,
                     addResponse     AddResponse,
                     delRequest      DelRequest,
                     delResponse     DelResponse,
                     modDNRequest    ModifyDNRequest,
                     modDNResponse   ModifyDNResponse,
                     compareRequest  CompareRequest,
                     compareResponse CompareResponse,
                     abandonRequest  AbandonRequest,
                     extendedReq     ExtendedRequest,
                     extendedResp    ExtendedResponse },
              controls       [0] Controls OPTIONAL }

     MessageID ::= INTEGER (0 .. maxInt)

     maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --
```

The function of the LDAPMessage is to provide an envelope containing common fields required in all protocol exchanges. At this time the only common fields are the message ID and the controls.

If the server receives a PDU from the client in which the LDAPMessage SEQUENCE tag cannot be recognized, the messageID cannot be parsed, the tag of the protocolOp is not recognized as a request, or the encoding structures or lengths of data fields are found to be incorrect, then the server MUST return the notice of disconnection described in section 4.4.1, with resultCode protocolError, and immediately close the connection. In other cases that the server cannot parse the request received by the client, the server MUST return an appropriate response to the request, with the resultCode set to protocolError.

If the client receives a PDU from the server which cannot be parsed, the client may discard the PDU, or may abruptly close the connection.

The ASN.1 type Controls is defined in section 4.1.12.

#### 4.1.1.1. Message ID

All LDAPMessage envelopes encapsulating responses contain the messageID value of the corresponding request LDAPMessage.

The message ID of a request MUST have a value different from the values of any other requests outstanding in the LDAP session of which this message is a part.

A client MUST NOT send a second request with the same message ID as an earlier request on the same connection if the client has not received the final response from the earlier request. Otherwise the behavior is undefined. Typical clients increment a counter for each request.

A client MUST NOT reuse the message id of an abandonRequest or of the abandoned operation until it has received a response from the server for another request invoked subsequent to the abandonRequest, as the abandonRequest itself does not have a response.

### 4.1.2. String Types

The LDAPString is a notational convenience to indicate that, although strings of LDAPString type encode as OCTET STRING types, the ISO 10646 [13] character set (a superset of Unicode) is used, encoded following the UTF-8 algorithm [14]. Note that in the UTF-8 algorithm characters which are the same as ASCII (0x0000 through 0x007F) are represented as that same ASCII character in a single byte. The other byte values are used to form a variable-length encoding of an arbitrary character.

```
LDAPString ::= OCTET STRING
```

The LDAPOID is a notational convenience to indicate that the permitted value of this string is a (UTF-8 encoded) dotted-decimal representation of an OBJECT IDENTIFIER.

```
LDAPOID ::= OCTET STRING
```

For example,

```
1.3.6.1.4.1.1466.1.2.3
```

### 4.1.3. Distinguished Name and Relative Distinguished Name

An LDAPDN and a RelativeLDAPDN are respectively defined to be the representation of a Distinguished Name and a Relative Distinguished Name after encoding according to the specification in [4], such that





```
<distinguished-name> ::= <name>

<relative-distinguished-name> ::= <name-component>
```

where <name> and <name-component> are as defined in [4].

```
LDAPDN ::= LDAPString

RelativeLDAPDN ::= LDAPString
```

Only Attribute Types can be present in a relative distinguished name component; the options of Attribute Descriptions (next section) MUST NOT be used in specifying distinguished names.

### 4.1.4. Attribute Type

An AttributeType takes on as its value the textual string associated with that AttributeType in its specification.

```
AttributeType ::= LDAPString
```

Each attribute type has a unique OBJECT IDENTIFIER which has been assigned to it. This identifier may be written as decimal digits with components separated by periods, e.g. "2.5.4.10".

A specification may also assign one or more textual names for an attribute type. These names MUST begin with a letter, and only contain ASCII letters, digit characters and hyphens. They are case insensitive. (These ASCII characters are identical to ISO 10646 characters whose UTF-8 encoding is a single byte between 0x00 and 0x7F.)

If the server has a textual name for an attribute type, it MUST use a textual name for attributes returned in search results. The dotted-decimal OBJECT IDENTIFIER is only used if there is no textual name for an attribute type.

Attribute type textual names are non-unique, as two different specifications (neither in standards track RFCs) may choose the same name.

A server which masters or shadows entries SHOULD list all the attribute types it supports in the subschema entries, using the attributeTypes attribute. Servers which support an open-ended set of attributes SHOULD include at least the attributeTypes value for the 'objectClass' attribute. Clients MAY retrieve the attributeTypes value from subschema entries in order to obtain the OBJECT IDENTIFIER and other information associated with attribute types.

Some attribute type names which are used in this version of LDAP are described in [5]. Servers may implement additional attribute types.

### 4.1.5. Attribute Description

An AttributeDescription is a superset of the definition of the AttributeType. It has the same ASN.1 definition, but allows additional options to be specified. They are also case insensitive.

```
AttributeDescription ::= LDAPString
```

A value of AttributeDescription is based on the following BNF:

```
<AttributeDescription> ::= <AttributeType> [ ";" <options> ]

<options>  ::= <option> | <option> ";" <options>

<option>   ::= <opt-char> <opt-char>*

<opt-char> ::=  ASCII-equivalent letters, numbers and hyphen
```

Examples of valid AttributeDescription:

```
cn
userCertificate;binary
```

One option, "binary", is defined in this document. Additional options may be defined in IETF standards-track and experimental RFCs. Options beginning with "x-" are reserved for private experiments. Any option could be associated with any AttributeType, although not all combinations may be supported by a server.

An AttributeDescription with one or more options is treated as a subtype of the attribute type without any options. Options present in an AttributeDescription are never mutually exclusive. Implementations MUST generate the <options> list sorted in ascending order, and servers MUST treat any two AttributeDescription with the same AttributeType and options as equivalent. A server will treat an AttributeDescription with any options it does not implement as an unrecognized attribute type.

The data type "AttributeDescriptionList" describes a list of 0 or more attribute types. (A list of zero elements has special significance in the Search request.)

```
AttributeDescriptionList ::= SEQUENCE OF
        AttributeDescription
```







#### 4.1.5.1. Binary Option

If the "binary" option is present in an AttributeDescription, it overrides any string-based encoding representation defined for that attribute in [5]. Instead the attribute is to be transferred as a binary value encoded using the Basic Encoding Rules [11]. The syntax of the binary value is an ASN.1 data type definition which is referenced by the "SYNTAX" part of the attribute type definition.

The presence or absence of the "binary" option only affects the transfer of attribute values in protocol; servers store any particular attribute in a single format. If a client requests that a server return an attribute in the binary format, but the server cannot generate that format, the server MUST treat this attribute type as an unrecognized attribute type. Similarly, clients MUST NOT expect servers to return an attribute in binary format if the client requested that attribute by name without the binary option.

This option is intended to be used with attributes whose syntax is a complex ASN.1 data type, and the structure of values of that type is needed by clients. Examples of this kind of syntax are "Certificate" and "CertificateList".

### 4.1.6. Attribute Value

A field of type AttributeValue takes on as its value either a string encoding of a AttributeValue data type, or an OCTET STRING containing an encoded binary value, depending on whether the "binary" option is present in the companion AttributeDescription to this AttributeValue.

The definition of string encodings for different syntaxes and types may be found in other documents, and in particular [5].

```
AttributeValue ::= OCTET STRING
```

Note that there is no defined limit on the size of this encoding; thus protocol values may include multi-megabyte attributes (e.g. photographs).

Attributes may be defined which have arbitrary and non-printable syntax. Implementations MUST NEITHER simply display nor attempt to decode as ASN.1 a value if its syntax is not known. The implementation may attempt to discover the subschema of the source entry, and retrieve the values of attributeTypes from it.

Clients MUST NOT send attribute values in a request which are not valid according to the syntax defined for the attributes.

### 4.1.7. Attribute Value Assertion

The AttributeValueAssertion type definition is similar to the one in the X.500 directory standards. It contains an attribute description and a matching rule assertion value suitable for that type.

```
AttributeValueAssertion ::= SEQUENCE {
        attributeDesc   AttributeDescription,
        assertionValue  AssertionValue }

AssertionValue ::= OCTET STRING
```

If the "binary" option is present in attributeDesc, this signals to the server that the assertionValue is a binary encoding of the assertion value.

For all the string-valued user attributes described in [5], the assertion value syntax is the same as the value syntax. Clients may use attribute values as assertion values in compare requests and search filters.

Note however that the assertion syntax may be different from the value syntax for other attributes or for non-equality matching rules. These may have an assertion syntax which contains only part of the value. See section 20.2.1.8 of X.501 [6] for examples.

### 4.1.8. Attribute

An attribute consists of a type and one or more values of that type. (Though attributes MUST have at least one value when stored, due to access control restrictions the set may be empty when transferred in protocol. This is described in section 4.5.2, concerning the PartialAttributeList type.)

```
Attribute ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

Each attribute value is distinct in the set (no duplicates). The order of attribute values within the vals set is undefined and implementation-dependent, and MUST NOT be relied upon.

### 4.1.9. Matching Rule Identifier

A matching rule is a means of expressing how a server should compare an AssertionValue received in a search filter with an abstract data value. The matching rule defines the syntax of the assertion value and the process to be performed in the server.





An X.501(1993) Matching Rule is identified in the LDAP protocol by the printable representation of its OBJECT IDENTIFIER, either as one of the strings given in [5], or as decimal digits with components separated by periods, e.g. "caseIgnoreIA5Match" or "1.3.6.1.4.1.453.33.33".

```
MatchingRuleId ::= LDAPString
```

Servers which support matching rules for use in the extensibleMatch search filter MUST list the matching rules they implement in subschema entries, using the matchingRules attributes. The server SHOULD also list there, using the matchingRuleUse attribute, the attribute types with which each matching rule can be used. More information is given in section 4.4 of [5].

### 4.1.10. Result Message

The LDAPResult is the construct used in this protocol to return success or failure indications from servers to clients. In response to various requests servers will return responses containing fields of type LDAPResult to indicate the final status of a protocol operation request.

```
LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED {
                        success                      (0),
                        operationsError              (1),
                        protocolError                (2),
                        timeLimitExceeded            (3),
                        sizeLimitExceeded            (4),
                        compareFalse                 (5),
                        compareTrue                  (6),

                        authMethodNotSupported       (7),
                        strongAuthRequired           (8),
                                -- 9 reserved --
                        referral                     (10),  -- new
                        adminLimitExceeded           (11),  -- new
                        unavailableCriticalExtension (12),  -- new
                        confidentialityRequired      (13),  -- new
                        saslBindInProgress           (14),  -- new
                        noSuchAttribute              (16),
                        undefinedAttributeType       (17),
                        inappropriateMatching        (18),
                        constraintViolation          (19),
                        attributeOrValueExists       (20),
                        invalidAttributeSyntax       (21),
                                -- 22-31 unused --
```

```
                        noSuchObject                (32),
                        aliasProblem                (33),
                        invalidDNSyntax             (34),
                        -- 35 reserved for undefined isLeaf --
                        aliasDereferencingProblem   (36),
                                 -- 37-47 unused --
                        inappropriateAuthentication (48),
                        invalidCredentials          (49),
                        insufficientAccessRights    (50),
                        busy                        (51),
                        unavailable                 (52),
                        unwillingToPerform          (53),
                        loopDetect                  (54),
                                 -- 55-63 unused --
                        namingViolation             (64),
                        objectClassViolation        (65),
                        notAllowedOnNonLeaf         (66),
                        notAllowedOnRDN             (67),
                        entryAlreadyExists          (68),
                        objectClassModsProhibited   (69),
                                 -- 70 reserved for CLDAP --
                        affectsMultipleDSAs         (71), -- new
                                 -- 72-79 unused --
                        other                       (80) },
                        -- 81-90 reserved for APIs --
              matchedDN       LDAPDN,
              errorMessage    LDAPString,
              referral        [3] Referral OPTIONAL }
```

All the result codes with the exception of success, compareFalse and compareTrue are to be treated as meaning the operation could not be completed in its entirety.

Most of the result codes are based on problem indications from X.511 error data types. Result codes from 16 to 21 indicate an AttributeProblem, codes 32, 33, 34 and 36 indicate a NameProblem, codes 48, 49 and 50 indicate a SecurityProblem, codes 51 to 54 indicate a ServiceProblem, and codes 64 to 69 and 71 indicates an UpdateProblem.

If a client receives a result code which is not listed above, it is to be treated as an unknown error condition.

The errorMessage field of this construct may, at the server's option, be used to return a string containing a textual, human-readable (terminal control and page formatting characters should be avoided) error diagnostic. As this error diagnostic is not standardized,





implementations MUST NOT rely on the values returned. If the server chooses not to return a textual diagnostic, the errorMessage field of the LDAPResult type MUST contain a zero length string.

For result codes of noSuchObject, aliasProblem, invalidDNSyntax and aliasDereferencingProblem, the matchedDN field is set to the name of the lowest entry (object or alias) in the directory that was matched. If no aliases were dereferenced while attempting to locate the entry, this will be a truncated form of the name provided, or if aliases were dereferenced, of the resulting name, as defined in section 12.5 of X.511 [8]. The matchedDN field is to be set to a zero length string with all other result codes.

### 4.1.11. Referral

The referral error indicates that the contacted server does not hold the target entry of the request. The referral field is present in an LDAPResult if the LDAPResult.resultCode field value is referral, and absent with all other result codes. It contains a reference to another server (or set of servers) which may be accessed via LDAP or other protocols. Referrals can be returned in response to any operation request (except unbind and abandon which do not have responses). At least one URL MUST be present in the Referral.

The referral is not returned for a singleLevel or wholeSubtree search in which the search scope spans multiple naming contexts, and several different servers would need to be contacted to complete the operation. Instead, continuation references, described in section 4.5.3, are returned.

```
Referral ::= SEQUENCE OF LDAPURL  -- one or more

LDAPURL ::= LDAPString -- limited to characters permitted in URLs
```

If the client wishes to progress the operation, it MUST follow the referral by contacting any one of servers. All the URLs MUST be equally capable of being used to progress the operation. (The mechanisms for how this is achieved by multiple servers are outside the scope of this document.)

URLs for servers implementing the LDAP protocol are written according to [9]. If an alias was dereferenced, the <dn> part of the URL MUST be present, with the new target object name. If the <dn> part is present, the client MUST use this name in its next request to progress the operation, and if it is not present the client will use the same name as in the original request. Some servers (e.g. participating in distributed indexing) may provide a different filter in a referral for a search operation. If the filter part of the URL

is present in an LDAPURL, the client MUST use this filter in its next request to progress this search, and if it is not present the client MUST use the same filter as it used for that search. Other aspects of the new request may be the same or different as the request which generated the referral.

Note that UTF-8 characters appearing in a DN or search filter may not be legal for URLs (e.g. spaces) and MUST be escaped using the % method in RFC 1738 [7].

Other kinds of URLs may be returned, so long as the operation could be performed using that protocol.

4.1.12. Controls

A control is a way to specify extension information. Controls which are sent as part of a request apply only to that request and are not saved.

```
Controls ::= SEQUENCE OF Control

Control ::= SEQUENCE {
        controlType             LDAPOID,
        criticality             BOOLEAN DEFAULT FALSE,
        controlValue            OCTET STRING OPTIONAL }
```

The controlType field MUST be a UTF-8 encoded dotted-decimal representation of an OBJECT IDENTIFIER which uniquely identifies the control. This prevents conflicts between control names.

The criticality field is either TRUE or FALSE.

If the server recognizes the control type and it is appropriate for the operation, the server will make use of the control when performing the operation.

If the server does not recognize the control type and the criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unsupportedCriticalExtension.

If the control is not appropriate for the operation and criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unsupportedCriticalExtension.

If the control is unrecognized or inappropriate but the criticality field is FALSE, the server MUST ignore the control.







The controlValue contains any information associated with the control, and its format is defined for the control. The server MUST be prepared to handle arbitrary contents of the controlValue octet string, including zero bytes. It is absent only if there is no value information which is associated with a control of its type.

This document does not define any controls. Controls may be defined in other documents. The definition of a control consists of:

- the OBJECT IDENTIFIER assigned to the control,

- whether the control is always noncritical, always critical, or critical at the client's option,

- the format of the controlValue contents of the control.

Servers list the controls which they recognize in the supportedControl attribute in the root DSE.

4.2. Bind Operation

The function of the Bind Operation is to allow authentication information to be exchanged between the client and server.

The Bind Request is defined as follows:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                 -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }
```

Parameters of the Bind Request are:

- version: A version number indicating the version of the protocol to be used in this protocol session. This document describes version 3 of the LDAP protocol. Note that there is no version negotiation, and the client just sets this parameter to the version it desires. If the client requests protocol version 2, a server that supports the version 2 protocol as described in [2] will not return any v3-

specific protocol fields. (Note that not all LDAP servers will support protocol version 2, since they may be unable to generate the attribute syntaxes associated with version 2.)

- name: The name of the directory object that the client wishes to bind as. This field may take on a null value (a zero length string) for the purposes of anonymous binds, when authentication has been performed at a lower layer, or when using SASL credentials with a mechanism that includes the LDAPDN in the credentials.

- authentication: information used to authenticate the name, if any, provided in the Bind Request.

Upon receipt of a Bind Request, a protocol server will authenticate the requesting client, if necessary. The server will then return a Bind Response to the client indicating the status of the authentication.

Authorization is the use of this authentication information when performing operations. Authorization MAY be affected by factors outside of the LDAP Bind request, such as lower layer security services.

4.2.1. Sequencing of the Bind Request

For some SASL authentication mechanisms, it may be necessary for the client to invoke the BindRequest multiple times. If at any stage the client wishes to abort the bind process it MAY unbind and then drop the underlying connection. Clients MUST NOT invoke operations between two Bind requests made as part of a multi-stage bind.

A client may abort a SASL bind negotiation by sending a BindRequest with a different value in the mechanism field of SaslCredentials, or an AuthenticationChoice other than sasl.

If the client sends a BindRequest with the sasl mechanism field as an empty string, the server MUST return a BindResponse with authMethodNotSupported as the resultCode. This will allow clients to abort a negotiation if it wishes to try again with the same SASL mechanism.

Unlike LDAP v2, the client need not send a Bind Request in the first PDU of the connection. The client may request any operations and the server MUST treat these as unauthenticated. If the server requires that the client bind before browsing or modifying the directory, the server MAY reject a request other than binding, unbinding or an extended request with the "operationsError" result.







If the client did not bind before sending a request and receives an operationsError, it may then send a Bind Request. If this also fails or the client chooses not to bind on the existing connection, it will close the connection, reopen it and begin again by first sending a PDU with a Bind Request. This will aid in interoperating with servers implementing other versions of LDAP.

Clients MAY send multiple bind requests on a connection to change their credentials. A subsequent bind process has the effect of abandoning all operations outstanding on the connection. (This simplifies server implementation.) Authentication from earlier binds are subsequently ignored, and so if the bind fails, the connection will be treated as anonymous. If a SASL transfer encryption or integrity mechanism has been negotiated, and that mechanism does not support the changing of credentials from one identity to another, then the client MUST instead establish a new connection.

4.2.2. Authentication and Other Security Services

The simple authentication option provides minimal authentication facilities, with the contents of the authentication field consisting only of a cleartext password. Note that the use of cleartext passwords is not recommended over open networks when there is no authentication or encryption being performed by a lower layer; see the "Security Considerations" section.

If no authentication is to be performed, then the simple authentication option MUST be chosen, and the password be of zero length. (This is often done by LDAPv2 clients.) Typically the DN is also of zero length.

The sasl choice allows for any mechanism defined for use with SASL [12]. The mechanism field contains the name of the mechanism. The credentials field contains the arbitrary data used for authentication, inside an OCTET STRING wrapper. Note that unlike some Internet application protocols where SASL is used, LDAP is not text-based, thus no base64 transformations are performed on the credentials.

If any SASL-based integrity or confidentiality services are enabled, they take effect following the transmission by the server and reception by the client of the final BindResponse with resultCode success.

The client can request that the server use authentication information from a lower layer protocol by using the SASL EXTERNAL mechanism.

4.2.3. Bind Response

The Bind Response is defined as follows.

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
     COMPONENTS OF LDAPResult,
     serverSaslCreds    [7] OCTET STRING OPTIONAL }
```

BindResponse consists simply of an indication from the server of he status of the client's request for authentication.

f the bind was successful, the resultCode will be success, therwise it will be one of:

- operationsError: server encountered an internal error,
- protocolError: unrecognized version number or incorrect PDU structure,
- authMethodNotSupported: unrecognized SASL mechanism name,
- strongAuthRequired: the server requires authentication be performed with a SASL mechanism,
- referral: this server cannot accept this bind and the client should try another,
- saslBindInProgress: the server requires the client to send a new bind request, with the same sasl mechanism, to continue the authentication process,
- inappropriateAuthentication: the server requires the client which had attempted to bind anonymously or without supplying credentials to provide some form of credentials,
- invalidCredentials: the wrong password was supplied or the SASL credentials could not be processed,
- unavailable: the server is shutting down.

If the server does not support the client's requested protocol version, it MUST set the resultCode to protocolError.

If the client receives a BindResponse response where the resultCode was protocolError, it MUST close the connection as the server will be unwilling to accept further operations. (This is for compatibility with earlier versions of LDAP, in which the bind was always the first operation, and there was no negotiation.)







The serverSaslCreds are used as part of a SASL-defined bind mechanism to allow the client to authenticate the server to which it is communicating, or to perform "challenge-response" authentication. If the client bound with the password choice, or the SASL mechanism does not require the server to return information to the client, then this field is not to be included in the result.

4.3. Unbind Operation

The function of the Unbind Operation is to terminate a protocol session. The Unbind Operation is defined as follows:

```
UnbindRequest ::= [APPLICATION 2] NULL
```

The Unbind Operation has no response defined. Upon transmission of an UnbindRequest, a protocol client may assume that the protocol session is terminated. Upon receipt of an UnbindRequest, a protocol server may assume that the requesting client has terminated the session and that all outstanding requests may be discarded, and may close the connection.

4.4. Unsolicited Notification

An unsolicited notification is an LDAPMessage sent from the server to the client which is not in response to any LDAPMessage received by the server. It is used to signal an extraordinary condition in the server or in the connection between the client and the server. The notification is of an advisory nature, and the server will not expect any response to be returned from the client.

The unsolicited notification is structured as an LDAPMessage in which the messageID is 0 and protocolOp is of the extendedResp form. The responseName field of the ExtendedResponse is present. The LDAPOID value MUST be unique for this notification, and not be used in any other situation.

One unsolicited notification is defined in this document.

4.4.1. Notice of Disconnection

This notification may be used by the server to advise the client that the server is about to close the connection due to an error condition. Note that this notification is NOT a response to an unbind requested by the client: the server MUST follow the procedures of section 4.3. This notification is intended to assist clients in distinguishing between an error condition and a transient network

failure. As with a connection close due to network failure, the client MUST NOT assume that any outstanding requests which modified the directory have succeeded or failed.

The responseName is 1.3.6.1.4.1.1466.20036, the response field is absent, and the resultCode is used to indicate the reason for the disconnection.

The following resultCode values are to be used in this notification:

- protocolError: The server has received data from the client in which the LDAPMessage structure could not be parsed.

- strongAuthRequired: The server has detected that an established underlying security association protecting communication between the client and server has unexpectedly failed or been compromised.

- unavailable: This server will stop accepting new connections and operations on all existing connections, and be unavailable for an extended period of time. The client may make use of an alternative server.

After sending this notice, the server MUST close the connection. After receiving this notice, the client MUST NOT transmit any further on the connection, and may abruptly close the connection.

### 4.5. Search Operation

The Search Operation allows a client to request that a search be performed on its behalf by a server. This can be used to read attributes from a single entry, from entries immediately below a particular entry, or a whole subtree of entries.

#### 4.5.1. Search Request

The Search Request is defined as follows:

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED {
                neverDerefAliases       (0),
                derefInSearching        (1),
                derefFindingBaseObj     (2),
                derefAlways             (3) },
        sizeLimit       INTEGER (0 .. maxInt),
        timeLimit       INTEGER (0 .. maxInt),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList }

Filter ::= CHOICE {
        and             [0] SET OF Filter,
        or              [1] SET OF Filter,
        not             [2] Filter,
        equalityMatch   [3] AttributeValueAssertion,
        substrings      [4] SubstringFilter,
        greaterOrEqual  [5] AttributeValueAssertion,
        lessOrEqual     [6] AttributeValueAssertion,
        present         [7] AttributeDescription,
        approxMatch     [8] AttributeValueAssertion,
        extensibleMatch [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
        type            AttributeDescription,
        -- at least one must be present
        substrings      SEQUENCE OF CHOICE {
                initial [0] LDAPString,
                any     [1] LDAPString,
                final   [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRule    [1] MatchingRuleId OPTIONAL,
        type            [2] AttributeDescription OPTIONAL,
        matchValue      [3] AssertionValue,
        dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```

Parameters of the Search Request are:

- baseObject: An LDAPDN that is the base object entry relative to which the search is to be performed.

- scope: An indicator of the scope of the search to be performed. The semantics of the possible values of this field are identical to the semantics of the scope field in the X.511 Search Operation.

- derefAliases: An indicator as to how alias objects (as defined in X.501) are to be handled in searching. The semantics of the possible values of this field are:

  neverDerefAliases: do not dereference aliases in searching or in locating the base object of the search;

derefInSearching: dereference aliases in subordinates of the base object in searching, but not in locating the base object of the search;

derefFindingBaseObj: dereference aliases in locating the base object of the search, but not when searching subordinates of the base object;

derefAlways: dereference aliases both in searching and in locating the base object of the search.

- sizelimit: A sizelimit that restricts the maximum number of entries to be returned as a result of the search. A value of 0 in this field indicates that no client-requested sizelimit restrictions are in effect for the search. Servers may enforce a maximum number of entries to return.

- timelimit: A timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0 in this field indicates that no client-requested timelimit restrictions are in effect for the search.

- typesOnly: An indicator as to whether search results will contain both attribute types and values, or just attribute types. Setting this field to TRUE causes only attribute types (no values) to be returned. Setting this field to FALSE causes both attribute types and values to be returned.

- filter: A filter that defines the conditions that must be fulfilled in order for the search to match a given entry.

  The 'and', 'or' and 'not' choices can be used to form combinations of filters. At least one filter element MUST be present in an 'and' or 'or' choice. The others match against individual attribute values of entries in the scope of the search. (Implementor's note: the 'not' filter is an example of a tagged choice in an implicitly-tagged module. In BER this is treated as if the tag was explicit.)

  A server MUST evaluate filters according to the three-valued logic of X.511(93) section 7.8.1. In summary, a filter is evaluated to either "TRUE", "FALSE" or "Undefined". If the filter evaluates to TRUE for a particular entry, then the attributes of that entry are returned as part of the search result (subject to any applicable access control restrictions). If the filter evaluates to FALSE or Undefined, then the entry is ignored for the search.







A filter of the "and" choice is TRUE if all the filters in the SET OF evaluate to TRUE, FALSE if at least one filter is FALSE, and otherwise Undefined. A filter of the "or" choice is FALSE if all of the filters in the SET OF evaluate to FALSE, TRUE if at least one filter is TRUE, and Undefined otherwise. A filter of the "not" choice is TRUE if the filter being negated is FALSE, FALSE if it is TRUE, and Undefined if it is Undefined.

The present match evaluates to TRUE where there is an attribute or subtype of the specified attribute description present in an entry, and FALSE otherwise (including a presence test with an unrecognized attribute description.)

The extensibleMatch is new in this version of LDAP. If the matchingRule field is absent, the type field MUST be present, and the equality match is performed for that type. If the type field is absent and matchingRule is present, the matchValue is compared against all attributes in an entry which support that matchingRule, and the matchingRule determines the syntax for the assertion value (the filter item evaluates to TRUE if it matches with at least one attribute in the entry, FALSE if it does not match any attribute in the entry, and Undefined if the matchingRule is not recognized or the assertionValue cannot be parsed.) If the type field is present and matchingRule is present, the matchingRule MUST be one permitted for use with that type, otherwise the filter item is undefined. If the dnAttributes field is set to TRUE, the match is applied against all the attributes in an entry's distinguished name as well, and also evaluates to TRUE if there is at least one attribute in the distinguished name for which the filter item evaluates to TRUE. (Editors note: The dnAttributes field is present so that there does not need to be multiple versions of generic matching rules such as for word matching, one to apply to entries and another to apply to entries and dn attributes as well).

A filter item evaluates to Undefined when the server would not be able to determine whether the assertion value matches an entry. If an attribute description in an equalityMatch, substrings, greaterOrEqual, lessOrEqual, approxMatch or extensibleMatch filter is not recognized by the server, a matching rule id in the extensibleMatch is not recognized by the server, the assertion value cannot be parsed, or the type of filtering requested is not implemented, then the filter is Undefined. Thus for example if a server did not recognize the attribute type shoeSize, a filter of (shoeSize=*) would evaluate to FALSE, and the filters (shoeSize=12), (shoeSize>=12) and (shoeSize<=12) would evaluate to Undefined.

Servers MUST NOT return errors if attribute descriptions or matching rule ids are not recognized, or assertion values cannot be parsed. More details of filter processing are given in section 7.8 of X.511 [8].

- attributes: A list of the attributes to be returned from each entry which matches the search filter. There are two special values which may be used: an empty list with no attributes, and the attribute description string "*". Both of these signify that all user attributes are to be returned. (The "*" allows the client to request all user attributes in addition to specific operational attributes).

  Attributes MUST be named at most once in the list, and are returned at most once in an entry. If there are attribute descriptions in the list which are not recognized, they are ignored by the server.

  If the client does not want any attributes returned, it can specify a list containing only the attribute with OID "1.1". This OID was chosen arbitrarily and does not correspond to any attribute in use.

  Client implementors should note that even if all user attributes are requested, some attributes of the entry may not be included in search results due to access control or other restrictions. Furthermore, servers will not return operational attributes, such as objectClasses or attributeTypes, unless they are listed by name, since there may be extremely large number of values for certain operational attributes. (A list of operational attributes for use in LDAP is given in [5].)

Note that an X.500 "list"-like operation can be emulated by the client requesting a one-level LDAP search operation with a filter checking for the existence of the objectClass attribute, and that an X.500 "read"-like operation can be emulated by a base object LDAP search operation with the same filter. A server which provides a gateway to X.500 is not required to use the Read or List operations, although it may choose to do so, and if it does must provide the same semantics as the X.500 search operation.

### 4.5.2. Search Result

The results of the search attempted by the server upon receipt of a Search Request are returned in Search Responses, which are LDAP messages containing either SearchResultEntry, SearchResultReference, ExtendedResponse or SearchResultDone data types.

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
```





```
        attributes      PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
-- implementors should note that the PartialAttributeList may
-- have zero elements (if none of the attributes of that entry
-- were requested, or could be returned), and that the vals set
-- may also have zero elements (if types only was requested, or
-- all values were excluded from the result.)

SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL
-- at least one LDAPURL element must be present

SearchResultDone ::= [APPLICATION 5] LDAPResult
```

Upon receipt of a Search Request, a server will perform the necessary search of the DIT.

If the LDAP session is operating over a connection-oriented transport such as TCP, the server will return to the client a sequence of responses in separate LDAP messages. There may be zero or more responses containing SearchResultEntry, one for each entry found during the search. There may also be zero or more responses containing SearchResultReference, one for each area not explored by this server during the search. The SearchResultEntry and SearchResultReference PDUs may come in any order. Following all the SearchResultReference responses and all SearchResultEntry responses to be returned by the server, the server will return a response containing the SearchResultDone, which contains an indication of success, or detailing any errors that have occurred.

Each entry returned in a SearchResultEntry will contain all attributes, complete with associated values if necessary, as specified in the attributes field of the Search Request. Return of attributes is subject to access control and other administrative policy. Some attributes may be returned in binary format (indicated by the AttributeDescription in the response having the binary option present).

Some attributes may be constructed by the server and appear in a SearchResultEntry attribute list, although they are not stored attributes of an entry. Clients MUST NOT assume that all attributes can be modified, even if permitted by access control.

LDAPMessage responses of the ExtendedResponse form are reserved for returning information associated with a control requested by the client. These may be defined in future versions of this document.

### 4.5.3. Continuation References in the Search Result

If the server was able to locate the entry referred to by the baseObject but was unable to search all the entries in the scope at and under the baseObject, the server may return one or more SearchResultReference, each containing a reference to another set of servers for continuing the operation. A server MUST NOT return any SearchResultReference if it has not located the baseObject and thus has not searched any entries; in this case it would return a SearchResultDone containing a referral resultCode.

In the absence of indexing information provided to a server from servers holding subordinate naming contexts, SearchResultReference responses are not affected by search filters and are always returned when in scope.

The SearchResultReference is of the same data type as the Referral. URLs for servers implementing the LDAP protocol are written according to [9]. The <dn> part MUST be present in the URL, with the new target object name. The client MUST use this name in its next request. Some servers (e.g. part of a distributed index exchange system) may provide a different filter in the URLs of the SearchResultReference. If the filter part of the URL is present in an LDAP URL, the client MUST use the new filter in its next request to progress the search, and if the filter part is absent the client will use again the same filter. Other aspects of the new search request may be the same or different as the search which generated the continuation references.

Other kinds of URLs may be returned so long as the operation could be performed using that protocol.

The name of an unexplored subtree in a SearchResultReference need not be subordinate to the base object.

In order to complete the search, the client MUST issue a new search operation for each SearchResultReference that is returned. Note that the abandon operation described in section 4.11 applies only to a particular operation sent on a connection between a client and server, and if the client has multiple outstanding search operations to different servers, it MUST abandon each operation individually.

#### 4.5.3.1. Example

For example, suppose the contacted server (hosta) holds the entry "O=MNN,C=WW" and the entry "CN=Manager,O=MNN,C=WW". It knows that either LDAP-capable servers (hostb) or (hostc) hold "OU=People,O=MNN,C=WW" (one is the master and the other server a







shadow), and that LDAP-capable server (hostd) holds the subtree "OU=Roles,O=MNN,C=WW". If a subtree search of "O=MNN,C=WW" is requested to the contacted server, it may return the following:

```
SearchResultEntry for O=MNN,C=WW
SearchResultEntry for CN=Manager,O=MNN,C=WW
SearchResultReference {
  ldap://hostb/OU=People,O=MNN,C=WW
  ldap://hostc/OU=People,O=MNN,C=WW
}
SearchResultReference {
  ldap://hostd/OU=Roles,O=MNN,C=WW
}
SearchResultDone (success)
```

Client implementors should note that when following a SearchResultReference, additional SearchResultReference may be generated. Continuing the example, if the client contacted the server (hostb) and issued the search for the subtree "OU=People,O=MNN,C=WW", the server might respond as follows:

```
SearchResultEntry for OU=People,O=MNN,C=WW
SearchResultReference {
 ldap://hoste/OU=Managers,OU=People,O=MNN,C=WW
}
SearchResultReference {
 ldap://hostf/OU=Consultants,OU=People,O=MNN,C=WW
}
SearchResultDone (success)
```

If the contacted server does not hold the base object for the search, then it will return a referral to the client. For example, if the client requests a subtree search of "O=XYZ,C=US" to hosta, the server may return only a SearchResultDone containing a referral.

```
SearchResultDone (referral) {
  ldap://hostg/
}
```

## 4.6. Modify Operation

The Modify Operation allows a client to request that a modification of an entry be performed on its behalf by a server. The Modify Request is defined as follows:

```
     ModifyRequest ::= [APPLICATION 6] SEQUENCE {
             object          LDAPDN,
             modification    SEQUENCE OF SEQUENCE {
```

```
                    operation       ENUMERATED {
                                            add     (0),
                                            delete  (1),
                                            replace (2) },
                    modification    AttributeTypeAndValues } }

     AttributeTypeAndValues ::= SEQUENCE {
             type    AttributeDescription,
             vals    SET OF AttributeValue }
```

Parameters of the Modify Request are:

- object: The object to be modified. The value of this field contains the DN of the entry to be modified. The server will not perform any alias dereferencing in determining the object to be modified.

- modification: A list of modifications to be performed on the entry. The entire list of entry modifications MUST be performed in the order they are listed, as a single atomic operation. While individual modifications may violate the directory schema, the resulting entry after the entire list of modifications is performed MUST conform to the requirements of the directory schema. The values that may be taken on by the 'operation' field in each modification construct have the following semantics respectively:

  add: add values listed to the given attribute, creating the attribute if necessary;

  delete: delete values listed from the given attribute, removing the entire attribute if no values are listed, or if all current values of the attribute are listed for deletion;

  replace: replace all existing values of the given attribute with the new values listed, creating the attribute if it did not already exist. A replace with no value will delete the entire attribute if it exists, and is ignored if the attribute does not exist.

The result of the modify attempted by the server upon receipt of a Modify Request is returned in a Modify Response, defined as follows:

```
     ModifyResponse ::= [APPLICATION 7] LDAPResult
```

Upon receipt of a Modify Request, a server will perform the necessary modifications to the DIT.





The server will return to the client a single Modify Response indicating either the successful completion of the DIT modification, or the reason that the modification failed. Note that due to the requirement for atomicity in applying the list of modifications in the Modify Request, the client may expect that no modifications of the DIT have been performed if the Modify Response received indicates any sort of error, and that all requested modifications have been performed if the Modify Response indicates successful completion of the Modify Operation. If the connection fails, whether the modification occurred or not is indeterminate.

The Modify Operation cannot be used to remove from an entry any of its distinguished values, those values which form the entry's relative distinguished name. An attempt to do so will result in the server returning the error notAllowedOnRDN. The Modify DN Operation described in section 4.9 is used to rename an entry.

If an equality match filter has not been defined for an attribute type, clients MUST NOT attempt to delete individual values of that attribute from an entry using the "delete" form of a modification, and MUST instead use the "replace" form.

Note that due to the simplifications made in LDAP, there is not a direct mapping of the modifications in an LDAP ModifyRequest onto the EntryModifications of a DAP ModifyEntry operation, and different implementations of LDAP-DAP gateways may use different means of representing the change. If successful, the final effect of the operations on the entry MUST be identical.

## 4.7. Add Operation

The Add Operation allows a client to request the addition of an entry into the directory. The Add Request is defined as follows:

```
     AddRequest ::= [APPLICATION 8] SEQUENCE {
             entry           LDAPDN,
             attributes      AttributeList }

     AttributeList ::= SEQUENCE OF SEQUENCE {
             type    AttributeDescription,
             vals    SET OF AttributeValue }
```

Parameters of the Add Request are:

- entry: the Distinguished Name of the entry to be added. Note that the server will not dereference any aliases in locating the entry to be added.

- attributes: the list of attributes that make up the content of the entry being added. Clients MUST include distinguished values (those forming the entry's own RDN) in this list, the objectClass attribute, and values of any mandatory attributes of the listed object classes. Clients MUST NOT supply the createTimestamp or creatorsName attributes, since these will be generated automatically by the server.

The entry named in the entry field of the AddRequest MUST NOT exist for the AddRequest to succeed. The parent of the entry to be added MUST exist. For example, if the client attempted to add "CN=JS,O=Foo,C=US", the "O=Foo,C=US" entry did not exist, and the "C=US" entry did exist, then the server would return the error noSuchObject with the matchedDN field containing "C=US". If the parent entry exists but is not in a naming context held by the server, the server SHOULD return a referral to the server holding the parent entry.

Servers implementations SHOULD NOT restrict where entries can be located in the directory. Some servers MAY allow the administrator to restrict the classes of entries which can be added to the directory.

Upon receipt of an Add Request, a server will attempt to perform the add requested. The result of the add attempt will be returned to the client in the Add Response, defined as follows:

```
AddResponse ::= [APPLICATION 9] LDAPResult
```

A response of success indicates that the new entry is present in the directory.

## 4.8. Delete Operation

The Delete Operation allows a client to request the removal of an entry from the directory. The Delete Request is defined as follows:

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

The Delete Request consists of the Distinguished Name of the entry to be deleted. Note that the server will not dereference aliases while resolving the name of the target entry to be removed, and that only leaf entries (those with no subordinate entries) can be deleted with this operation.

The result of the delete attempted by the server upon receipt of a Delete Request is returned in the Delete Response, defined as follows:







```
DelResponse ::= [APPLICATION 11] LDAPResult
```

Upon receipt of a Delete Request, a server will attempt to perform the entry removal requested. The result of the delete attempt will be returned to the client in the Delete Response.

## 4.9. Modify DN Operation

The Modify DN Operation allows a client to change the leftmost (least significant) component of the name of an entry in the directory, or to move a subtree of entries to a new location in the directory. The Modify DN Request is defined as follows:

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }
```

Parameters of the Modify DN Request are:

- entry: the Distinguished Name of the entry to be changed. This entry may or may not have subordinate entries.

- newrdn: the RDN that will form the leftmost component of the new name of the entry.

- deleteoldrdn: a boolean parameter that controls whether the old RDN attribute values are to be retained as attributes of the entry, or deleted from the entry.

- newSuperior: if present, this is the Distinguished Name of the entry which becomes the immediate superior of the existing entry.

The result of the name change attempted by the server upon receipt of a Modify DN Request is returned in the Modify DN Response, defined as follows:

```
ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

Upon receipt of a ModifyDNRequest, a server will attempt to perform the name change. The result of the name change attempt will be returned to the client in the Modify DN Response.

For example, if the entry named in the "entry" parameter was "cn=John Smith,c=US", the newrdn parameter was "cn=John Cougar Smith", and the newSuperior parameter was absent, then this operation would

attempt to rename the entry to be "cn=John Cougar Smith,c=US". If there was already an entry with that name, the operation would fail with error code entryAlreadyExists.

If the deleteoldrdn parameter is TRUE, the values forming the old RDN are deleted from the entry. If the deleteoldrdn parameter is FALSE, the values forming the old RDN will be retained as non-distinguished attribute values of the entry. The server may not perform the operation and return an error code if the setting of the deleteoldrdn parameter would cause a schema inconsistency in the entry.

Note that X.500 restricts the ModifyDN operation to only affect entries that are contained within a single server. If the LDAP server is mapped onto DAP, then this restriction will apply, and the resultCode affectsMultipleDSAs will be returned if this error occurred. In general clients MUST NOT expect to be able to perform arbitrary movements of entries and subtrees between servers.

## 4.10. Compare Operation

The Compare Operation allows a client to compare an assertion provided with an entry in the directory. The Compare Request is defined as follows:

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }
```

Parameters of the Compare Request are:

- entry: the name of the entry to be compared with.

- ava: the assertion with which an attribute in the entry is to be compared.

The result of the compare attempted by the server upon receipt of a Compare Request is returned in the Compare Response, defined as follows:

```
CompareResponse ::= [APPLICATION 15] LDAPResult
```

Upon receipt of a Compare Request, a server will attempt to perform the requested comparison. The result of the comparison will be returned to the client in the Compare Response. Note that errors and the result of comparison are all returned in the same construct.





Note that some directory systems may establish access controls which permit the values of certain attributes (such as userPassword) to be compared but not read. In a search result, it may be that an attribute of that type would be returned, but with an empty set of values.

## 4.11. Abandon Operation

The function of the Abandon Operation is to allow a client to request that the server abandon an outstanding operation. The Abandon Request is defined as follows:

```
AbandonRequest ::= [APPLICATION 16] MessageID
```

The MessageID MUST be that of a an operation which was requested earlier in this connection.

(The abandon request itself has its own message id. This is distinct from the id of the earlier operation being abandoned.)

There is no response defined in the Abandon Operation. Upon transmission of an Abandon Operation, a client may expect that the operation identified by the Message ID in the Abandon Request has been abandoned. In the event that a server receives an Abandon Request on a Search Operation in the midst of transmitting responses to the search, that server MUST cease transmitting entry responses to the abandoned request immediately, and MUST NOT send the SearchResponseDone. Of course, the server MUST ensure that only properly encoded LDAPMessage PDUs are transmitted.

Clients MUST NOT send abandon requests for the same operation multiple times, and MUST also be prepared to receive results from operations it has abandoned (since these may have been in transit when the abandon was requested).

Servers MUST discard abandon requests for message IDs they do not recognize, for operations which cannot be abandoned, and for operations which have already been abandoned.

## 4.12. Extended Operation

An extension mechanism has been added in this version of LDAP, in order to allow additional operations to be defined for services not available elsewhere in this protocol, for instance digitally signed operations and results.

The extended operation allows clients to make requests and receive responses with predefined syntaxes and semantics. These may be defined in RFCs or be private to particular implementations. Each request MUST have a unique OBJECT IDENTIFIER assigned to it.

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

The requestName is a dotted-decimal representation of the OBJECT IDENTIFIER corresponding to the request. The requestValue is information in a form defined by that request, encapsulated inside an OCTET STRING.

The server will respond to this with an LDAPMessage containing the ExtendedResponse.

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

If the server does not recognize the request name, it MUST return only the response fields from LDAPResult, containing the protocolError result code.

## 5. Protocol Element Encodings and Transfer

One underlying service is defined here. Clients and servers SHOULD implement the mapping of LDAP over TCP described in 5.2.1.

### 5.1. Mapping Onto BER-based Transport Services

The protocol elements of LDAP are encoded for exchange using the Basic Encoding Rules (BER) [11] of ASN.1 [3]. However, due to the high overhead involved in using certain elements of the BER, the following additional restrictions are placed on BER-encodings of LDAP protocol elements:

(1) Only the definite form of length encoding will be used.

(2) OCTET STRING values will be encoded in the primitive form only.

(3) If the value of a BOOLEAN type is true, the encoding MUST have its contents octets set to hex "FF".





(4) If a value of a type is its default value, it MUST be absent. Only some BOOLEAN and INTEGER types have default values in this protocol definition.

These restrictions do not apply to ASN.1 types encapsulated inside of OCTET STRING values, such as attribute values, unless otherwise noted.

### 5.2. Transfer Protocols

This protocol is designed to run over connection-oriented, reliable transports, with all 8 bits in an octet being significant in the data stream.

#### 5.2.1. Transmission Control Protocol (TCP)

The LDAPMessage PDUs are mapped directly onto the TCP bytestream. It is recommended that server implementations running over the TCP MAY provide a protocol listener on the assigned port, 389. Servers may instead provide a listener on a different port number. Clients MUST support contacting servers on any valid TCP port.

## 6. Implementation Guidelines

This document describes an Internet protocol.

### 6.1. Server Implementations

The server MUST be capable of recognizing all the mandatory attribute type names and implement the syntaxes specified in [5]. Servers MAY also recognize additional attribute type names.

### 6.2. Client Implementations

Clients which request referrals MUST ensure that they do not loop between servers. They MUST NOT repeatedly contact the same server for the same request with the same target entry name, scope and filter. Some clients may be using a counter that is incremented each time referral handling occurs for an operation, and these kinds of clients MUST be able to handle a DIT with at least ten layers of naming contexts between the root and a leaf entry.

In the absence of prior agreements with servers, clients SHOULD NOT assume that servers support any particular schemas beyond those referenced in section 6.1. Different schemas can have different attribute types with the same names. The client can retrieve the subschema entries referenced by the subschemaSubentry attribute in the server's root DSE or in entries held by the server.

## 7. Security Considerations

When used with a connection-oriented transport, this version of the protocol provides facilities for the LDAP v2 authentication mechanism, simple authentication using a cleartext password, as well as any SASL mechanism [12]. SASL allows for integrity and privacy services to be negotiated.

It is also permitted that the server can return its credentials to the client, if it chooses to do so.

Use of cleartext password is strongly discouraged where the underlying transport service cannot guarantee confidentiality and may result in disclosure of the password to unauthorized parties.

When used with SASL, it should be noted that the name field of the BindRequest is not protected against modification. Thus if the distinguished name of the client (an LDAPDN) is agreed through the negotiation of the credentials, it takes precedence over any value in the unprotected name field.

Implementations which cache attributes and entries obtained via LDAP MUST ensure that access controls are maintained if that information is to be provided to multiple clients, since servers may have access control policies which prevent the return of entries or attributes in search results except to particular authenticated clients. For example, caches could serve result information only to the client whose request caused it to be cache.

## 8. Acknowledgements

This document is an update to RFC 1777, by Wengyik Yeong, Tim Howes, and Steve Kille. Design ideas included in this document are based on those discussed in ASID and other IETF Working Groups. The contributions of individuals in these working groups is gratefully acknowledged.

## 9. Bibliography

[1] ITU-T Rec. X.500, "The Directory: Overview of Concepts, Models and Service", 1993.

[2] Yeong, W., Howes, T., and S. Kille, "Lightweight Directory Access Protocol", RFC 1777, March 1995.

[3] ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", 1994.





[4] Kille, S., Wahl, M., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[5] Wahl, M., Coulbeck, A., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[6] ITU-T Rec. X.501, "The Directory: Models", 1993.

[7] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[8] ITU-T Rec. X.511, "The Directory: Abstract Service Definition", 1993.

[9] Howes, T., and M. Smith, "The LDAP URL Format", RFC 2255, December 1997.

[10] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[11] ITU-T Rec. X.690, "Specification of ASN.1 encoding rules: Basic, Canonical, and Distinguished Encoding Rules", 1994.

[12] Meyers, J., "Simple Authentication and Security Layer", RFC 2222, October 1997.

[13] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993.

[14] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

## 10. Authors' Addresses

Mark Wahl
Critical Angle Inc.
4815 W Braker Lane #502-385
Austin, TX 78759
USA

Phone: +1 512 372-3160
EMail: M.Wahl@critical-angle.com

Tim Howes
Netscape Communications Corp.
501 E. Middlefield Rd., MS MV068
Mountain View, CA 94043
USA

Phone: +1 650 937-3419
EMail: howes@netscape.com

Steve Kille
Isode Limited
The Dome, The Square
Richmond
TW9 1DT
UK

Phone: +44-181-332-9091
EMail: S.Kille@isode.com







## Appendix A - Complete ASN.1 Definition

```
Lightweight-Directory-Access-Protocol-V3 DEFINITIONS
IMPLICIT TAGS ::=

BEGIN

LDAPMessage ::= SEQUENCE {
        messageID       MessageID,
        protocolOp      CHOICE {
                bindRequest     BindRequest,
                bindResponse    BindResponse,
                unbindRequest   UnbindRequest,
                searchRequest   SearchRequest,
                searchResEntry  SearchResultEntry,
                searchResDone   SearchResultDone,
                searchResRef    SearchResultReference,
                modifyRequest   ModifyRequest,
                modifyResponse  ModifyResponse,
                addRequest      AddRequest,
                addResponse     AddResponse,
                delRequest      DelRequest,
                delResponse     DelResponse,
                modDNRequest    ModifyDNRequest,
                modDNResponse   ModifyDNResponse,
                compareRequest  CompareRequest,
                compareResponse CompareResponse,
                abandonRequest  AbandonRequest,
                extendedReq     ExtendedRequest,
                extendedResp    ExtendedResponse },
         controls       [0] Controls OPTIONAL }

MessageID ::= INTEGER (0 .. maxInt)

maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --

LDAPString ::= OCTET STRING

LDAPOID ::= OCTET STRING

LDAPDN ::= LDAPString

RelativeLDAPDN ::= LDAPString

AttributeType ::= LDAPString

AttributeDescription ::= LDAPString
```

```
AttributeDescriptionList ::= SEQUENCE OF
        AttributeDescription

AttributeValue ::= OCTET STRING

AttributeValueAssertion ::= SEQUENCE {
        attributeDesc   AttributeDescription,
        assertionValue  AssertionValue }

AssertionValue ::= OCTET STRING

Attribute ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

MatchingRuleId ::= LDAPString

LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED {
                        success                      (0),
                        operationsError              (1),
                        protocolError                (2),
                        timeLimitExceeded            (3),
                        sizeLimitExceeded            (4),
                        compareFalse                 (5),
                        compareTrue                  (6),
                        authMethodNotSupported       (7),
                        strongAuthRequired           (8),
                                -- 9 reserved --
                        referral                     (10),  -- new
                        adminLimitExceeded           (11),  -- new
                        unavailableCriticalExtension (12),  -- new
                        confidentialityRequired      (13),  -- new
                        saslBindInProgress           (14),  -- new
                        noSuchAttribute              (16),
                        undefinedAttributeType       (17),
                        inappropriateMatching        (18),
                        constraintViolation          (19),
                        attributeOrValueExists       (20),
                        invalidAttributeSyntax       (21),
                                -- 22-31 unused --
                        noSuchObject                 (32),
                        aliasProblem                 (33),
                        invalidDNSyntax              (34),
                        -- 35 reserved for undefined isLeaf --
                        aliasDereferencingProblem    (36),
                                -- 37-47 unused --
                        inappropriateAuthentication  (48),
```





```
                        invalidCredentials           (49),
                        insufficientAccessRights     (50),
                        busy                         (51),
                        unavailable                  (52),
                        unwillingToPerform           (53),
                        loopDetect                   (54),
                                -- 55-63 unused --
                        namingViolation              (64),
                        objectClassViolation         (65),
                        notAllowedOnNonLeaf          (66),
                        notAllowedOnRDN              (67),
                        entryAlreadyExists           (68),
                        objectClassModsProhibited    (69),
                                -- 70 reserved for CLDAP --
                        affectsMultipleDSAs          (71), -- new
                                -- 72-79 unused --
                        other                        (80) },
                        -- 81-90 reserved for APIs --
        matchedDN       LDAPDN,
        errorMessage    LDAPString,
        referral        [3] Referral OPTIONAL }

Referral ::= SEQUENCE OF LDAPURL

LDAPURL ::= LDAPString -- limited to characters permitted in URLs

Controls ::= SEQUENCE OF Control

Control ::= SEQUENCE {
        controlType             LDAPOID,
        criticality             BOOLEAN DEFAULT FALSE,
        controlValue            OCTET STRING OPTIONAL }

BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                 -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }

BindResponse ::= [APPLICATION 1] SEQUENCE {
```

```
        COMPONENTS OF LDAPResult,
        serverSaslCreds    [7] OCTET STRING OPTIONAL }

UnbindRequest ::= [APPLICATION 2] NULL

SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED {
                neverDerefAliases       (0),
                derefInSearching        (1),
                derefFindingBaseObj     (2),
                derefAlways             (3) },
        sizeLimit       INTEGER (0 .. maxInt),
        timeLimit       INTEGER (0 .. maxInt),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList }

Filter ::= CHOICE {
        and             [0] SET OF Filter,
        or              [1] SET OF Filter,
        not             [2] Filter,
        equalityMatch   [3] AttributeValueAssertion,
        substrings      [4] SubstringFilter,
        greaterOrEqual  [5] AttributeValueAssertion,
        lessOrEqual     [6] AttributeValueAssertion,
        present         [7] AttributeDescription,
        approxMatch     [8] AttributeValueAssertion,
        extensibleMatch [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
        type            AttributeDescription,
        -- at least one must be present
        substrings      SEQUENCE OF CHOICE {
                initial [0] LDAPString,
                any     [1] LDAPString,
                final   [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRule    [1] MatchingRuleId OPTIONAL,
        type            [2] AttributeDescription OPTIONAL,
        matchValue      [3] AssertionValue,
        dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```







```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
        attributes      PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL

SearchResultDone ::= [APPLICATION 5] LDAPResult

ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {
                operation       ENUMERATED {
                                        add     (0),
                                        delete  (1),
                                        replace (2) },
                modification    AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

ModifyResponse ::= [APPLICATION 7] LDAPResult

AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

AddResponse ::= [APPLICATION 9] LDAPResult

DelRequest ::= [APPLICATION 10] LDAPDN

DelResponse ::= [APPLICATION 11] LDAPResult

ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }

ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }

CompareResponse ::= [APPLICATION 15] LDAPResult

AbandonRequest ::= [APPLICATION 16] MessageID

ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }

ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }

END
```







Full Copyright Statement

Network Working Group
Request for Comments: 2252
Category: Standards Track

M. Wahl
Critical Angle Inc.
A. Coulbeck
Isode Inc.
T. Howes
Netscape Communications Corp.
S. Kille
Isode Limited
December 1997

# Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions

## 1. Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## Copyright Notice



## IESG Note

This document describes a directory access protocol that provides both read and update access. Update access requires secure authentication, but this document does not mandate implementation of any satisfactory authentication mechanisms.

In accordance with RFC 2026, section 4.4.1, this specification is being approved by IESG as a Proposed Standard despite this limitation, for the following reasons:

a. to encourage implementation and interoperability testing of these protocols (with or without update access) before they are deployed, and

b. to encourage deployment and use of these protocols in read-only applications. (e.g. applications where LDAPv3 is used as a query language for directories which are updated by some secure mechanism other than LDAP), and





c. to avoid delaying the advancement and deployment of other Internet standards-track protocols which require the ability to query, but not update, LDAPv3 directory servers.

Readers are hereby warned that until mandatory authentication mechanisms are standardized, clients and servers written according to this specification which make use of update functionality are UNLIKELY TO INTEROPERATE, or MAY INTEROPERATE ONLY IF AUTHENTICATION IS REDUCED TO AN UNACCEPTABLY WEAK LEVEL.

Implementors are hereby discouraged from deploying LDAPv3 clients or servers which implement the update functionality, until a Proposed Standard for mandatory authentication in LDAPv3 has been approved and published as an RFC.

## 2. Abstract

The Lightweight Directory Access Protocol (LDAP) [1] requires that the contents of AttributeValue fields in protocol elements be octet strings. This document defines a set of syntaxes for LDAPv3, and the rules by which attribute values of these syntaxes are represented as octet strings for transmission in the LDAP protocol. The syntaxes defined in this document are referenced by this and other documents that define attribute types. This document also defines the set of attribute types which LDAP servers should support.

## 3. Overview

This document defines the framework for developing schemas for directories accessible via the Lightweight Directory Access Protocol.

Schema is the collection of attribute type definitions, object class definitions and other information which a server uses to determine how to match a filter or attribute value assertion (in a compare operation) against the attributes of an entry, and whether to permit add and modify operations.

Section 4 states the general requirements and notations for attribute types, object classes, syntax and matching rule definitions.

Section 5 lists attributes, section 6 syntaxes and section 7 object classes.

Additional documents define schemas for representing real-world objects as directory entries.

## 4. General Issues

This document describes encodings used in an Internet protocol.

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 RFC 2119 [4].

Attribute Type and Object Class definitions are written in a string representation of the AttributeTypeDescription and ObjectClassDescription data types defined in X.501(93) [3]. Implementors are strongly advised to first read the description of how schema is represented in X.500 before reading the rest of this document.

### 4.1. Common Encoding Aspects

For the purposes of defining the encoding rules for attribute syntaxes, the following BNF definitions will be used. They are based on the BNF styles of RFC 822 [13].

```
a     = "a" / "b" / "c" / "d" / "e" / "f" / "g" / "h" / "i" /
        "j" / "k" / "l" / "m" / "n" / "o" / "p" / "q" / "r" /
        "s" / "t" / "u" / "v" / "w" / "x" / "y" / "z" / "A" /
        "B" / "C" / "D" / "E" / "F" / "G" / "H" / "I" / "J" /
        "K" / "L" / "M" / "N" / "O" / "P" / "Q" / "R" / "S" /
        "T" / "U" / "V" / "W" / "X" / "Y" / "Z"

d               = "0" / "1" / "2" / "3" / "4" /
                  "5" / "6" / "7" / "8" / "9"

hex-digit       =  d / "a" / "b" / "c" / "d" / "e" / "f" /
                        "A" / "B" / "C" / "D" / "E" / "F"

k               = a / d / "-" / ";"

p               = a / d / "'" / "(" / ")" / "+" / "," /
                  "-" / "." / "/" / ":" / "?" / " "

letterstring    = 1*a

numericstring   = 1*d

anhstring       = 1*k

keystring       = a [ anhstring ]

printablestring = 1*p
```





```
space           = 1*" "

whsp            = [ space ]

utf8            = <any sequence of octets formed from the UTF-8 [9]
                   transformation of a character from ISO10646 [10]>

dstring         = 1*utf8

qdstring        = whsp "'" dstring "'" whsp

qdstringlist    = [ qdstring *( qdstring ) ]

qdstrings       = qdstring / ( whsp "(" qdstringlist ")" whsp )
```

In the following BNF for the string representation of OBJECT IDENTIFIERs, descr is the syntactic representation of an object descriptor, which consists of letters and digits, starting with a letter. An OBJECT IDENTIFIER in the numericoid format should not have leading zeroes (e.g. "0.9.3" is permitted but "0.09.3" should not be generated).

When encoding 'oid' elements in a value, the descr encoding option SHOULD be used in preference to the numericoid. An object descriptor is a more readable alias for a number OBJECT IDENTIFIER, and these (where assigned and known by the implementation) SHOULD be used in preference to numeric oids to the greatest extent possible. Examples of object descriptors in LDAP are attribute type, object class and matching rule names.

```
oid             = descr / numericoid

descr           = keystring

numericoid      = numericstring *( "." numericstring )

woid            = whsp oid whsp

; set of oids of either form
oids            = woid / ( "(" oidlist ")" )

oidlist         = woid *( "$" woid )

; object descriptors used as schema element names
qdescrs         = qdescr / ( whsp "(" qdescrlist ")" whsp )

qdescrlist      = [ qdescr *( qdescr ) ]
```

```
qdescr          = whsp "'" descr "'" whsp
```

## 4.2. Attribute Types

The attribute types are described by sample values for the subschema "attributeTypes" attribute, which is written in the AttributeTypeDescription syntax. While lines have been folded for readability, the values transferred in protocol would not contain newlines.

The AttributeTypeDescription is encoded according to the following BNF, and the productions for oid, qdescrs and qdstring are given in section 4.1. Implementors should note that future versions of this document may have expanded this BNF to include additional terms. Terms which begin with the characters "X-" are reserved for private experiments, and MUST be followed by a <qdstrings>.

```
AttributeTypeDescription = "(" whsp
      numericoid whsp              ; AttributeType identifier
    [ "NAME" qdescrs ]             ; name used in AttributeType
    [ "DESC" qdstring ]            ; description
    [ "OBSOLETE" whsp ]
    [ "SUP" woid ]                 ; derived from this other
                                   ; AttributeType
    [ "EQUALITY" woid              ; Matching Rule name
    [ "ORDERING" woid              ; Matching Rule name
    [ "SUBSTR" woid ]              ; Matching Rule name
    [ "SYNTAX" whsp noidlen whsp ] ; see section 4.3
    [ "SINGLE-VALUE" whsp ]        ; default multi-valued
    [ "COLLECTIVE" whsp ]          ; default not collective
    [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
    [ "USAGE" whsp AttributeUsage ]; default userApplications
    whsp ")"

AttributeUsage =
    "userApplications"     /
    "directoryOperation"   /
    "distributedOperation" / ; DSA-shared
    "dSAOperation"          ; DSA-specific, value depends on server
```

Servers are not required to provide the same or any text in the description part of the subschema values they maintain. Servers SHOULD provide at least one of the "SUP" and "SYNTAX" fields for each AttributeTypeDescription.

Servers MUST implement all the attribute types referenced in sections 5.1, 5.2 and 5.3.





Servers MAY recognize additional names and attributes not listed in this document, and if they do so, MUST publish the definitions of the types in the attributeTypes attribute of their subschema entries.

Schema developers MUST NOT create attribute definitions whose names conflict with attributes defined for use with LDAP in existing standards-track RFCs.

An AttributeDescription can be used as the value in a NAME part of an AttributeTypeDescription. Note that these are case insensitive.

Note that the AttributeTypeDescription does not list the matching rules which can can be used with that attribute type in an extensibleMatch search filter. This is done using the matchingRuleUse attribute described in section 4.5.

This document refines the schema description of X.501 by requiring that the syntax field in an AttributeTypeDescription be a string representation of an OBJECT IDENTIFIER for the LDAP string syntax definition, and an optional indication of the maximum length of a value of this attribute (defined in section 4.3.2).

## 4.3. Syntaxes

This section defines general requirements for LDAP attribute value syntax encodings. All documents defining attribute syntax encodings for use with LDAP are expected to conform to these requirements.

The encoding rules defined for a given attribute syntax must produce octet strings. To the greatest extent possible, encoded octet strings should be usable in their native encoded form for display purposes. In particular, encoding rules for attribute syntaxes defining non-binary values should produce strings that can be displayed with little or no translation by clients implementing LDAP. There are a few cases (e.g. audio) however, when it is not sensible to produce a printable representation, and clients MUST NOT assume that an unrecognized syntax is a string representation.

In encodings where an arbitrary string, not a Distinguished Name, is used as part of a larger production, and other than as part of a Distinguished Name, a backslash quoting mechanism is used to escape the following separator symbol character (such as "'", "$" or "#") if it should occur in that string. The backslash is followed by a pair of hexadecimal digits representing the next character. A backslash itself in the string which forms part of a larger syntax is always transmitted as '\5C' or '\5c'. An example is given in section 6.27.

Syntaxes are also defined for matching rules whose assertion value syntax is different from the attribute value syntax.

### 4.3.1 Binary Transfer of Values

This encoding format is used if the binary encoding is requested by the client for an attribute, or if the attribute syntax name is "1.3.6.1.4.1.1466.115.121.1.5". The contents of the LDAP AttributeValue or AssertionValue field is a BER-encoded instance of the attribute value or a matching rule assertion value ASN.1 data type as defined for use with X.500. (The first byte inside the OCTET STRING wrapper is a tag octet. However, the OCTET STRING is still encoded in primitive form.)

All servers MUST implement this form for both generating attribute values in search responses, and parsing attribute values in add, compare and modify requests, if the attribute type is recognized and the attribute syntax name is that of Binary. Clients which request that all attributes be returned from entries MUST be prepared to receive values in binary (e.g. userCertificate;binary), and SHOULD NOT simply display binary or unrecognized values to users.

### 4.3.2. Syntax Object Identifiers

Syntaxes for use with LDAP are named by OBJECT IDENTIFIERs, which are dotted-decimal strings. These are not intended to be displayed to users.

```
noidlen = numericoid [ "{" len "}" ]

len     = numericstring
```

The following table lists some of the syntaxes that have been defined for LDAP thus far. The H-R column suggests whether a value in that syntax would likely be a human readable string. Clients and servers need not implement all the syntaxes listed here, and MAY implement other syntaxes.

Other documents may define additional syntaxes. However, the definition of additional arbitrary syntaxes is strongly deprecated since it will hinder interoperability: today's client and server implementations generally do not have the ability to dynamically recognize new syntaxes. In most cases attributes will be defined with the syntax for directory strings.





| Value being represented | H-R | OBJECT IDENTIFIER |
|---|---|---|
| ACI Item | N | 1.3.6.1.4.1.1466.115.121.1.1 |
| Access Point | Y | 1.3.6.1.4.1.1466.115.121.1.2 |
| Attribute Type Description | Y | 1.3.6.1.4.1.1466.115.121.1.3 |
| Audio | N | 1.3.6.1.4.1.1466.115.121.1.4 |
| Binary | N | 1.3.6.1.4.1.1466.115.121.1.5 |
| Bit String | Y | 1.3.6.1.4.1.1466.115.121.1.6 |
| Boolean | Y | 1.3.6.1.4.1.1466.115.121.1.7 |
| Certificate | N | 1.3.6.1.4.1.1466.115.121.1.8 |
| Certificate List | N | 1.3.6.1.4.1.1466.115.121.1.9 |
| Certificate Pair | N | 1.3.6.1.4.1.1466.115.121.1.10 |
| Country String | Y | 1.3.6.1.4.1.1466.115.121.1.11 |
| DN | Y | 1.3.6.1.4.1.1466.115.121.1.12 |
| Data Quality Syntax | Y | 1.3.6.1.4.1.1466.115.121.1.13 |
| Delivery Method | Y | 1.3.6.1.4.1.1466.115.121.1.14 |
| Directory String | Y | 1.3.6.1.4.1.1466.115.121.1.15 |
| DIT Content Rule Description | Y | 1.3.6.1.4.1.1466.115.121.1.16 |
| DIT Structure Rule Description | Y | 1.3.6.1.4.1.1466.115.121.1.17 |
| DL Submit Permission | Y | 1.3.6.1.4.1.1466.115.121.1.18 |
| DSA Quality Syntax | Y | 1.3.6.1.4.1.1466.115.121.1.19 |
| DSE Type | Y | 1.3.6.1.4.1.1466.115.121.1.20 |
| Enhanced Guide | Y | 1.3.6.1.4.1.1466.115.121.1.21 |
| Facsimile Telephone Number | Y | 1.3.6.1.4.1.1466.115.121.1.22 |
| Fax | N | 1.3.6.1.4.1.1466.115.121.1.23 |
| Generalized Time | Y | 1.3.6.1.4.1.1466.115.121.1.24 |
| Guide | Y | 1.3.6.1.4.1.1466.115.121.1.25 |
| IA5 String | Y | 1.3.6.1.4.1.1466.115.121.1.26 |
| INTEGER | Y | 1.3.6.1.4.1.1466.115.121.1.27 |
| JPEG | N | 1.3.6.1.4.1.1466.115.121.1.28 |
| LDAP Syntax Description | Y | 1.3.6.1.4.1.1466.115.121.1.54 |
| LDAP Schema Definition | Y | 1.3.6.1.4.1.1466.115.121.1.56 |
| LDAP Schema Description | Y | 1.3.6.1.4.1.1466.115.121.1.57 |
| Master And Shadow Access Points | Y | 1.3.6.1.4.1.1466.115.121.1.29 |
| Matching Rule Description | Y | 1.3.6.1.4.1.1466.115.121.1.30 |
| Matching Rule Use Description | Y | 1.3.6.1.4.1.1466.115.121.1.31 |
| Mail Preference | Y | 1.3.6.1.4.1.1466.115.121.1.32 |
| MHS OR Address | Y | 1.3.6.1.4.1.1466.115.121.1.33 |
| Modify Rights | Y | 1.3.6.1.4.1.1466.115.121.1.55 |
| Name And Optional UID | Y | 1.3.6.1.4.1.1466.115.121.1.34 |
| Name Form Description | Y | 1.3.6.1.4.1.1466.115.121.1.35 |
| Numeric String | Y | 1.3.6.1.4.1.1466.115.121.1.36 |
| Object Class Description | Y | 1.3.6.1.4.1.1466.115.121.1.37 |
| Octet String | Y | 1.3.6.1.4.1.1466.115.121.1.40 |
| OID | Y | 1.3.6.1.4.1.1466.115.121.1.38 |
| Other Mailbox | Y | 1.3.6.1.4.1.1466.115.121.1.39 |
| Postal Address | Y | 1.3.6.1.4.1.1466.115.121.1.41 |
| Protocol Information | Y | 1.3.6.1.4.1.1466.115.121.1.42 |

| | | |
|---|---|---|
| Presentation Address | Y | 1.3.6.1.4.1.1466.115.121.1.43 |
| Printable String | Y | 1.3.6.1.4.1.1466.115.121.1.44 |
| Substring Assertion | Y | 1.3.6.1.4.1.1466.115.121.1.58 |
| Subtree Specification | Y | 1.3.6.1.4.1.1466.115.121.1.45 |
| Supplier Information | Y | 1.3.6.1.4.1.1466.115.121.1.46 |
| Supplier Or Consumer | Y | 1.3.6.1.4.1.1466.115.121.1.47 |
| Supplier And Consumer | Y | 1.3.6.1.4.1.1466.115.121.1.48 |
| Supported Algorithm | N | 1.3.6.1.4.1.1466.115.121.1.49 |
| Telephone Number | Y | 1.3.6.1.4.1.1466.115.121.1.50 |
| Teletex Terminal Identifier | Y | 1.3.6.1.4.1.1466.115.121.1.51 |
| Telex Number | Y | 1.3.6.1.4.1.1466.115.121.1.52 |
| UTC Time | Y | 1.3.6.1.4.1.1466.115.121.1.53 |

A suggested minimum upper bound on the number of characters in value with a string-based syntax, or the number of bytes in a value for all other syntaxes, may be indicated by appending this bound count inside of curly braces following the syntax name's OBJECT IDENTIFIER in an Attribute Type Description. This bound is not part of the syntax name itself. For instance, "1.3.6.4.1.1466.0{64}" suggests that server implementations should allow a string to be 64 characters long, although they may allow longer strings. Note that a single character of the Directory String syntax may be encoded in more than one byte since UTF-8 is a variable-length encoding.

#### 4.3.3. Syntax Description

The following BNF may be used to associate a short description with a syntax OBJECT IDENTIFIER. Implementors should note that future versions of this document may expand this definition to include additional terms. Terms whose identifier begins with "X-" are reserved for private experiments, and MUST be followed by a <qdstrings>.

```
SyntaxDescription = "(" whsp
    numericoid whsp
    [ "DESC" qdstring ]
    whsp ")"
```

### 4.4. Object Classes

The format for representation of object classes is defined in X.501 [3]. In general every entry will contain an abstract class ("top" or "alias"), at least one structural object class, and zero or more auxiliary object classes. Whether an object class is abstract, structural or auxiliary is defined when the object class identifier is assigned. An object class definition should not be changed without having a new identifier assigned to it.





Object class descriptions are written according to the following BNF. Implementors should note that future versions of this document may expand this definition to include additional terms. Terms whose identifier begins with "X-" are reserved for private experiments, and MUST be followed by a <qdstrings> encoding.

```
ObjectClassDescription = "(" whsp
    numericoid whsp      ; ObjectClass identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" whsp ]
    [ "SUP" oids ]       ; Superior ObjectClasses
    [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                         ; default structural
    [ "MUST" oids ]      ; AttributeTypes
    [ "MAY" oids ]       ; AttributeTypes
whsp ")"
```

These are described as sample values for the subschema "objectClasses" attribute for a server which implements the LDAP schema. While lines have been folded for readability, the values transferred in protocol would not contain newlines.

Servers SHOULD implement all the object classes referenced in section 7, except for extensibleObject, which is optional. Servers MAY implement additional object classes not listed in this document, and if they do so, MUST publish the definitions of the classes in the objectClasses attribute of their subschema entries.

Schema developers MUST NOT create object class definitions whose names conflict with attributes defined for use with LDAP in existing standards-track RFCs.

### 4.5. Matching Rules

Matching rules are used by servers to compare attribute values against assertion values when performing Search and Compare operations. They are also used to identify the value to be added or deleted when modifying entries, and are used when comparing a purported distinguished name with the name of an entry.

Most of the attributes given in this document will have an equality matching rule defined.

Matching rule descriptions are written according to the following BNF. Implementors should note that future versions of this document may have expanded this BNF to include additional terms. Terms whose identifier begins with "X-" are reserved for private experiments, and

MUST be followed by a <qdstrings> encoding.

```
MatchingRuleDescription = "(" whsp
    numericoid whsp  ; MatchingRule identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" whsp ]
    "SYNTAX" numericoid
whsp ")"
```

Values of the matchingRuleUse list the attributes which are suitable for use with an extensible matching rule.

```
MatchingRuleUseDescription = "(" whsp
    numericoid whsp  ; MatchingRule identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" ]
   "APPLIES" oids    ; AttributeType identifiers
whsp ")"
```

Servers which support matching rules and the extensibleMatch SHOULD implement all the matching rules in section 8.

Servers MAY implement additional matching rules not listed in this document, and if they do so, MUST publish the definitions of the matching rules in the matchingRules attribute of their subschema entries. If the server supports the extensibleMatch, then the server MUST publish the relationship between the matching rules and attributes in the matchingRuleUse attribute.

For example, a server which implements a privately-defined matching rule for performing sound-alike matches on Directory String-valued attributes would include the following in the subschema entry (1.2.3.4.5 is an example, the OID of an actual matching rule would be different):

```
matchingRule: ( 1.2.3.4.5 NAME 'soundAlikeMatch'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

If this matching rule could be used with the attributes 2.5.4.41 and 2.5.4.15, the following would also be present:

```
matchingRuleUse: ( 1.2.3.4.5 APPLIES (2.5.4.41 $ 2.5.4.15) )
```







A client could then make use of this matching rule by sending a search operation in which the filter is of the extensibleMatch choice, the matchingRule field is "soundAlikeMatch", and the type field is "2.5.4.41" or "2.5.4.15".

## 5. Attribute Types

All LDAP server implementations MUST recognize the attribute types defined in this section.

Servers SHOULD also recognize all the attributes from section 5 of [12].

### 5.1. Standard Operational Attributes

Servers MUST maintain values of these attributes in accordance with the definitions in X.501(93).

#### 5.1.1. createTimestamp

This attribute SHOULD appear in entries which were created using the Add operation.

```
( 2.5.18.1 NAME 'createTimestamp' EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  SINGLE-VALUE NO-USER-MODIFICATION USAGE directoryOperation )
```

#### 5.1.2. modifyTimestamp

This attribute SHOULD appear in entries which have been modified using the Modify operation.

```
( 2.5.18.2 NAME 'modifyTimestamp' EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  SINGLE-VALUE NO-USER-MODIFICATION USAGE directoryOperation )
```

#### 5.1.3. creatorsName

This attribute SHOULD appear in entries which were created using the Add operation.

```
( 2.5.18.3 NAME 'creatorsName' EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  SINGLE-VALUE NO-USER-MODIFICATION USAGE directoryOperation )
```

5.1.4. modifiersName

This attribute SHOULD appear in entries which have been modified using the Modify operation.

```
( 2.5.18.4 NAME 'modifiersName' EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  SINGLE-VALUE NO-USER-MODIFICATION USAGE directoryOperation )
```

5.1.5. subschemaSubentry

The value of this attribute is the name of a subschema entry (or subentry if the server is based on X.500(93)) in which the server makes available attributes specifying the schema.

```
( 2.5.18.10 NAME 'subschemaSubentry'
  EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 NO-USER-MODIFICATION
  SINGLE-VALUE USAGE directoryOperation )
```

5.1.6. attributeTypes

This attribute is typically located in the subschema entry.

```
( 2.5.21.5 NAME 'attributeTypes'
  EQUALITY objectIdentifierFirstComponentMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.3 USAGE directoryOperation )
```

5.1.7. objectClasses

This attribute is typically located in the subschema entry.

```
( 2.5.21.6 NAME 'objectClasses'
  EQUALITY objectIdentifierFirstComponentMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.37 USAGE directoryOperation )
```

5.1.8. matchingRules

This attribute is typically located in the subschema entry.

```
( 2.5.21.4 NAME 'matchingRules'
  EQUALITY objectIdentifierFirstComponentMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.30 USAGE directoryOperation )
```





5.1.9. matchingRuleUse

This attribute is typically located in the subschema entry.

```
( 2.5.21.8 NAME 'matchingRuleUse'
  EQUALITY objectIdentifierFirstComponentMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.31 USAGE directoryOperation )
```

5.2. LDAP Operational Attributes

These attributes are only present in the root DSE (see [1] and [3]).

Servers MUST recognize these attribute names, but it is not required that a server provide values for these attributes, when the attribute corresponds to a feature which the server does not implement.

5.2.1. namingContexts

The values of this attribute correspond to naming contexts which this server masters or shadows. If the server does not master any information (e.g. it is an LDAP gateway to a public X.500 directory) this attribute will be absent. If the server believes it contains the entire directory, the attribute will have a single value, and that value will be the empty string (indicating the null DN of the root). This attribute will allow a client to choose suitable base objects for searching when it has contacted a server.

```
( 1.3.6.1.4.1.1466.101.120.5 NAME 'namingContexts'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 USAGE dSAOperation )
```

5.2.2. altServer

The values of this attribute are URLs of other servers which may be contacted when this server becomes unavailable. If the server does not know of any other servers which could be used this attribute will be absent. Clients may cache this information in case their preferred LDAP server later becomes unavailable.

```
( 1.3.6.1.4.1.1466.101.120.6 NAME 'altServer'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 USAGE dSAOperation )
```

5.2.3. supportedExtension

The values of this attribute are OBJECT IDENTIFIERs identifying the supported extended operations which the server supports.

If the server does not support any extensions this attribute will be absent.

```
    ( 1.3.6.1.4.1.1466.101.120.7 NAME 'supportedExtension'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 USAGE dSAOperation )
```

5.2.4. supportedControl

The values of this attribute are the OBJECT IDENTIFIERs identifying controls which the server supports. If the server does not support any controls, this attribute will be absent.

```
    ( 1.3.6.1.4.1.1466.101.120.13 NAME 'supportedControl'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 USAGE dSAOperation )
```

5.2.5. supportedSASLMechanisms

The values of this attribute are the names of supported SASL mechanisms which the server supports. If the server does not support any mechanisms this attribute will be absent.

```
    ( 1.3.6.1.4.1.1466.101.120.14 NAME 'supportedSASLMechanisms'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 USAGE dSAOperation )
```

5.2.6. supportedLDAPVersion

The values of this attribute are the versions of the LDAP protocol which the server implements.

```
    ( 1.3.6.1.4.1.1466.101.120.15 NAME 'supportedLDAPVersion'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 USAGE dSAOperation )
```

5.3. LDAP Subschema Attribute

This attribute is typically located in the subschema entry.

5.3.1. ldapSyntaxes

Servers MAY use this attribute to list the syntaxes which are implemented. Each value corresponds to one syntax.

```
    ( 1.3.6.1.4.1.1466.101.120.16 NAME 'ldapSyntaxes'
      EQUALITY objectIdentifierFirstComponentMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.54 USAGE directoryOperation )
```

5.4. X.500 Subschema attributes

These attributes are located in the subschema entry. All servers SHOULD recognize their name, although typically only X.500 servers will implement their functionality.





5.4.1. dITStructureRules

```
 ( 2.5.21.1 NAME 'dITStructureRules' EQUALITY integerFirstComponentMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.17 USAGE directoryOperation )
```

5.4.2. nameForms

```
    ( 2.5.21.7 NAME 'nameForms'
      EQUALITY objectIdentifierFirstComponentMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.35 USAGE directoryOperation )
```

5.4.3. ditContentRules

```
    ( 2.5.21.2 NAME 'dITContentRules'
      EQUALITY objectIdentifierFirstComponentMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.16 USAGE directoryOperation )
```

6. Syntaxes

Servers SHOULD recognize all the syntaxes described in this section.

6.1. Attribute Type Description

```
   ( 1.3.6.1.4.1.1466.115.121.1.3 DESC 'Attribute Type Description' )
```

Values in this syntax are encoded according to the BNF given at the start of section 4.2. For example,

```
        ( 2.5.4.0 NAME 'objectClass'
          SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

6.2. Binary

```
   ( 1.3.6.1.4.1.1466.115.121.1.5 DESC 'Binary' )
```

Values in this syntax are encoded as described in section 4.3.1.

6.3. Bit String

```
   ( 1.3.6.1.4.1.1466.115.121.1.6 DESC 'Bit String' )
```

Values in this syntax are encoded according to the following BNF:

```
   bitstring = "'" *binary-digit "'B"

   binary-digit = "0" / "1"
```

Example:

'0101111101'B

6.4. Boolean

( 1.3.6.1.4.1.1466.115.121.1.7 DESC 'Boolean' )

Values in this syntax are encoded according to the following BNF:

boolean = "TRUE" / "FALSE"

Boolean values have an encoding of "TRUE" if they are logically true, and have an encoding of "FALSE" otherwise.

6.5. Certificate

( 1.3.6.1.4.1.1466.115.121.1.8 DESC 'Certificate' )

Because of the changes from X.509(1988) and X.509(1993) and additional changes to the ASN.1 definition to support certificate extensions, no string representation is defined, and values in this syntax MUST only be transferred using the binary encoding, by requesting or returning the attributes with descriptions "userCertificate;binary" or "caCertificate;binary". The BNF notation in RFC 1778 for "User Certificate" is not recommended to be used.

6.6. Certificate List

( 1.3.6.1.4.1.1466.115.121.1.9 DESC 'Certificate List' )

Because of the incompatibility of the X.509(1988) and X.509(1993) definitions of revocation lists, values in this syntax MUST only be transferred using a binary encoding, by requesting or returning the attributes with descriptions "certificateRevocationList;binary" or "authorityRevocationList;binary". The BNF notation in RFC 1778 for "Authority Revocation List" is not recommended to be used.

6.7. Certificate Pair

( 1.3.6.1.4.1.1466.115.121.1.10 DESC 'Certificate Pair' )

Because the Certificate is being carried in binary, values in this syntax MUST only be transferred using a binary encoding, by requesting or returning the attribute description "crossCertificatePair;binary". The BNF notation in RFC 1778 for "Certificate Pair" is not recommended to be used.





6.8. Country String

( 1.3.6.1.4.1.1466.115.121.1.11 DESC 'Country String' )

A value in this syntax is encoded the same as a value of Directory String syntax. Note that this syntax is limited to values of exactly two printable string characters, as listed in ISO 3166 [14].

CountryString = p p

Example:
US

6.9. DN

( 1.3.6.1.4.1.1466.115.121.1.12 DESC 'DN' )

Values in the Distinguished Name syntax are encoded to have the representation defined in [5]. Note that this representation is not reversible to an ASN.1 encoding used in X.500 for Distinguished Names, as the CHOICE of any DirectoryString element in an RDN is no longer known.

Examples (from [5]):
CN=Steve Kille,O=Isode Limited,C=GB
OU=Sales+CN=J. Smith,O=Widget Inc.,C=US
CN=L. Eagle,O=Sue\, Grabbit and Runn,C=GB
CN=Before\0DAfter,O=Test,C=GB
1.3.6.1.4.1.1466.0=#04024869,O=Test,C=GB
SN=Lu\C4\8Di\C4\87

6.10. Directory String

( 1.3.6.1.4.1.1466.115.121.1.15 DESC 'Directory String' )

A string in this syntax is encoded in the UTF-8 form of ISO 10646 (a superset of Unicode). Servers and clients MUST be prepared to receive encodings of arbitrary Unicode characters, including characters not presently assigned to any character set.

For characters in the PrintableString form, the value is encoded as the string value itself.

If it is of the TeletexString form, then the characters are transliterated to their equivalents in UniversalString, and encoded in UTF-8 [9].

If it is of the UniversalString or BMPString forms [10], UTF-8 is used to encode them.

Note: the form of DirectoryString is not indicated in protocol unless the attribute value is carried in binary. Servers which convert to DAP MUST choose an appropriate form. Servers MUST NOT reject values merely because they contain legal Unicode characters outside of the range of printable ASCII.

Example:

This is a string of DirectoryString containing #!%#@

## 6.11. DIT Content Rule Description

( 1.3.6.1.4.1.1466.115.121.1.16 DESC 'DIT Content Rule Description' )

Values in this syntax are encoded according to the following BNF. Implementors should note that future versions of this document may have expanded this BNF to include additional terms.

```
DITContentRuleDescription = "("
    numericoid   ; Structural ObjectClass identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" ]
    [ "AUX" oids ]    ; Auxiliary ObjectClasses
    [ "MUST" oids ]   ; AttributeType identifiers
    [ "MAY" oids ]    ; AttributeType identifiers
    [ "NOT" oids ]    ; AttributeType identifiers
   ")"
```

## 6.12. Facsimile Telephone Number

( 1.3.6.1.4.1.1466.115.121.1.22 DESC 'Facsimile Telephone Number' )

Values in this syntax are encoded according to the following BNF:

```
fax-number    = printablestring [ "$" faxparameters ]

faxparameters = faxparm / ( faxparm "$" faxparameters )

faxparm = "twoDimensional" / "fineResolution" /
          "unlimitedLength" /
          "b4Length" / "a3Width" / "b4Width" / "uncompressed"
```





In the above, the first printablestring is the telephone number, based on E.123 [15], and the faxparm tokens represent fax parameters.

## 6.13. Fax

( 1.3.6.1.4.1.1466.115.121.1.23 DESC 'Fax' )

Values in this syntax are encoded as if they were octet strings containing Group 3 Fax images as defined in [7].

## 6.14. Generalized Time

( 1.3.6.1.4.1.1466.115.121.1.24 DESC 'Generalized Time' )

Values in this syntax are encoded as printable strings, represented as specified in X.208. Note that the time zone must be specified. It is strongly recommended that GMT time be used. For example,

199412161032Z

## 6.15. IA5 String

( 1.3.6.1.4.1.1466.115.121.1.26 DESC 'IA5 String' )

The encoding of a value in this syntax is the string value itself.

## 6.16. INTEGER

( 1.3.6.1.4.1.1466.115.121.1.27 DESC 'INTEGER' )

Values in this syntax are encoded as the decimal representation of their values, with each decimal digit represented by the its character equivalent. So the number 1321 is represented by the character string "1321".

## 6.17. JPEG

( 1.3.6.1.4.1.1466.115.121.1.28 DESC 'JPEG' )

Values in this syntax are encoded as strings containing JPEG images in the JPEG File Interchange Format (JFIF), as described in [8].

## 6.18. Matching Rule Description

( 1.3.6.1.4.1.1466.115.121.1.30 DESC 'Matching Rule Description' )

Values of type matchingRules are encoded as strings according to the BNF given in section 4.5.

6.19. Matching Rule Use Description

( 1.3.6.1.4.1.1466.115.121.1.31 DESC 'Matching Rule Use Description' )

Values of type matchingRuleUse are encoded as strings according to the BNF given in section 4.5.

6.20. MHS OR Address

( 1.3.6.1.4.1.1466.115.121.1.33 DESC 'MHS OR Address' )

Values in this syntax are encoded as strings, according to the format defined in [11].

6.21. Name And Optional UID

( 1.3.6.1.4.1.1466.115.121.1.34 DESC 'Name And Optional UID' )

Values in this syntax are encoded according to the following BNF:

```
NameAndOptionalUID = DistinguishedName [ "#" bitstring ]
```

Although the '#' character may occur in a string representation of a distinguished name, no additional special quoting is done. This syntax has been added subsequent to RFC 1778.

Example:

```
1.3.6.1.4.1.1466.0=#04024869,O=Test,C=GB#'0101'B
```

6.22. Name Form Description

( 1.3.6.1.4.1.1466.115.121.1.35 DESC 'Name Form Description' )

Values in this syntax are encoded according to the following BNF. Implementors should note that future versions of this document may have expanded this BNF to include additional terms.

```
NameFormDescription = "(" whsp
    numericoid whsp  ; NameForm identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" whsp ]
    "OC" woid         ; Structural ObjectClass
    "MUST" oids       ; AttributeTypes
    [ "MAY" oids ]    ; AttributeTypes
whsp ")"
```





6.23. Numeric String

( 1.3.6.1.4.1.1466.115.121.1.36 DESC 'Numeric String' )

The encoding of a string in this syntax is the string value itself. Example:

```
1997
```

6.24. Object Class Description

( 1.3.6.1.4.1.1466.115.121.1.37 DESC 'Object Class Description' )

Values in this syntax are encoded according to the BNF in section 4.4.

6.25. OID

( 1.3.6.1.4.1.1466.115.121.1.38 DESC 'OID' )

Values in the Object Identifier syntax are encoded according to the BNF in section 4.1 for "oid".

Example:

```
1.2.3.4
cn
```

6.26. Other Mailbox

( 1.3.6.1.4.1.1466.115.121.1.39 DESC 'Other Mailbox' )

Values in this syntax are encoded according to the following BNF:

```
otherMailbox = mailbox-type "$" mailbox

mailbox-type = printablestring

mailbox = <an encoded IA5 String>
```

In the above, mailbox-type represents the type of mail system in which the mailbox resides, for example "MCIMail"; and mailbox is the actual mailbox in the mail system defined by mailbox-type.

6.27. Postal Address

( 1.3.6.1.4.1.1466.115.121.1.41 DESC 'Postal Address' )

Values in this syntax are encoded according to the following BNF:

```
postal-address = dstring *( "$" dstring )
```

In the above, each dstring component of a postal address value is encoded as a value of type Directory String syntax. Backslashes and dollar characters, if they occur in the component, are quoted as described in section 4.3. Many servers limit the postal address to six lines of up to thirty characters.

Example:

```
1234 Main St.$Anytown, CA 12345$USA
\241,000,000 Sweepstakes$PO Box 1000000$Anytown, CA 12345$USA
```

### 6.28. Presentation Address

( 1.3.6.1.4.1.1466.115.121.1.43 DESC 'Presentation Address' )

Values in this syntax are encoded with the representation described in RFC 1278 [6].

### 6.29. Printable String

( 1.3.6.1.4.1.1466.115.121.1.44 DESC 'Printable String' )

The encoding of a value in this syntax is the string value itself. PrintableString is limited to the characters in production p of section 4.1.

Example:

```
This is a PrintableString
```

### 6.30. Telephone Number

( 1.3.6.1.4.1.1466.115.121.1.50 DESC 'Telephone Number' )

Values in this syntax are encoded as if they were Printable String types. Telephone numbers are recommended in X.520 to be in international form, as described in E.123 [15].

Example:

```
+1 512 305 0280
```





### 6.31. UTC Time

( 1.3.6.1.4.1.1466.115.121.1.53 DESC 'UTC Time' )

Values in this syntax are encoded as if they were printable strings with the strings containing a UTCTime value. This is historical; new attribute definitions SHOULD use GeneralizedTime instead.

### 6.32. LDAP Syntax Description

( 1.3.6.1.4.1.1466.115.121.1.54 DESC 'LDAP Syntax Description' )

Values in this syntax are encoded according to the BNF in section 4.3.3.

### 6.33. DIT Structure Rule Description

( 1.3.6.1.4.1.1466.115.121.1.17 DESC 'DIT Structure Rule Description' )

Values with this syntax are encoded according to the following BNF:

```
DITStructureRuleDescription = "(" whsp
    ruleidentifier whsp            ; DITStructureRule identifier
    [ "NAME" qdescrs ]
    [ "DESC" qdstring ]
    [ "OBSOLETE" whsp ]
    "FORM" woid whsp               ; NameForm
    [ "SUP" ruleidentifiers whsp ] ; superior DITStructureRules
")"

ruleidentifier = integer

ruleidentifiers = ruleidentifier |
    "(" whsp ruleidentifierlist whsp ")"

ruleidentifierlist = [ ruleidentifier *( ruleidentifier ) ]
```

## 7. Object Classes

Servers SHOULD recognize all the names of standard classes from section 7 of [12].

### 7.1. Extensible Object Class

The extensibleObject object class, if present in an entry, permits that entry to optionally hold any attribute. The MAY attribute list of this class is implicitly the set of all attributes.

```
( 1.3.6.1.4.1.1466.101.120.111 NAME 'extensibleObject'
  SUP top AUXILIARY )
```

The mandatory attributes of the other object classes of this entry are still required to be present.

Note that not all servers will implement this object class, and those which do not will reject requests to add entries which contain this object class, or modify an entry to add this object class.

### 7.2. subschema

This object class is used in the subschema entry.

```
( 2.5.20.1 NAME 'subschema' AUXILIARY
  MAY ( dITStructureRules $ nameForms $ ditContentRules $
  objectClasses $ attributeTypes $ matchingRules $
  matchingRuleUse ) )
```

The ldapSyntaxes operational attribute may also be present in subschema entries.

## 8. Matching Rules

Servers which implement the extensibleMatch filter SHOULD allow all the matching rules listed in this section to be used in the extensibleMatch. In general these servers SHOULD allow matching rules to be used with all attribute types known to the server, when the assertion syntax of the matching rule is the same as the value syntax of the attribute.

Servers MAY implement additional matching rules.

### 8.1. Matching Rules used in Equality Filters

Servers SHOULD be capable of performing the following matching rules.

For all these rules, the assertion syntax is the same as the value syntax.

```
( 2.5.13.0 NAME 'objectIdentifierMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

If the client supplies a filter using an objectIdentifierMatch whose matchValue oid is in the "descr" form, and the oid is not recognized by the server, then the filter is Undefined.

```
( 2.5.13.1 NAME 'distinguishedNameMatch'
```





```
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )

( 2.5.13.2 NAME 'caseIgnoreMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 2.5.13.8 NAME 'numericStringMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.36 )

( 2.5.13.11 NAME 'caseIgnoreListMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.41 )

( 2.5.13.14 NAME 'integerMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )

( 2.5.13.16 NAME 'bitStringMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.6 )

( 2.5.13.20 NAME 'telephoneNumberMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50 )

( 2.5.13.22 NAME 'presentationAddressMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.43 )

( 2.5.13.23 NAME 'uniqueMemberMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.34 )

( 2.5.13.24 NAME 'protocolInformationMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.42 )

( 2.5.13.27 NAME 'generalizedTimeMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )

( 1.3.6.1.4.1.1466.109.114.1 NAME 'caseExactIA5Match'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

( 1.3.6.1.4.1.1466.109.114.2 NAME 'caseIgnoreIA5Match'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

When performing the caseIgnoreMatch, caseIgnoreListMatch, telephoneNumberMatch, caseExactIA5Match and caseIgnoreIA5Match, multiple adjoining whitespace characters are treated the same as an individual space, and leading and trailing whitespace is ignored.

Clients MUST NOT assume that servers are capable of transliteration of Unicode values.

8.2. Matching Rules used in Inequality Filters

Servers SHOULD be capable of performing the following matching rules, which are used in greaterOrEqual and lessOrEqual filters.

```
( 2.5.13.28 NAME 'generalizedTimeOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )

( 2.5.13.3 NAME 'caseIgnoreOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

The sort ordering for a caseIgnoreOrderingMatch is implementation-dependent.

8.3. Syntax and Matching Rules used in Substring Filters

The Substring Assertion syntax is used only as the syntax of assertion values in the extensible match. It is not used as the syntax of attributes, or in the substring filter.

```
( 1.3.6.1.4.1.1466.115.121.1.58 DESC 'Substring Assertion' )
```

The Substring Assertion is encoded according to the following BNF:

```
substring = [initial] any [final]
initial = value
any = "*" *(value "*")
final = value
```

The <value> production is UTF-8 encoded string. Should the backslash or asterix characters be present in a production of <value>, they are quoted as described in section 4.3.

Servers SHOULD be capable of performing the following matching rules, which are used in substring filters.

```
( 2.5.13.4 NAME 'caseIgnoreSubstringsMatch'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )

( 2.5.13.21 NAME 'telephoneNumberSubstringsMatch'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )

( 2.5.13.10 NAME 'numericStringSubstringsMatch'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
```







8.4. Matching Rules for Subschema Attributes

Servers which allow subschema entries to be modified by clients MUST support the following matching rules, as they are the equality matching rules for several of the subschema attributes.

```
( 2.5.13.29 NAME 'integerFirstComponentMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )

( 2.5.13.30 NAME 'objectIdentifierFirstComponentMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

Implementors should note that the assertion syntax of these matching rules, an INTEGER or OID, is different from the value syntax of attributes for which this is the equality matching rule.

If the client supplies an extensible filter using an objectIdentifierFirstComponentMatch whose matchValue is in the "descr" form, and the OID is not recognized by the server, then the filter is Undefined.

9. Security Considerations

9.1. Disclosure

Attributes of directory entries are used to provide descriptive information about the real-world objects they represent, which can be people, organizations or devices. Most countries have privacy laws regarding the publication of information about people.

9.2. Use of Attribute Values in Security Applications

The transformations of an AttributeValue value from its X.501 form to an LDAP string representation are not always reversible back to the same BER or DER form. An example of a situation which requires the DER form of a distinguished name is the verification of an X.509 certificate.

For example, a distinguished name consisting of one RDN with one AVA, in which the type is commonName and the value is of the TeletexString choice with the letters 'Sam' would be represented in LDAP as the string CN=Sam. Another distinguished name in which the value is still 'Sam' but of the PrintableString choice would have the same representation CN=Sam.

Applications which require the reconstruction of the DER form of the value SHOULD NOT use the string representation of attribute syntaxes when converting a value to LDAP format. Instead it SHOULD use the

Binary syntax.

## 10. Acknowledgements

This document is based substantially on RFC 1778, written by Tim Howes, Steve Kille, Wengyik Yeong and Colin Robbins.

Many of the attribute syntax encodings defined in this and related documents are adapted from those used in the QUIPU and the IC R3 X.500 implementations. The contributions of the authors of both these implementations in the specification of syntaxes are gratefully acknowledged.







## 11. Authors' Addresses

Mark Wahl
Critical Angle Inc.
4815 West Braker Lane #502-385
Austin, TX 78759
USA

Phone: +1 512 372-3160
EMail: M.Wahl@critical-angle.com

Andy Coulbeck
Isode Inc.
9390 Research Blvd Suite 305
Austin, TX 78759
USA

Phone: +1 512 231-8993
EMail: A.Coulbeck@isode.com

Tim Howes
Netscape Communications Corp.
501 E. Middlefield Rd, MS MV068
Mountain View, CA 94043
USA

Phone: +1 650 937-3419
EMail: howes@netscape.com

Steve Kille
Isode Limited
The Dome, The Square
Richmond
TW9 1DT
UK

Phone: +44-181-332-9091
EMail: S.Kille@isode.com

## 12. Bibliography

[1] Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

[2] The Directory: Selected Attribute Types. ITU-T Recommendation X.520, 1993.

[3] The Directory: Models. ITU-T Recommendation X.501, 1993.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[5] Wahl, M., Kille, S., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[6] Kille, S., "A String Representation for Presentation Addresses", RFC 1278, November 1991.

[7] Terminal Equipment and Protocols for Telematic Services - Standardization of Group 3 facsimile apparatus for document transmission. CCITT, Recommendation T.4.

[8] JPEG File Interchange Format (Version 1.02). Eric Hamilton, C-Cube Microsystems, Milpitas, CA, September 1, 1992.

[9] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

[10] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993 (With amendments).

[11] Hardcastle-Kille, S., "Mapping between X.400(1988) / ISO 10021 and RFC 822", RFC 1327, May 1992.

[12] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997.

[13] Crocker, D., "Standard of the Format of ARPA-Internet Text Messages", STD 11, RFC 822, August 1982.

[14] ISO 3166, "Codes for the representation of names of countries".

[15] ITU-T Rec. E.123, Notation for national and international telephone numbers, 1988.





## 13. Full Copyright Statement