Network Working Group
Internet Draft: AP
Document: draft-earhart-ap-spec-01.txt
Expires July 1998

R. Earhart
Carnegie Mellon

January 1998

Access Protocol

Status of this Memo

Copyright

Abstract

The Access Protocol defines a standard extensible framework upon which application-specific protocols may be layered, providing a piece of infrastructure for a common class of internet protocols.

Attributions

Substantial portions of this protocol and of the text of this document come from [ACAP], which itself borrows much from [IMAP4].

1. Motivation

There are an increasing number of internet application-level protocols, solving a wide variety of problems. But as time goes on, it's becoming increasingly obvious that in the course of their development, regargless of their application-level purpose, many of the protocols need to solve the same infrastructure problems, such as

- Representation of commands (interleaving, protocol structure)
- Representation of command data
- Security (Authentication and authorization)
- Internationalization (UTF8, language control, etc.)
- Error reporting
- And a variety of minor issues (inactivity timeouts, etc.)

It's hoped that by defining a common infrastructure between application-specific command suites and the underlying stream protocol provided by services such as TCP, a number of these problems can be solved in a general way, allowing application-specific protocols to be more rapidly developed and deployed.

In addition, by abstracting the infrastructure from the application, it's hoped that each will be able to evolve independantly, and that the state of the art in protocol design will improve and advance faster than if each new infrastructure-level idea had to be individually incorporated into each application level protocol.

ASN.1/BER is *not* used, as there is a significant feeling in the applications area community that the complexity of these standards is a significant barrier to implementation.

It is recognized that not all application level protocols will fit into this model; TELNET is a good example of a protocol that does not belong in this framework. Nevertheless, it is believed that this is of sufficient utility to enough protocols to be worth advancing as an IETF standard.

2. Conventions Used in this Document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

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 [KEYWORDS].

3. Protocol Overview

3.1. Link Level

The Access Protocol assumes a reliable data stream such as provided by TCP. The command protocol that uses the AP is responsible for specifying any parameters to be used in constructing the stream.

3.2. Commands and Responses

An AP session consists of the establishment of a client/server 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.

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 AP client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line. Both clients and servers MUST be capable of handling lines of arbitrary length.

3.2.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with a identifier composed of one to thirty-two characters (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 section 4.1.3); in the other case, the command arguments require server feedback (see the AUTHENTICATE command). In some of these cases, the server sends a command continuation request if it is ready for the next part 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 or NO 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 or intermediate 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.

The server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result.

3.2.1. Server Protocol Sender and Client Protocol Receiver

Data transmitted by the server to the client come in four forms: command continuation requests, command completion results, intermediate responses, and untagged responses.

A command continuation request is prefixed with the token "+".

A command completion result 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).

An intermediate response returns data which can only be interpreted within the context of a command in progress. 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 an intermediate response identifies the command to which the response applies. A tagged response other than "OK", "NO", or "BAD" is an intermediate response.

An untagged response returns data or status messages which may be interpreted outside the context of a command in progress. It is prefixed with the token "*". Untagged 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 untagged data that resulted from a specific command and untagged data that were sent unilaterally.

The protocol receiver of an AP client reads a response line from the server. It then takes action on the response based upon the first token of the response, which may be a tag, a "*", or a "+" as described above.

A client MUST be prepared to accept any server response at all times. This includes untagged data that it may not have requested.

This topic is discussed in greater detail in the Server Responses section.

3.3. State and Flow Diagram

An AP server is in one of at least three states. Most commands are valid in only certain states. It is a protocol error for the client to attempt a command while the server is in an inappropriate state for that command. In this case, a server MUST respond with a BAD command completion result.

3.3.1. Non-Authenticated State

In non-authenticated state, the user must supply authentication credentials before most commands will be permitted. This state is entered when a connection starts.

3.3.2. Authenticated State

In authenticated state, the user is authenticated and most commands will be permitted. This state is entered when acceptable authentication credentials have been provided.

3.3.3. Logout State

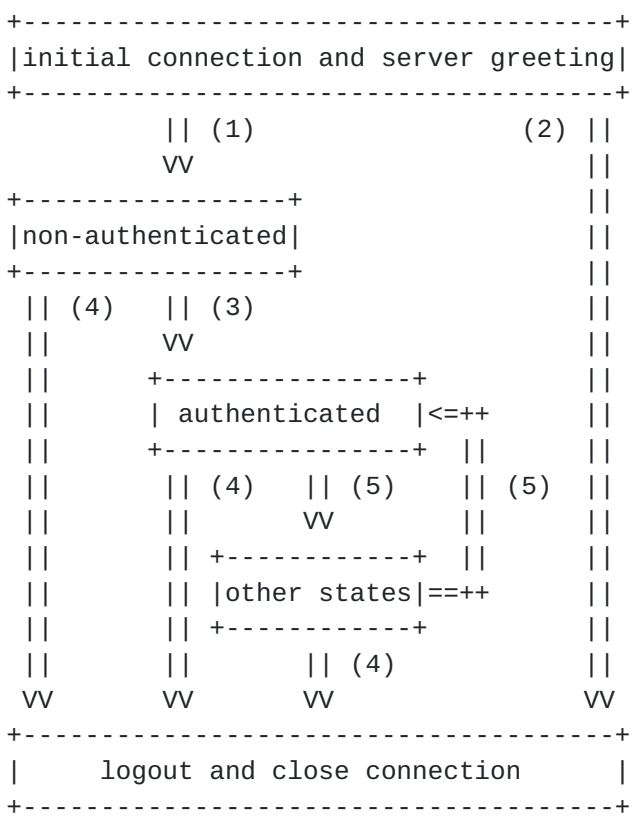
In logout state, the session is being terminated, and the server will close the connection. This state can be entered as a result of a client request or by unilateral server decision.

3.3.4. Other States

Protocols using AP MAY define more states, as desired. These states MUST only be reachable from the authenticated state, and MUST only transition between themselves, to the authenticated state, or to the logout state.

Protocol-specific states MUST only affect the operation of commands defined in those protocols, or in extensions to those protocols. In particular, the NOOP and LOGOUT commands MUST always be available.

Protocols MAY define new commands which transition to the logout state.



(1) connection (AP greeting)
(2) rejected connection (BYE greeting)
(3) successful AUTHENTICATE command
(4) LOGOUT or other closing command, server shutdown, or connection closed.
(5) State-transition command defined by protocol using AP

3.4. Operational Considerations

3.4.1. Untagged Status Updates

At any time, a server MAY send data that the client did not request. It is recognized that this will cause perfectly good TCP connections to be torn down if the network is unavailable for some transient reason; nevertheless, this is better than forcing the client to poll

the server for update information.

3.4.2. Response when No Command in Progress

Server implementations are permitted to send an untagged response 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.

3.4.3. Autologout Timer

Servers MAY implement an inactivity autologout timer. If such a timer is implemented, that timer MUST be at least 30 minutes' duration. The receipt of ANY data from the client during that interval MUST suffice to reset the autologout timer.

> Open Issue: is this really necessary? I'd rather forbid timers and the NOOP command, and say that it's the responsibility of the underlying stream protocol to ensure that the other side's still alive...)

3.4.4. Multiple Commands in Progress

The client is not required to wait for the completion result of a command before sending another command, subject to flow control constraints on the underlying data stream. Similarly, a server is not required to process a command to completion before beginning processing of the next command, although the server MUST compute the results of a command as though any changes caused by previous commands had taken place, and as though any changes caused by subsequent commands have not yet taken place.

Protocols which use this protocol as their basis SHOULD NOT define commands in such a way as to create an ambiguity when results from seperate commands are interlaced or reordered.

4. Protocol Elements

4.1. Data Formats

AP uses textual commands and responses. Data in AP can be in one of four forms: atom, number, string, parenthesized list, or NIL.

4.1.1. Atom

An atom consists of one to 1024 non-special characters.

4.1.2. Number

A number consists of one or more digit characters, and represents a numeric value.

4.1.3. 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 restrictions of what may be 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.

There are two forms of literals transmitted from client to server. The form where the open brace ("{") and number of octets is immediately followed by a close brace ("}") and CRLF is called a synchronizing literal. When sending a synchronizing literal, 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). The other form of literal, the non-synchronizing literal, is used to transmit a string from client to server without waiting for a command continuation request. The non-synchronizing literal differs from the synchronizing literal by having a plus ("+") between the number of octets and the close brace ("}") and by having the octet data immediately following the CRLF.

A quoted string is a sequence of zero to 1024 octets excluding CR, LF, double quote (<">), or backslash ("\") with double quote (<">) characters at each end.

The empty string is respresented as "" (a quoted string with zero characters between double quotes), as {0} followed by CRLF (a synchronizing literal with an octet count of 0), or as {0+} followed by a CRLF (a non-synchronizing literal with an octet count of 0).

> Note: Even if the octet count is 0, a client transmitting a synchronizing literal MUST wait to receive a command continuation

request.

4.1.4. 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.

4.1.5. 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 ().

4.2. Server Status Responses

Server status responses (defined in the ABNF as "status-response") MAY include an optional response code. A response code consists of data inside parentheses in the form of an atom, possibly followed by a space and arguments (defined in the ABNF as "resp-code"). The response code contains additional information or status codes for client software beyond the condition triggering the status response, 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:

AUTH-TOO-WEAK
: This response code is returned on a tagged NO result from an AUTHENTICATE command. It indicates that site security policy forbids the use of the requested mechanism for the specified authentication identity.

ENCRYPT-NEEDED
: This response code is returned on a tagged NO result from an AUTHENTICATE command. It indicates that site security policy requires the use of a strong encryption mechanism for the specified authentication identity and mechanism.

SASL This response code can occur in the tagged OK response to a successful AUTHENTICATE command and includes the optional final server response data from the server as specified by SASL [SASL].

TRANSITION-NEEDED
This response code occurs on a NO response to an AUTHENTICATE command. It indicates that the user name is valid, but the entry in the authentication database needs to be updated in order to permit authentication with the specified mechanism. This can happen if a user has an entry in a system authentication database such as Unix /etc/passwd, but does not have credentials suitable for use by the specified mechanism.

TRYLATER A command failed due to a temporary server failure. The client MAY continue using local information and try the command later.

Additional response codes MAY be defined by protocols layered on top of AP or by particular client or server implementations of those protocols. Additional response codes not defined in standards-track documents MUST be prefixed with an "X". Client implementations MUST ignore response codes that they do not recognize.

4.3. Server Command Continuation Request

The command continuation request is indicated by a "+" token instead of a tag. This 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 AUTHENTICATE 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 synchronizing literal. Protocols layered upon this protocol may define additional commands which use continuations, although these should be few and far between.

The client is not permitted to send the octets of a synchronizing 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, assuming it checks for non-synchronizing literals at the end of each line. 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.

5. Protocol Specification

5.1. Initial Connection

Upon session startup, the server sends one of two untagged responses: AP or BYE. The BYE response is documented in section 5.2.8.

Open Issue: I'm tempted to change this a little - have the client send its capabilities list in its greeting, and have the server send back a tagged OK, NO, BAD, or BYE response. This would cause negligable network load when used with TCP (the data could be carried in the initial TCP SYN packet which has to be sent anyway), and would allow the server to discover what the client's capable of, at the expense of changing the state diagram a little, hosing backwards compatibility, and adding an additional step for people accessing servers via telnet.

5.1.1. AP Untagged Response

Data: capability list

The untagged AP response indicates that the session is ready to accept commands and contains a space-separated listing of capbilities that the server supports. Each capability is an atom name, possibly followed by an argument in parenthesis (the argument MAY contain parenthesis, but the parenthesis MUST be balanced).

AP capability names MUST be defined in a standards track or IESG approved experimental RFC and registered with IANA according to the rules in section <section>.

Client implementations MAY require any capability names, but MUST ignore any unknown capability names. It is recommended that clients require as few capabilities as possible.

The following initial capabilities are defined:

IMPLEMENTATION
: The IMPLEMENTATION capability has one argument which is a string describing the server implementation. AP clients MUST NOT alter their behavior based on this value. It is intended primarily for debugging purposes.

SASL The SASL capability includes a list of the authentication mechanisms supported by the server. See [SASL] for more information.

Example: S: * AP IMPLEMENTATION ("ACME v3.5") SASL ("GSSAPI")

5.2. Any State

The following commands and responses are valid in any state.

5.2.1 NOOP Command

Arguments: none

Data: no specific data for this command (but see below)

Result: OK - noop completed
BAD - command unknown or arguments invalid

The NOOP command always succeeds. It does nothing. It can be used to reset any inactivity autologout timer on the server.

Example: C: a002 NOOP
S: a002 OK "NOOP completed"

5.2.2 LANG Command

Arguments: list of language preferences

Data: intermediate response: LANG

Result: OK - lang completed
NO - no matching language available
BAD - command unknown or arguments invalid

One or more arguments are supplied to indicate the client's preferred languages [LANG-TAGS] for error messages. The server will match each client preference in order against its internal table of available error string languages. For a client preference to match a server language, the client's language tag MUST be a prefix of the server's tag and match up to a "-" or the end of string. If a match is found, the server returns an intermediate LANG response and an OK response. The LANG response indicates the actual language selected.

If no LANG command is issued, all error text strings MUST be in the registered language "i-default" [CHARSET-LANG-POLICY], intended for an international audience.

Example: C: A003 LANG "fr-ca" "fr" "en-ca" "en-uk"
S: A003 LANG "fr-ca"
S: A003 OK "Bonjour"

5.2.3 LANG Intermediate Response

Data: language for error responses

The LANG response indicates the language which will be used for responses (in the ABNF, the final "quoted" element of resp-body).

5.2.4 LOGOUT Command

Arguments: none

Data: mandatory 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 session. 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 "Server logging out"
S: A023 OK "LOGOUT completed"

(Server and client then close the connection)

5.2.5. OK Response

Data: 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.

Example: S: * OK "Main disk is back on-line"

5.2.6. NO Response

Data: 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 may still complete successfully. The human-readable text describes the condition.

Example: C: A222 AUTHENTICATE "FOOBAR"
S: A222 NO "Unknown SASL mechanism"

5.2.7 BAD Response

Data: 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 may also indicate an internal server failure. The human-readable text describes the condition.

Example: C: ...empty line...
S: * BAD "Empty command line"
C: A443 BLURDYBLOOP
S: A443 BAD "Unknown command"
C: A444 NOOP Hello
S: A444 BAD "invalid arguments"

5.2.8. BYE Untagged Response

Data: optional response code
human-readable text

The untagged BYE response 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 may

be sent as part of a normal logout sequence, or as a panic shutdown announcement by the server. It SHOULD also used by server implementations as an announcement of an inactivity autologout.

This response is also used as one of two possible greetings at session startup. As a greeting, it indicates that the server is not willing to accept a session from this client.

Example: S: * BYE "Autologout; idle for too long"

5.2.9. ALERT Untagged Response

Data: optional response code
human-readable text

The human-readable text contains a special human generated alert message that MUST be presented to the user in a fashion that calls the user's attention to the message. This is intended to be used for vital messages from the server administrator to the user, such as a warning that the server will soon be shut down for maintenance.

Example: S: * ALERT "This server will be shut down in 10 minutes for system maintenance."

5.3. Non-Authenticated State

In non-authenticated state, the AUTHENTICATE command establishes authentication and enters authenticated state. The AUTHENTICATE command provides a general mechanism for a variety of authentication techniques.

Server implementations may allow non-authenticated access to certain information. The convention is to use an AUTHENTICATE command with the SASL ANONYMOUS mechanism [ANON].

Once authenticated (including as anonymous), it is not possible to re-enter non-authenticated state.

In addition to the universal commands (NOOP and LOGOUT), the only command valid in non-authenticated state is AUTHENTICATE.

5.3.1 AUTHENTICATE Command

Arguments: SASL mechanism name
optional initial response

Data: continuation data may 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 a SASL [SASL] authentication mechanism to the server. If the server supports the requested authentication mechanism, it performs an authentication protocol exchange to authenticate and identify the user. Optionally, it also negotiates a security layer for subsequent protocol interactions. If the requested authentication mechanism is not supported, the server rejects 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 with the "+" token followed by a string. The client answer consists of a line consisting of a string. If the client wishes to cancel an authentication exchange, it should issue a line with a single unquoted "*". If the server receives such an answer, it must reject the AUTHENTICATE command by sending a tagged BAD response.

The optional initial-response argument to the AUTHENTICATE command is used to save a round trip when using authentication mechanisms that are defined to send no data in the initial challenge. When the initial-response argument is used with such a mechanism, the initial empty challenge is not sent to the client and the server uses the data in the initial-response argument as if it were sent in response to the empty challenge. If the initial-response argument to the AUTHENTICATE command is used with a mechanism that sends data in the initial challenge, the server rejects the AUTHENTICATE command by sending a tagged NO response.

The service name specified by this protocol's profile of SASL is "ap".

If a security layer is negotiated through the SASL authentication exchange, it 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.

All AP implementations MUST implement the CRAM-MD5 SASL mechanism [CRAM-MD5], although they MAY offer a configuration option to disable it if site security policy dictates. The example below is the same example described in the CRAM-MD5 specification.

If an AUTHENTICATE command fails with a NO response, the client may try another authentication mechanism by issuing another AUTHENTICATE command. In other words, the client may request authentication types in decreasing order of preference.

```
Example:  S: * OK IMPLEMENTATION ("Blorfysoft v3.5")
             SASL ("CRAM-MD4" "KERBEROS_V4")
          C: A001 AUTHENTICATE "CRAM-MD5"
          S: + "1896.697170952@postoffice.reston.mci.net>"
          C: "tim b913a602c7eda7a495b4e6e7334d3890"
          S: A001 OK "CRAM-MD5 authentication successful"

          Note: the line breaks in the first client answer are for
          editorial clarity and are not in real authenticators.
```

5.3. Authenticated State

In the authenticated state, the universal commands (NOOP and LOGOUT) are valid, in addition to any commands defined by protocols that use AP as their foundation.

6. Design Philosophy

Protocols layered on top of AP SHOULD define a set of commands to be valid in the authenticated state. In addition, protocols MAY define an atom to be returned in the initial AP greeting, possibly allowing multiple protocols to be used over the same connection.

Ideally, protocols should limit themselves as much as possible to a simple, uncomplicated suite of commands that relate to each other. Where possible, protocols should be broken up into orthogonal components, such that the components may be reused in other protocols.

Example: Instead of defining an advisory lock mechanism, advisory locking should be split into a seperate extension, useable by whatever protocols happen to require it.

Quota management is another set of commands that could be written as an extension and made available to all protocols

that involve quotas.

New commands and responses SHOULD only be defined for the Authenticated state.

Responses to commands SHOULD be tagged. This is essential for allowing multiple commands to execute simultaneously, and experience shows that this leads to much simpler implementations for both clients and servers.

Where textual data is exchanged, protocols SHOULD use UTF8 [UTF8] whenever possible, for internationalization.

New command completion responses MUST NOT be defined -- every command MUST be completed by an "OK", "NO", or "BAD" response.

7. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) notation as specified in [ABNF] This uses the ABNF core rules as specified in Appendix A of the ABNF specification [ABNF].

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.

Protocols based on AP should refer to this formal syntax, and augment selected parts via the ABNF "=/" operator, as indicated in the comments.

The client produces a sequence of octets matching "command-client"; the server consumes these, and returns a sequence of octets matching "response-server".

Protocols using AP MAY augment "capability" (subject to the requirement that "capability" MUST match "capability-generic"), "command" (subject to the requirement that "command" MUST match "command-generic"), "response" (subject to the requirement that "response" MUST match "response-generic"), "resp-code" (subject to the requirement that "resp-code" MUST match "resp-code-generic"), or "status-response" (subject to the requirement that "status-response" MUST match "status-response-generic"). Other syntax elements SHOULD NOT be redefined.

For readability, rules which MAY be augmented are defined using the

"=/" operator; all other rules are defined using "=".

A number of symbols are defined solely for use by protocols using AP.

```
   ATOM-CHAR        = "!" / %x23-27 / %x2A-5B / %x5D-7A / %x7C-7E
                      ; Any CHAR except ATOM-SPECIALS

   ATOM-SPECIALS    = "(" / ")" / "{" / SPACE / CTL / QUOTED-SPECIALS

   DIGIT-NZ         = %x31-39
                        ; 1-9

   QUOTED-CHAR      = SAFE-UTF8-CHAR / "\" QUOTED-SPECIALS

   QUOTED-SPECIALS  = <"> / "\"

   SAFE-CHAR        = %x01-09 / %x0B-0C / %x0E-21 /
                      %x23-5B / %x5D-7F
                      ; Any CHAR except CR, LF, or QUOTED-SPECIALS

   SAFE-UTF8-CHAR   = SAFE-CHAR / UTF8-2 / UTF8-3 / UTF8-4 /
                      UTF8-5 / UTF8-6
   TAG-CHAR         = %x21 / %x23-27 / %x2C-5B / %x5D-7A / %x7C-7E
                      ; Any ATOM-CHAR except "*" or "+"

   TEXT-UTF8-CHAR   = SAFE-UTF8-CHAR / QUOTED-SPECIALS

   UTF8-1           = %x80-BF

   UTF8-2           = %xC0-DF UTF8-1

   UTF8-3           = %xE0-EF 2UTF8-1

   UTF8-4           = %xF0-F7 3UTF8-1

   UTF8-5           = %xF8-FB 4UTF8-1

   UTF8-6           = %xFC-FD 5UTF8-1

   UTF8-CHAR        = TEXT-UTF8-CHAR / CR / LF

   argument         = atom
                    / string
                    / number
                    / "(" [argument *(SP argument)] ")"

   atom             = 1*1024ATOM-CHAR
```

```
auth-type        = <"> auth-type-name <">

auth-type-name   = iana-token
                   ; As defined in [SASL]

capability       =/ "IMPLEMENTATION" SP "(" quoted ")"

capability       =/ "SASL" SP "(" auth-type *(SP auth-type) ")"

     ; Other capabilities MAY be defined by protocols using AP,
     ; but MUST syntactically match capability-generic

capability-arg   = atom
                 / quoted
                 / "(" [capability-arg *(SP capability-arg)] ")"

capability-generic = atom [SP "(" [capability-arg] ")"]

command-client   = tag SP command CRLF

command          =/ "NOOP"

command          =/ "LOGOUT"

command          =/ "AUTHENTICATE" SP auth-type
                    [SP string] *(CRLF string)

     ; Other commands MAY be defined by protocols using AP,
     ; but MUST syntactically match command-generic

command-generic  = atom *(SP argument)

iana-token       = atom
                   ; MUST be registered with IANA

literal          = "{" number [ "+" ] "}" CRLF *OCTET
                   ; The number represents the number of octets

literal-utf8     = "{" number [ "+" ] "}" CRLF *UTF8-CHAR
                   ; The number represents the number of octets,
                   ; not the number of characters

nil              = "NIL"

number           = 1*DIGIT

nz-number        = DIGIT-NZ *DIGIT
```

```
quoted          = <"> *QUOTED-CHAR <">

resp-argument   = atom
                / quoted
                / number
                / "(" [resp-argument *(SP resp-argument)] ")"

resp-body       = SP ["(" resp-code ")" SP] quoted

resp-code       =/ "AUTH-TOO-WEAK"

resp-code       =/ "ENCRYPT-NEEDED"

resp-code       =/ "SASL"

resp-code       =/ "TRANSITION-NEEDED"

resp-code       =/ "TRYLATER"

     ; Other resp-codes MAY be defined by protocols using AP,
     ; but MUST syntactically match resp-code-generic

resp-code-generic = atom *(SP resp-argument)

response         =/ "AP" *(SP capability)

response         =/ status-response

     ; Other responses MAY be defined by protocols using AP,
     ; but MUST syntactically match response-generic

response-generic = atom *(SP argument)

response-server  = (tag / "*") response CRLF

status-response  =/ "OK" resp-body

status-response  =/ "NO" resp-body

status-response  =/ "BAD" resp-body

status-response  =/ "BYE" resp-body

status-response  =/ "ALERT" resp-body

     ; Other status-responses MAY be defined by protocols using AP,
     ; but MUST syntactically match status-response-generic
```

```
status-response-generic = atom resp-body

string          = quoted / literal

string-utf8     = quoted / literal-utf8

tag             = 1*32TAG-CHAR
```

8. Security Considerations

AP protocol transactions are sent in the clear over the network unless some form of privacy protection is negotiated in the AUTHENTICATE command.

AP's security is defined by [SASL], and thus has the same security considerations.

9. References

[ABNF] Crocker, D., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

<url:ftp://ds.internic.net/rfc/rfc2234.txt>

[ACAP] Myers, J., and Newman, C., "Application Configuration Access Protocol (ACAP)", RFC 2244, November 1997.

<url:ftp://ds.internic.net/rfc/rfc2244.txt>

[ANON] Newman, C., "Anonymous SASL Mechanism", RFC 2245, November 1997.

<url:ftp://ds.internic.net/rfc/rfc2245.txt>

[CHARSET-LANG-POLICY] Alvestrand, H., "IETF Policy on Character Sets and Languages", work in progress.

[COMPARATOR] Newman, C., Myers, J., "Comparators", work in progress.

[CRAM-MD5] Klensin, J., Catoe, R., and Krumviede, P., "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 1997.

<url:ftp://ds.internic.net/rfc/rfc2195.txt>

[IMAP4] Crispin, M., "Internet Message Access Protocol - Version

4rev1", RFC 2060, December 1996.

<url:ftp://ds.internic.net/rfc/rfc2060.txt>

[LANG-TAGS] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

<url:ftp://ds.internic.net/rfc/rfc1766.txt>

[SASL] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

<url:ftp://ds.internic.net/rfc/rfc2222.txt>

[UTF8] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

<url:ftp://ds.internic.net/rfc/rfc2044.txt>

10. Full Copyright Statement

11. Author's Address

Robert H. Earhart
Carnegie Mellon
5000 Forbes Ave.
Pittsburgh PA, 15213-3890

Email: earhart+@cmu.edu

Expires July 1998