Network Working Group
Request for Comments: 2244
Category: Standards Track

C. Newman
Innosoft
J. G. Myers
Netscape

November 1997

ACAP -- Application Configuration Access Protocol

Status of this Memo

Copyright Notice

Abstract

The Application Configuration Access Protocol (ACAP) is designed to support remote storage and access of program option, configuration and preference information. The data store model is designed to allow a client relatively simple access to interesting data, to allow new information to be easily added without server re-configuration, and to promote the use of both standardized data and custom or proprietary data. Key features include "inheritance" which can be used to manage default values for configuration settings and access control lists which allow interesting personal information to be shared and group information to be restricted.

Table of Contents

ACAP Protocol Specification

1. Introduction

1.1. Conventions Used in this Document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. If such lines are wrapped without a new "C:" or "S:" label, then the wrapping is for editorial clarity and is not part of the command.

The key words "REQUIRED", "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [KEYWORDS].

1.2. ACAP Data Model

An ACAP server exports a hierarchical tree of entries. Each level of the tree is called a dataset, and each dataset is made up of a list of entries. Each entry has a unique name and may contain any number of named attributes. Each attribute within an entry may be single valued or multi-valued and may have associated metadata to assist access and interpretation of the value.

The rules with which a client interprets the data within a portion of ACAP's tree of entries are called a dataset class.

1.3. ACAP Design Goals

ACAP's primary purpose is to allow users access to their configuration data from multiple network-connected computers. Users can then sit down in front of any network-connected computer, run any ACAP-enabled application and have access to their own configuration data. Because it is hoped that many applications will become ACAP-enabled, client simplicity was preferred to server or protocol simplicity whenever reasonable.

ACAP is designed to be easily manageable. For this reason, it includes "inheritance" which allows one dataset to inherit default attributes from another dataset. In addition, access control lists are included to permit delegation of management and quotas are included to control storage. Finally, an ACAP server which is conformant to this base specification should be able to support most dataset classes defined in the future without requiring a server reconfiguration or upgrade.

ACAP is designed to operate well with a client that only has intermittent access to an ACAP server. For this reason, each entry has a server maintained modification time so that the client may detect changes. In addition, the client may ask the server for a list of entries which have been removed since it last accessed the server.

ACAP presumes that a dataset may be potentially large and/or the client's network connection may be slow, and thus offers server sorting, selective fetching and change notification for entries within a dataset.

As required for most Internet protocols, security, scalability and internationalization were important design goals.

Given these design goals, an attempt was made to keep ACAP as simple as possible. It is a traditional Internet text based protocol which massively simplifies protocol debugging. It was designed based on the successful IMAP [IMAP4] protocol framework, with a few refinements.

1.4. Validation

By default, any value may be stored in any attribute for which the user has appropriate permission and quota. This rule is necessary to allow the addition of new simple dataset classes without reconfiguring or upgrading the server.

In some cases, such as when the value has special meaning to the server, it is useful to have the server enforce validation by returning the INVALID response code to a STORE command. These cases MUST be explicitly identified in the dataset class specification which SHOULD include specific fixed rules for validation. Since a given ACAP server may be unaware of any particular dataset class specification, clients MUST NOT depend on the presence of enforced validation on the server.

1.5. Definitions

access control list (ACL)
: A set of identifier, rights pairs associated with an object. An ACL is used to determine which operations a user is permitted to perform on that object. See section 3.5.

attribute
: A named value within an entry. See section 3.1.

comparator
: A named function which can be used to perform one or more of three comparison operations: ordering, equality and substring matching. See section 3.4.

context
: An ordered subset of entries in a dataset, created by a SEARCH command with a MAKECONTEXT modifier. See section 3.3.

dataset
: One level of hierarchy in ACAP's tree of entries.

dataset class specification
: The rules which allow a client to interpret the data within a portion of ACAP's tree of entries.

entry
: A set of attributes with a unique entry name. See section 3.1.

metadata
: Information describing an attribute, its value and any access controls associated with that attribute. See section 3.1.2.

NIL
: This represents the non-existence of a particular data item.

NUL
: A control character encoded as 0 in US-ASCII [US-ASCII].

octet
: An 8-bit value. On most modern computer systems, an octet is one byte.

SASL
: Simple Authentication and Security Layer [SASL].

UTC
: Universal Coordinated Time as maintained by the Bureau International des Poids et Mesures (BIPM).

UTF-8
: An 8-bit transformation format of the Universal Character Set [UTF8]. Note that an incompatible change was made to the coded character set referenced by [UTF8], so for the purpose of this document, UTF-8 refers to the UTF-8 encoding as defined by version 2.0 of Unicode [UNICODE-2], or ISO 10646 [ISO-10646] including amendments one through seven.

1.6. ACAP Command Overview

The AUTHENTICATE, NOOP, LANG and LOGOUT commands provide basic protocol services. The SEARCH command is used to select, sort, fetch and monitor changes to attribute values and metadata. The UPDATECONTEXT and FREECONTEXT commands are also used to assist in monitoring changes in attribute values and metadata. The STORE command is used to add, modify and delete entries and attributes. The DELETEDSINCE command is used to assist a client in re-synchronizing a cache with the server. The GETQUOTA, SETACL, DELETEACL, LISTRIGHTS and MYRIGHTS commands are used to examine storage quotas and examine or modify access permissions.

2. Protocol Framework

2.1. Link Level

The ACAP protocol assumes a reliable data stream such as provided by TCP. When TCP is used, an ACAP server listens on port 674.

2.2. Commands and Responses

An ACAP 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.

ACAP is a text-based line-oriented protocol. In general, interactions transmitted by clients and servers are in the form of lines; that is, sequences of characters that end with a CRLF. The protocol receiver of an ACAP client or server is either reading a line, or is reading a sequence of octets with a known count (a literal) followed by a line. Both clients and servers must be capable of handling lines of arbitrary length.

2.2.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with a identifier (an alphanumeric string of no more than 32 characters, e.g., A0001, A0002, etc.) called a "tag". A different tag SHOULD be 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 2.6.3); 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 a 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 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 ACAP server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result.

2.2.2. 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 ACAP 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.

2.3. Server States

An ACAP server is in one of 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 will respond with a BAD command completion result.

2.3.1. Non-Authenticated State

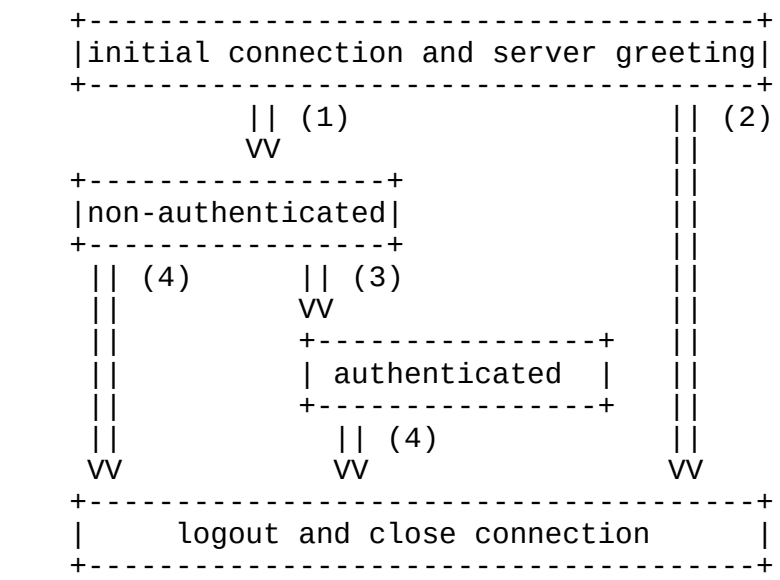
In non-authenticated state, the user must supply authentication credentials before most commands will be permitted. This state is entered when a connection starts.

2.3.2. 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.

2.3.3. 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.



(1) connection (ACAP greeting)
(2) rejected connection (BYE greeting)
(3) successful AUTHENTICATE command
(4) LOGOUT command, server shutdown, or connection closed

2.4. Operational Considerations

2.4.1. Untagged Status Updates

At any time, a server can send data that the client did not request.

2.4.2. 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.

2.4.3. Auto-logout Timer

If a server has an inactivity auto-logout timer, that timer MUST be of at least 30 minutes duration. The receipt of ANY command from the client during that interval MUST suffice to reset the auto-logout timer.

2.4.4. 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, unless an ambiguity would result because of a command that would affect the results of other commands. If there is such an ambiguity, the server executes commands to completion in the order given by the client.

2.5. 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.

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 (see section 2.6.3).

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.

```
Example:    C: A099 FREECONTEXT {10}
            S: + "Ready for additional command text"
            C: FRED
            C: FOOB
            S: A099 OK "FREECONTEXT completed"
            C: A044 BLURDYBLOOP {102856}
            S: A044 BAD "No such command as 'BLURDYBLOOP'"
```

2.6. Data Formats

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

2.6.1. Atom

An atom consists of one to 1024 non-special characters. It must begin with a letter. Atoms are used for protocol keywords.

2.6.2. Number

A number consists of one or more digit characters, and represents a numeric value. Numbers are restricted to the range of an unsigned 32-bit integer: 0 < number < 4,294,967,296.

2.6.3. 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 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 NUL, CR and LF, with double quote (<">) characters at each end.

The empty string is represented 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.

2.6.3.1. 8-bit and Binary Strings

Most strings in ACAP are restricted to UTF-8 characters and may not contain NUL octets. Attribute values MAY contain any octets including NUL.

2.6.4. 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.

2.6.5. 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 ().

3. Protocol Elements

This section defines data formats and other protocol elements used throughout the ACAP protocol.

3.1. Entries and Attributes

Within a dataset, each entry name is made up of zero or more UTF-8 characters other than slash ("/"). A slash separated list of entries, one at each level of the hierarchy, forms the full path to an entry.

Each entry is made up of a set of attributes. Each attribute has a hierarchical name in UTF-8, with each component of the name separated by a period (".").

The value of an attribute is either single or multi-valued. A single value is NIL (has no value), or a string of zero or more octets. A multi-value is a list of zero or more strings, each of zero or more octets.

Attribute names are not permitted to contain asterisk ("*") or percent ("%") and MUST be valid UTF-8 strings which do not contain NUL. Invalid attribute names result in a BAD response. Entry names

are not permitted to begin with "." or contain slash ("/") and MUST be valid UTF-8 strings which do not contain NUL. Invalid entry names in the entry field of a command result in a BAD response.

Use of non-visible UTF-8 characters in attribute and entry names is discouraged.

3.1.1. Predefined Attributes

Attribute names which do not contain a dot (".") are reserved for standardized attributes which have meaning in any dataset. The following attributes are defined by the ACAP protocol.

entry
: Contains the name of the entry. MUST be single valued. Attempts to use illegal or multi-valued values for the entry attribute are protocol errors and MUST result in a BAD completion response. This is a special case.

modtime
: Contains the date and time any read-write metadata in the entry was last modified. This value MUST be in UTC, MUST be automatically updated by the server.

  The value consists of 14 or more US-ASCII digits. The first four indicate the year, the next two indicate the month, the next two indicate the day of month, the next two indicate the hour (0 - 23), the next two indicate the minute, and the next two indicate the second. Any further digits indicate fractions of a second.

  The time, particularly fractions of a second, need not be accurate. It is REQUIRED, however, that any two entries in a dataset changed by successive modifications have strictly ascending modtime values. In addition, each STORE command within a dataset (including simultaneous stores from different connections) MUST use different modtime values.

  This attribute has enforced validation, so any attempt to STORE a value in this attribute MAY result in a NO response with an INVALID response code.

subdataset
: If this attribute is set, it indicates the existence of a sub-dataset of this entry.

The value consists of a list of relative ACAP URLs (see section 3.2) which may be used to locate the sub-dataset. The base URL is the full path to the entry followed by a slash ("/"). The value "." indicates a subdataset is located directly under this one. Multiple values indicate replicated copies of the subdataset.

For example, if the dataset "/folder/site/" has an entry "public-folder" with a subdataset attribute of ".", then there exists a dataset "/folder/site/public-folder/". If the value of the subdataset attribute was instead "//other.acap.domain//folder/site/public-folder/", that would indicate the dataset is actually located on a different ACAP server.

A dataset can be created by storing a "subdataset" attribute including ".", and a sub-hierarchy of datasets is deleted by storing a NIL value to the "subdataset" attribute on the entry in the parent dataset.

This attribute has enforced syntax validation. Specifically, if an attempt is made to STORE a non-list value (other than NIL), an empty list, or one of the values does not follow the URL syntax rules [BASIC-URL, REL-URL], then this will result in a NO response with an INVALID response code.

3.1.2. Attribute Metadata

Each attribute is made up of metadata items which describe that attribute, its value and any associated access controls. Metadata items may be either read-only, in which case the client is never permitted to modify the item, or read-write, in which case the client may modify the item if the access control list (ACL) permits.

The following metadata items are defined in this specification:

acl
: The access control list for the attribute, if one exists. If the attribute does not have an ACL, NIL is returned. Read-write. See section 3.5 for the contents of an ACL.

attribute
: The attribute name. Read-only.

myrights
: The set of rights that the client has to the attribute. Read-only. See section 3.5 for the possible rights.

size This is the length of the value. In the case of a multi-value, this is a list of lengths for each of the values. Read-only.

value The value. For a multi-value, this is a list of single values. Read-write.

Additional items of metadata may be defined in extensions to this protocol. Servers MUST respond to unrecognized metadata by returning a BAD command completion result.

3.2. ACAP URL scheme

ACAP URLs are used within the ACAP protocol for the "subdataset" attribute, referrals and inheritance. They provide a convenient syntax for referring to other ACAP datasets. The ACAP URL follows the common Internet scheme syntax as defined in [BASIC-URL] except that plaintext passwords are not permitted. If :<port> is omitted, the port defaults to 674.

An ACAP URL has the following general form:

```
url-acap  = "acap://" url-server "/" url-enc-entry [url-filter]
            [url-extension]
```

The <url-server> element includes the hostname, and optional user name, authentication mechanism and port number. The <url-enc-entry> element contains the name of an entry path encoded according to the rules in [BASIC-URL].

The <url-filter> element is an optional list of interesting attribute names. If omitted, the URL refers to all attributes of the named entry. The <url-extension> element is reserved for extensions to this URL scheme.

Note that unsafe or reserved characters such as " " or "?" MUST be hex encoded as described in the URL specification [BASIC-URL]. Hex encoded octets are interpreted according to UTF-8 [UTF8].

3.2.1. ACAP URL User Name and Authentication Mechanism

A user name and/or authentication mechanism may be supplied. They are used in the "AUTHENTICATE" command after making the connection to the ACAP server. If no user name or authentication mechanism is supplied, then the SASL ANONYMOUS [SASL-ANON] mechanism is used by default. If an authentication mechanism is supplied without a user

name, then one SHOULD be obtained from the specified mechanism or requested from the user as appropriate. If a user name is supplied without an authentication mechanism then ";AUTH=*" is assumed.

The ";AUTH=" authentication parameter is interpreted as described in the IMAP URL Scheme [IMAP-URL].

Note that if unsafe or reserved characters such as " " or ";" are present in the user name or authentication mechanism, they MUST be encoded as described in the URL specification [BASIC-URL].

3.2.2. Relative ACAP URLs

Because ACAP uses "/" as the hierarchy separator for dataset paths, it works well with the relative URL rules defined in the relative URL specification [REL-URL].

The <aauth> grammar element is considered part of the user name for purposes of resolving relative ACAP URLs.

The base URL for a relative URL stored in an attribute's value is formed by taking the path to the dataset containing that attribute, appending a "/" followed by the entry name of the entry containing that attribute followed by "/".

3.3. Contexts

A context is subset of entries in a dataset or datasets, created by a SEARCH command with a MAKECONTEXT modifier. Context names are client-generated strings and must not start with the slash ('/') character.

When a client creates a context, it may request automatic notification of changes. A client may also request enumeration of entries within a context. Enumeration simplifies the implementation of a "virtual scrollbar" by the client.

A context exists only within the ACAP session in which it was created. When the connection is closed, all contexts associated with that connection are automatically discarded. A server is required to support at least 100 active contexts within a session. If the server supports a larger limit it must advertise it in a CONTEXTLIMIT capability.

3.4. Comparators

A comparator is a named function which takes two input values and can be used to perform one or more of four comparison operations: ordering, equality, prefix and substring matching.

The ordering operation is used both for the SORT search modifier and the COMPARE and COMPARESTRICT search keys. Ordering comparators can determine the ordinal precedence of any two values. When used for ordering, a comparator's name can be prefixed with "+" or "-" to indicate that the ordering should be normal order or reversed order respectively. If no prefix is included, "+" is assumed.

For the purpose of ordering, a comparator may designate certain values as having an undefined ordinal precedence. Such values always collate with equal value after all other values regardless of whether normal or reversed ordering is used. Unless the comparator definition specifies otherwise, multi-values and NIL values have an undefined ordinal precedence.

The equality operation is used for the EQUAL search modifier, and simply determines if the two values are considered equal under the comparator function. When comparing a single value to a multi-value, the two are considered equal if any one of the multiple values is equal to the single value.

The prefix match operation is used for the PREFIX search modifier, and simply determines if the search value is a prefix of the item being searched. In the case of prefix search on a multi-value, the match is successful if the value is a prefix of any one of the multiple values.

The substring match operation is used for the SUBSTRING search modifier, and simply determines if search value is a substring of the item being searched. In the case of substring search on a multi-value, the match is successful if the value is a substring of any one of the multiple values.

Rules for naming and registering comparators will be defined in a future specification. Servers MUST respond to unknown or improperly used comparators with a BAD command completion result.

The following comparators are defined by this standard and MUST be implemented:

i;octet
Operations: Ordering, Equality, Prefix match, Substring match

For collation, the i;octet comparator interprets the value of an attribute as a series of unsigned octets with ordinal values from 0 to 255. When ordering two strings, each octet pair is compared in sequence until the octets are unequal or the end of the string is reached. When collating two strings where the shorter is a prefix of the longer, the shorter string is interpreted as having a smaller ordinal value. The "i;octet" or "+i;octet" forms collate smaller ordinal values earlier, and the "-i;octet" form collates larger ordinal values earlier.

For the equality function, two strings are equal if they are the same length and contain the same octets in the same order. NIL is equal only to itself.

For non-binary, non-nil single values, i;octet ordering is equivalent to the ANSI C [ISO-C] strcmp() function applied to C string representations of the values. For non-binary, non-nil single values, i;octet substring match is equivalent to the ANSI C strstr() function applied to the C string representations of the values.

i;ascii-casemap
Operations: Ordering, Equality, Prefix match, Substring match

The i;ascii-casemap comparator first applies a mapping to the attribute values which translates all US-ASCII letters to uppercase (octet values 0x61 to 0x7A are translated to octet values 0x41 to 0x5A respectively), then applies the i;octet comparator as described above. With this function the values "hello" and "HELLO" have the same ordinal value and are considered equal.

i;ascii-numeric
Operations: Ordering, Equality

The i;ascii-numeric comparator interprets strings as decimal positive integers represented as US-ASCII digits. All values which do not begin with a US-ASCII digit are considered equal with an ordinal value higher than all non-NIL single-valued

attributes. Otherwise, all US-ASCII digits (octet values 0x30 to 0x39) are interpreted starting from the beginning of the string to the first non-digit or the end of the string.

3.5. Access Control Lists (ACLs)

An access control list is a set of identifier, rights pairs used to restrict access to a given dataset, attribute or attribute within an entry. An ACL is represented by a multi-value with each value containing an identifier followed by a tab character followed by the rights. The syntax is defined by the "acl" rule in the formal syntax in section 8.

Identifier is a UTF-8 string. The identifier "anyone" is reserved to refer to the universal identity (all authentications, including anonymous). All user name strings accepted by the AUTHENTICATE command to authenticate to the ACAP server are reserved as identifiers for the corresponding user. Identifiers starting with a slash ("/") character are reserved for authorization groups which will be defined in a future specification. Identifiers MAY be prefixed with a dash ("-") to indicate a revocation of rights. All other identifiers have implementation-defined meanings.

Rights is a string listing a (possibly empty) set of alphanumeric characters, each character listing a set of operations which is being controlled. Letters are reserved for "standard" rights, listed below. The set of standard rights may only be extended by a standards-track or IESG approved experimental RFC. Digits are reserved for implementation or site defined rights. The currently defined standard rights are:

x - search (use EQUAL search key with i;octet comparator)
r - read (access with SEARCH command)
w - write (modify with STORE command)
i - insert (perform STORE on a previously NIL value)
a - administer (perform SETACL or STORE on ACL attribute/metadata)

An implementation may force rights to always or never be granted. In particular, implementations are expected to grant implicit read and administer rights to a user’s personal dataset storage in order to avoid denial of service problems. Rights are never tied, unlike the IMAP ACL extension [IMAP-ACL].

It is possible for multiple identifiers in an access control list to apply to a given user (or other authentication identity). For example, an ACL may include rights to be granted to the identifier matching the user, one or more implementation-defined identifiers

matching groups which include the user, and/or the identifier "anyone". These rights are combined by taking the union of all positive rights which apply to a given user and subtracting the union of all negative rights which apply to that user. A client MAY avoid this calculation by using the MYRIGHTS command and metadata items.

Each attribute of each entry of a dataset may potentially have an ACL. If an attribute in an entry does not have an ACL, then access is controlled by a default ACL for that attribute in the dataset, if it exists. If there is no default ACL for that attribute in the dataset, access is controlled by a default ACL for that dataset. The default ACL for a dataset must exist.

In order to perform any access or manipulation on an entry in a dataset, the client must have 'r' rights on the "entry" attribute of the entry. Implementations should take care not to reveal via error messages the existence of an entry for which the client does not have 'r' rights. A client does not need access to the "subdataset" attribute of the parent dataset in order to access the contents of a dataset.

Many of the ACL commands and responses include an "acl object" parameter, for specifying what the ACL applies to. This is a parenthesized list. The list contains just the dataset name when referring to the default ACL for a dataset. The list contains a dataset name and an attribute name when referring to the default ACL for an attribute in a dataset. The list contains a dataset name, an attribute name, and an entry name when referring to the ACL for an attribute of an entry of a dataset.

3.6. Server Response Codes

An OK, NO, BAD, ALERT or BYE response from the server MAY contain a response code to describe the event in a more detailed machine parsable fashion. A response code consists of data inside parentheses in the form of an atom, possibly followed by a space and arguments. Response codes are defined when there is a specific action that a client can take based upon the additional information. In order to support future extension, the response code is represented as a slash-separated hierarchy with each level of hierarchy representing increasing detail about the error. Clients MUST tolerate additional hierarchical response code detail which they don't understand.

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.

INVALID
: This response code indicates that a STORE command included data which the server implementation does not permit. It MUST NOT be used unless the dataset class specification for the attribute in question explicitly permits enforced server validation. The argument is the attribute which was invalid.

MODIFIED
: This response code indicates that a conditional store failed because the modtime on the entry is later than the modtime specified with the STORE command UNCHANGEDSINCE modifier. The argument is the entry which had been modified.

NOEXIST
: This response code indicates that a search or NOCREATE store failed because a specified dataset did not exist. The argument is the dataset which does not exist.

PERMISSION
: A command failed due to insufficient permission based on the access control list or implicit rights. The argument is the acl-object which caused the permission failure.

QUOTA
: A STORE or SETACL command which would have increased the size of the dataset failed due to insufficient quota.

REFER
: This response code may be returned in a tagged NO response to any command that takes a dataset name as a parameter. It has one or more arguments with the syntax of relative URLs. It is a referral, indicating that the command should be retried using one of the relative URLs.

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].

TOOMANY
This response code may be returned in a tagged OK response to a SEARCH command which includes the LIMIT modifier. The argument returns the total number of matching entries.

TOOOLD
The modtime specified in the DELETEDSINCE command is too old, so deletedsince information is no longer available.

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.

TRYFREECONTEXT
This response code may be returned in a tagged NO response to a SEARCH command which includes the MAKECONTEXT modifier. It indicates that a new context may not be created due to the server's limit on the number of existing contexts.

WAYTOOMANY
This response code may be returned in a tagged NO response to a SEARCH command which includes a HARDLIMIT search modifier. It indicates that the SEARCH would have returned more entries than the HARDLIMIT permitted.

Additional response codes MUST be registered with IANA according to the proceedures in section 7.2. Client implementations MUST tolerate response codes that they do not recognize.

4. Namespace Conventions

4.1. Dataset Namespace

The dataset namespace is a slash-separated hierarchy. The first component of the dataset namespace is a dataset class. Dataset classes MUST have a vendor prefix (vendor.<vendor/product>) or be specified in a standards track or IESG approved experimental RFC. See section 7.3 for the registration template.

The second component of the dataset name is "site", "group", "host", or "user" referring to server-wide data, administrative group data, per-host data and per-user data respectively.

For "group", "host", and "user" areas, the third component of the path is the group name, the fully qualified host domain name, or the user name. A path of the form "/<dataset-class>/~/" is a convenient abbreviation for "/<dataset-class>/user/<current-user>/".

Dataset names which begin with "/byowner/" are reserved as an alternate view of the namespace. This provides a way to see all the dataset classes which a particular owner uses. For example, "/byowner/~/<dataset-class>/" is an alternate name for "/<dataset-class>/~/". Byowner provides a way to view a list of dataset classes owned by a given user; this is done using the dataset "/byowner/user/<current-user>/" with the NOINHERIT SEARCH modifier.

The dataset "/" may be used to find all dataset classes visible to the current user. A dataset of the form "/<dataset-class>/user/" may be used to find all users which have made a dataset or entry of that class visible to the current user.

The formal syntax for a dataset name is defined by the "dataset-name" rule in section 4.3.

4.2. Attribute Namespace

Attribute names which do not contain a dot (".") are reserved for standardized attributes which have meaning in any dataset. In order to simplify client implementations, the attribute namespace is intended to be unique across all datasets. To achieve this, attribute names are prefixed with the dataset class name followed by a dot (".") . Attributes which affect management of the dataset are prefixed with "dataset.". In addition, a subtree of the "vendor." attribute namespace may be registered with IANA according to the rules in section 7.4. ACAP implementors are encouraged to help define interoperable dataset classes specifications rather than using the private attribute namespace.

Some users or sites may wish to add their own private attributes to certain dataset classes. In order to enable this, the "user.<user-name>." and "site." subtrees of the attribute namespace are reserved for user-specific and site-specific attributes respectively and will not be standardized. Such attributes are not interoperable so are discouraged in favor of defining standard attributes. A future extension is expected to permit discovery of syntax for user or site-specific attributes. Clients wishing to support display of user or site-specific attributes should display the value of any non-NIL single-valued "user.<user-name>." or "site." attribute which has valid UTF-8 syntax.

The formal syntax for an attribute name is defined by the "attribute-name" rule in the next section.

4.3. Formal Syntax for Dataset and Attribute Namespace

The naming conventions for datasets and attributes are defined by the following ABNF. Note that this grammar is not part of the ACAP protocol syntax in section 8, as dataset names and attribute names are encoded as strings within the ACAP protocol.

```
attribute-dacl  = "dataset.acl" *("." name-component)

attribute-dset  = dataset-std 1*("." name-component)
                  ;; MUST be defined in a dataset class specification

attribute-name  = attribute-std / attr-site / attr-user / vendor-name

attribute-std   = "entry" / "subdataset" / "modtime" /
                  "dataset.inherit" / attribute-dacl / attribute-dset

attr-site       = "site" 1*("." name-component)

attr-user       = "user." name-component 1*("." name-component)

byowner         = "/byowner/" owner "/"
                  [dataset-class "/" dataset-sub]

dataset-class   = dataset-std / vendor-name

dataset-normal  = "/" [dataset-class "/"
                  (owner-prefix / dataset-tail)]

dataset-name    = byowner / dataset-normal
```

```
dataset-std     = name-component
                  ;; MUST be registered with IANA and the spec MUST
                  ;; be published as a standards track or
                  ;; IESG-approved experimental RFC

dataset-sub     = *(dname-component "/")
                  ;; The rules for this portion of the namespace may
                  ;; be further restricted by the dataset class
                  ;; specification.

dataset-tail    = owner "/" dataset-sub

dname-component = 1*UTF8-CHAR
                  ;; MUST NOT begin with "." or contain "/"

name-component  = 1*UTF8-CHAR
                  ;; MUST NOT contain ".", "/", "%", or "*"

owner           = "site" / owner-host / owner-group /
                  owner-user / "~"

owner-group     = "group/" dname-component

owner-host      = "host/" dname-component

owner-prefix    = "group/" / "host/" / "user/"

owner-user      = "user/" dname-component

vendor-name     = vendor-token *("." name-component)

vendor-token    = "vendor." name-component
                  ;; MUST be registered with IANA
```

5. Dataset Management

The entry with an empty name ("") in the dataset is used to hold management information for the dataset as a whole.

5.1. Dataset Inheritance

It is possible for one dataset to inherit data from another. The dataset from which the data is inherited is called the base dataset. Data in the base dataset appears in the inheriting dataset, except when overridden by a STORE to the inheriting dataset.

The base dataset is usually a system-wide or group-wide set of defaults. A system-wide dataset usually has one inheriting dataset per user, allowing each user to add to or modify the defaults as appropriate.

An entry which exists in both the inheriting and base dataset inherits a modtime equal to the greater of the two modtimes. An attribute in such an entry is inherited from the base dataset if it was never modified by a STORE command in the inheriting dataset or if DEFAULT was stored to that attribute. This permits default entries to be amended rather than replaced in the inheriting dataset.

The "subdataset" attribute is not directly inherited. If the base dataset includes a "subdataset" attribute and the inheriting dataset does not, then the "subdataset" attribute will inherit a virtual value of a list containing a ".". The subdataset at that node is said to be a "virtual" dataset as it is simply a virtual copy of the appropriate base dataset with all "subdataset" attributes changed to a list containing a ".". A virtual dataset is not visible if NOINHERIT is specified on the SEARCH command.

Servers MUST support at least two levels of inheritance. This permits a user's dataset such as "/options/user/fred/common" to inherit from a group dataset such as "/options/group/dinosaur operators/common" which in turn inherits from a server-wide dataset such as "/options/site/common".

5.2. Dataset Attributes

The following attributes apply to management of the dataset when stored in the "" entry of a dataset. These attributes are not inherited.

dataset.acl
: This holds the default access control list for the dataset. This attribute is validated, so an invalid access control list in a STORE command will result in a NO response with an INVALID response code.

dataset.acl.<attribute>
: This holds the default access control list for an attribute within the dataset. This attribute is validated, so an invalid access control list in a STORE command will result in a NO response with an INVALID response code.

dataset.inherit
: This holds the name of a dataset from which to inherit according to the rules in the previous section. This attribute MAY refer

to a non-existent dataset, in which case nothing is inherited. This attribute is validated, so illegal dataset syntax or an attempt to store a multi-value will result in a NO response with an INVALID response code.

5.3. Dataset Creation

When a dataset is first created (by storing a "." in the subdataset attribute or storing an entry in a previously non-existent dataset), the dataset attributes are initialized with the values from the parent dataset in the "/byowner/" hierarchy. In the case of the "dataset.inherit" attribute, the appropriate hierarchy component is added. For example, given the following entry (note that \t refers to the US-ASCII horizontal tab character):

```
entry path        "/byowner/user/joe/"
dataset.acl       ("joe\txrwia" "fred\txr")
dataset.inherit   "/byowner/site"
```

If a new dataset class "/byowner/user/joe/new" is created, it will have the following dataset attributes:

```
entry path        "/byowner/user/joe/new/"
dataset.acl       ("joe\txrwia" "fred\txr")
dataset.inherit   "/byowner/site/new"
```

Note that the dataset "/byowner/user/joe/new/" is equivalent to "/new/user/joe/".

5.4. Dataset Class Capabilities

Certain dataset classes or dataset class features may only be useful if there is an active updating client or integrated server support for the feature. The dataset class "capability" is reserved to allow clients or servers to advertise such features. The "entry" attribute within this dataset class is the name of the dataset class whose features are being described. The attributes are prefixed with "capability.<dataset-class>." and are defined by the appropriate dataset class specification.

Since it is possible for an unprivileged user to run an active client for himself, a per-user capability dataset is useful. The dataset "/capability/~/" holds information about all features available to the user (via inheritance), and the dataset "/capability/site/" holds information about all features supported by the site.

5.5. Dataset Quotas

Management and scope of quotas is implementation dependent. Clients can check the applicable quota limit and usage (in bytes) with the GETQUOTA command. Servers can notify the client of a low quota situation with the QUOTA untagged response.

6. Command and Response Specifications

ACAP commands and responses are described in this section. Commands are organized first by the state in which the command is permitted, then by a general category of command type.

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 data to be returned; these are identified by "Data:" 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 data 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. Initial Connection

Upon session startup, the server sends one of two untagged responses: ACAP or BYE. The untagged BYE response is described in section 6.2.8.

6.1.1. ACAP Untagged Response

Data: capability list

The untagged ACAP response indicates the session is ready to accept commands and contains a space-separated listing of capabilities that the server supports. Each capability is represented by a list containing the capability name optionally followed by capability specific string arguments.

ACAP capability names MUST be registered with IANA according to the rules in section 7.1.

Client implementations SHOULD NOT require any capability name beyond those defined in this specification, and MUST tolerate any unknown capability names. A client implementation MAY be configurable to require SASL mechanisms other than CRAM-MD5 [CRAM-MD5] for site security policy reasons.

The following initial capabilities are defined:

CONTEXTLIMIT
: The CONTEXTLIMIT capability has one argument which is a number describing the maximum number of contexts the server supports per connection. The number 0 indicates the server has no limit, otherwise this number MUST be greater than 100.

IMPLEMENTATION
: The IMPLEMENTATION capability has one argument which is a string describing the server implementation. ACAP 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 section 6.3.1.

```
Example:    S: * ACAP (IMPLEMENTATION "ACME v3.5")
                      (SASL "CRAM-MD5") (CONTEXTLIMIT "200")
```

6.2. Any State

The following commands and responses are valid in any state.

6.2.1. 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 auto-logout timer on the server.

```
Example:    C: a002 NOOP
```

S: a002 OK "NOOP completed"

6.2.2. 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 and appropriate comparators for use with the languages listed in the LANG command.

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" "i;octet" "i;ascii-numeric"
"i;ascii-casemap" "en;primary" "fr;primary"
S: A003 OK "Bonjour"

6.2.3. LANG Intermediate Response

Data: language for error responses
appropriate comparators

The LANG response indicates the language which will be used for error responses and the comparators which are appropriate for the languages listed in the LANG command. The comparators SHOULD be in approximate order from most efficient (usually "i;octet") to most appropriate for human text in the preferred language.

6.2.4. 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 "ACAP Server logging out"
S: A023 OK "LOGOUT completed"
(Server and client then close the connection)

6.2.5. 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 "Master ACAP server is back up"

6.2.6. 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: A010 SEARCH "/addressbook/" DEPTH 3 RETURN ("*")
EQUAL "entry" "+i;octet" "bozo"
S: * NO "Master ACAP server is down, your data may

```
                be out of date."
            S: A010 OK "search done"
               ...
            C: A222 STORE ("/folder/site/comp.mail.misc"
                           "folder.creation-time" "19951206103412")
            S: A222 NO (PERMISSION ("/folder/site/")) "Permission
            denied"
```

6.2.7. 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"
```

6.2.8. 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 is also used by some server implementations as an announcement of an inactivity auto-logout.

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

Example:
```
S: * BYE "Auto-logout; idle for too long"
```

6.2.9. 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 ACAP server will be shut down in
10 minutes for system maintenance."

6.3. 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 by supporting the SASL ANONYMOUS [SASL-ANON] mechanism.

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

Only the any-state commands (NOOP, LANG and LOGOUT) and the AUTHENTICATE command are valid in non-authenticated state.

6.3.1. 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 "acap".

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

6.4. Searching

This section describes the SEARCH command, for retrieving data from datasets.

6.4.1. SEARCH Command

```
Arguments:  dataset or context name
            optional list of modifiers
            search criteria

Data:       intermediate responses: ENTRY, MODTIME, REFER
            untagged responses: ADDTO, REMOVEFROM, CHANGE, MODTIME

Result:     OK - search completed
            NO - search failure: can't perform search
            BAD - command unknown or arguments invalid
```

The SEARCH command identifies a subset of entries in a dataset and returns information on that subset to the client. Inherited entries and attributes are included in the search unless the NOINHERIT search modifier is included or the user does not have permission to read the attributes in the base dataset.

The first argument to SEARCH identifies what is to be searched. If the string begins with a slash ("/"), it is the name of a dataset to be searched, otherwise it is a name of a context that was created by a SEARCH command given previously in the session.

A successful SEARCH command MAY result in intermediate ENTRY responses and MUST result in a MODTIME intermediate response.

Following that are zero or more modifiers to the search. Each modifier may be specified at most once. The defined modifiers are:

DEPTH number
: The SEARCH command will traverse the dataset tree up to the specified depth. ENTRY responses will include the full path to the entry. A value of "0" indicates that the search should traverse the entire tree. A value of "1" is the default and indicates only the specified dataset should be searched. If a dataset is traversed which is not located on the current server, then a REFER intermediate response is returned for that subtree and the search continues.

HARDLIMIT number
: If the SEARCH command would result in more than number entries, the SEARCH fails with a NO completion result with a WAYTOOMANY response code.

LIMIT number number
: Limits the number of intermediate ENTRY responses that the search may generate. The first numeric argument specifies the limit, the second number specifies the number of entries to return if the number of matches exceeds the limit. If the limit is exceeded, the SEARCH command still succeeds, returning the total number of matches in a TOOMANY response code in the tagged OK response.

MAKECONTEXT [ENUMERATE] [NOTIFY] context
: Causes the SEARCH command to create a context with the name given in the argument to refer to the matching entries. If the SEARCH is successful, the context name may then be given as an argument to subsequent SEARCH commands to search the set of matching entries. If a context with the specified name already exists, it is first freed. If a new context may not be created due to the server’s limit on the number of existing contexts, the command fails, returning a TRYFREECONTEXT response code in the NO completion response.

  The optional "ENUMERATE" and "NOTIFY" arguments may be included to request enumeration of the context (for virtual scroll bars) or change notifications for the context. If "NOTIFY" is not requested, the context represents a snapshot of the entries at the time the SEARCH was issued.

  ENUMERATE requests that the contents of the context be ordered according to the SORT modifier and that sequential numbers, starting with one, be assigned to the entries in the context. This permits the RANGE modifier to be used to fetch portions of the ordered context.

NOTIFY requests that the server send untagged ADDTO, REMOVEFROM, CHANGE, and MODTIME responses while the context created by this SEARCH command exists. The server MAY issue untagged ADDTO, REMOVEFROM, CHANGE and MODTIME notifications for a context at any time between the issuing of the SEARCH command with MAKECONTEXT NOTIFY and the completion of a FREECONTEXT command for the context. Notifications are only issued for changes which occur after the server receives the SEARCH command which created the context. After issuing a sequence of ADDTO, REMOVEFROM or CHANGE notifications, the server MUST issue an untagged MODTIME notification indicating that the client has all updates to the entries in the context up to and including the given modtime value. Servers are permitted a reasonable delay to batch change notifications before sending them to the client.

The position arguments of the ADDTO, REMOVEFROM and CHANGE notifications are 0 if ENUMERATE is not requested.

NOINHERIT
: This causes the SEARCH command to operate without inheritance. It can be used to tell which values are explicit overrides. If MAKECONTEXT is also specified, the created context is also not affected by inheritance.

RETURN (metadata...)
: Specifies what is to be returned in intermediate ENTRY responses. If this modifier is not specified, no intermediate ENTRY responses are returned.

Inside the parentheses is an optional list of attributes, each optionally followed by a parenthesized list of metadata. If the parenthesized list of metadata is not specified, it defaults to "(value)".

An attribute name with a trailing "*" requests all attributes with that prefix. A "*" by itself requests all attributes. If the parenthesized list of metadata is not specified for an attribute with a trailing "*", it defaults to "(attribute value)". Results matching such an attribute pattern are grouped in parentheses.

Following the last intermediate ENTRY response, the server returns a single intermediate MODTIME response.

SORT (attribute comparator ...)
Specifies the order in which any resulting ENTRY replies are to be returned to the client. The SORT modifier takes as an argument a parenthesized list of one or more attribute/comparator pairs. Attribute lists the attribute to sort on, comparator specifies the name of the collation rule to apply to the values of the attribute. Successive attribute/comparator pairs are used to order two entries only when all preceding pairs indicate the two entries collate the same.

If the SORT modifier is used in conjunction with the MAKECONTEXT modifier, the SORT modifier specifies the ordering of entries in the created context.

If no SORT modifier is specified, or none of the attribute/comparator pairs indicates an order for the two entries, the server uses the order of the entries that exists in the context or dataset being searched.

Following the modifiers is the search criteria. Searching criteria consist of one or more search keys. Search keys may be combined using the AND, and OR search keys. For example, the criteria (the newline is for readability and not part of the criteria):

```
    AND COMPARE "modtime" "+i;octet" "19951206103400"
        COMPARE "modtime" "-i;octet" "19960112000000"
```

refers to all entries modified between 10:34 December 6 1995 and midnight January 12, 1996 UTC.

The currently defined search keys are as follows.

ALL This matches all entries.

AND search-key1 search-key2
Entries that match both search keys.

COMPARE attribute comparator value
Entries for which the value of the specified attribute collates using the specified comparator the same or later than the specified value.

COMPARESTRICT attribute comparator value
Entries for which the specified attribute collates using the specified comparator later than the specified value.

EQUAL attribute comparator value
Entries for which the value of the attribute is equal to the specified value using the specified comparator.

NOT search-key
Entries that do not match the specified search key.

OR search-key1 search-key2
Entries that match either search key.

PREFIX attribute comparator value
Entries which begin with the specified value using the specified comparator. If the specified comparator doesn't support substring matching, a BAD response is returned.

RANGE start end time
Entries which are within the specified range of the enumerated context's ordering. The lowest-ordered entry in the context is assigned number one, the next lowest entry is assigned number two, and so on. The numeric arguments specify the lowest and highest numbers to match. The time specifies that the client has processed notifications for the context up to the specified time. If the context has been modified since then, the server MUST either return a NO with a MODIFIED response code, or return the results that the SEARCH would have returned if none of the changes since that time had been made.

RANGE is only permitted on enumerated contexts. If RANGE is used with a dataset or non-enumerated context, the server MUST return a BAD response.

SUBSTRING attribute comparator value
Entries which contain the specified value, using the specified comparator. If the specified comparator doesn't support substring matching, a BAD response is returned.

6.4.2. ENTRY Intermediate Response

Data: entry name
entry data

The ENTRY intermediate response occurs as a result of a SEARCH or STORE command. This is the means by which dataset entries are returned to the client.

The ENTRY response begins with the entry name, if a SEARCH command without the DEPTH modifier was issued, or the entry path in other cases. This is followed by a set of zero or more items, one for each metadata item in the RETURN search modifier. Results matching an attribute pattern or returning multiple metadata items are grouped in parentheses.

6.4.3. MODTIME Intermediate Response

Data: modtime value

The MODTIME intermediate response occurs as a result of a SEARCH command. It indicates that the just created context or the previously returned ENTRY responses include all updates to the returned entries up to and including the modtime value in the argument.

6.4.4. REFER Intermediate Response

Data: dataset path
relative ACAP URLs

The REFER intermediate response occurs as a result of a multi-level SEARCH where one of the levels is located on a different server. The response indicates the dataset which is not located on the current server and one or more relative ACAP URLs for where that dataset may be found.

6.4.5. Search Examples

Here are some SEARCH command exchanges between the client and server:

```
C: A046 SEARCH "/addressbook/" DEPTH 3 RETURN ("addressbook.Alias"
        "addressbook.Email" "addressbook.List") OR NOT EQUAL
        "addressbook.Email" "i;octet" NIL NOT EQUAL
        "addressbook.List" "i;octet" NIL
S: A046 ENTRY "/addressbook/user/joe/A0345" "fred"
        "fred@stone.org" NIL
S: A046 ENTRY "/addressbook/user/fred/A0537" "joe" "joe@stone.org"
        NIL
S: A046 ENTRY "/addressbook/group/Dinosaur Operators/A423"
        "saurians" NIL "1"
S: A046 MODTIME "19970728105252"
S: A046 OK "SEARCH completed"

C: A047 SEARCH "/addressbook/user/fred/" RETURN ("*") EQUAL "entry"
        "i;octet" "A0345"
S: A047 ENTRY "A0345" (("modtime" "19970728102226")
```

```
          ("addressbook.Alias" "fred") ("addressbook.Email"
          "fred@stone.org") ("addressbook.CommonName"
          "Fred Flintstone") ("addressbook.Surname" "Flintstone")
          ("addressbook.GivenName" "Fred"))
   S: A047 MODTIME "19970728105258"
   S: A047 OK "SEARCH completed"

   C: A048 SEARCH "/options/~/vendor.example/" RETURN
          ("option.value"("size" "value" "myrights"))
          SORT ("entry" "i;octet") COMPARE "modtime" "i;octet"
          "19970727123225"
   S: A048 ENTRY "blurdybloop" (5 "ghoti" "rwia")
   S: A048 ENTRY "buckybits" (2 "10" "rwia")
   S: A048 ENTRY "windowSize" (7 "100x100" "rwia")
   S: A048 MODTIME "19970728105304"
   S: A048 OK "SEARCH completed"

   C: A049 SEARCH "/addressbook/~/public" RETURN ("addressbook.Alias"
          "addressbook.Email") MAKECONTEXT ENUMERATE "blob" LIMIT 100 1
          SORT ("addressbook.Alias" "i;octet") NOT EQUAL
          "addressbook.Email" NIL
   S: A049 ENTRY "A437" "aaguy" "aaguy@stone.org"
   S: A049 MODTIME "19970728105308"
   S: A049 OK (TOOMANY 347) "Context 'blob' created"

   C: A050 SEARCH "blob" RANGE 2 2 "19970728105308" ALL
   S: A050 ENTRY "A238" "abguy" "abguy@stone.org"
   S: A050 MODTIME "19970728105310"
   S: A050 OK "SEARCH Completed"
```

6.5. Contexts

The following commands use contexts created by a SEARCH command with a MAKECONTEXT modifier.

6.5.1. FREECONTEXT Command

```
Arguments:  context name

Data:       no specific data for this command

Result:     OK - freecontext completed
            NO - freecontext failure: no such context
            BAD - command unknown or arguments invalid
```

The FREECONTEXT command causes the server to free all state associated with the named context. The context may no longer be searched and the server will no longer issue any untagged responses for the context. The context is no longer counted against the server's limit on the number of contexts.

Example: C: A683 FREECONTEXT "blurdybloop"
S: A683 OK "Freecontext completed"

6.5.2. UPDATECONTEXT Command

Arguments: list of context names

Data: untagged responses: ADDTO REMOVEFROM CHANGE MODTIME

Result: OK - Updatecontext completed: all updates completed
NO - Updatecontext failed: no such context
not a notify context
BAD - command unknown or arguments invalid

The UPDATECONTEXT command causes the server to ensure that the client is notified of all changes known to the server for the contexts listed as arguments up to the current time. The contexts listed in the arguments must have been previously given to a successful SEARCH command with a MAKECONTEXT NOTIFY modifier. A MODTIME untagged response MUST be returned if any read-write metadata in the context changed since the last MODTIME for that context. This includes metadata which is not listed in the RETURN modifier for the context.

While a server may issue untagged ADDTO, REMOVEFROM, CHANGE, and MODTIME at any time, the UPDATECONTEXT command is used to "prod" the server to send any notifications it has not sent yet.

The UPDATECONTEXT command SHOULD NOT be used to poll for updates.

Example: C: Z4S9 UPDATECONTEXT "blurdybloop" "blarfl"
S: Z4S9 OK "client has been notified of all changes"

6.5.3. ADDTO Untagged Response

Data: context name
entry name
position
metadata list

The untagged ADDTO response informs the client that an entry has been added to a context. The response includes the position number of the added entry (the first entry in the context is numbered 1) and those metadata contained in the entry which match the RETURN statement when the context was created.

For enumerated contexts, the ADDTO response implicitly adds one to the position of all members of the context which had position numbers that were greater than or equal to the ADDTO position number. For non-enumerated contexts, the position field is always 0.

Example: S: * ADDTO "blurdybloop" "fred" 15
("addressbook.Email" "fred@stone.org")

6.5.4. REMOVEFROM Untagged Response

Data: context name
entry name
old position

The untagged REMOVEFROM response informs the client that an entry has been removed from a context. The response includes the position number that the removed entry used to have (the first entry in the context is numbered 1).

For enumerated contexts, the REMOVEFROM response implicitly subtracts one from the position numbers of all members of the context which had position numbers greater than the REMOVEFROM position number. For non-enumerated contexts, the position field is always 0.

Example: S: * REMOVEFROM "blurdybloop" "fred" 15

6.5.5. CHANGE Untagged Response

Data: context name
entry name
old position
new position
metadata list

The untagged CHANGE response informs the client that an entry in a context has either changed position in the context or has changed the values of one or more of the attributes specified in the RETURN modifier when the context was created.

The response includes the previous and current position numbers of the entry (which are 0 if ENUMERATE was not specified on the context) and the attribute metadata requested in the RETURN modifier when the context was created.

For enumerated contexts, the CHANGE response implicitly changes the position numbers of all entries which had position numbers between the old and new position. If old position is less than new position, than one is subtracted from all entries which had position numbers in that range. Otherwise one is added to all entries which had position numbers in that range. If the old position and new position are the same, then no implicit position renumbering occurs.

CHANGE responses are not issued for entries which have changed position implicitly due to another ADDTO, REMOVEFROM or CHANGE response.

Example: S: * CHANGE "blurdybloop" "fred" 15 10
("addressbook.Email" "fred@stone.org")

6.5.6. MODTIME Untagged Response

Data: context name
modtime value

The untagged MODTIME response informs the client that it has received all updates to entries in the context which have modtime values less than or equal to the modtime value in the argument.

Example: S: * MODTIME mycontext "19970320162338"

6.6. Dataset modification

The following commands and responses handle modification of datasets.

6.6.1. STORE Command

Arguments: entry store list

Data: intermediate responses: ENTRY

Result: OK - store completed
NO - store failure: can't store that name
UNCHANGEDSINCE specified and entry changed
BAD - command unknown or arguments invalid
invalid UTF-8 syntax in attribute name

Creates, modifies, or deletes the named entries in the named datasets. The values of metadata not specified in the command are not changed. Setting the "value" metadata of an attribute to NIL removes that attribute from the entry. Setting the "value" of the "entry" attribute to NIL removes that entry from the dataset and cancels inheritance for the entire entry. Setting the "value" of the "entry" attribute to DEFAULT removes that entry from the inheriting dataset and reverts the entry and its attributes to inherited values, if any. Changing the value of the "entry" attribute renames the entry.

Storing DEFAULT to the "value" metadata of an attribute is equivalent to storing NIL, except that inheritance is enabled for that attribute. If a non-NIL value is inherited then an ENTRY intermediate response is generated to notify the client of the this change. The ENTRY response includes the entry-path and the attribute name and value metadata for each attribute which reverted to a non-NIL inherited setting.

Storing NIL to the "value" metadata of an attribute MAY be treated equivalent to storing DEFAULT to that "value" if there is a NIL value in the base dataset.

The STORE command is followed by one or more entry store lists. Each entry store list begins with an entry path followed by STORE modifiers, followed by zero or more attribute store items. Each attribute store item is made up of the attribute name followed by NIL (to remove the attribute's value), DEFAULT (to revert the item to any inherited value), a single value (to set the attribute's single value), or a list of metadata items to modify. The following STORE modifiers may be specified:

NOCREATE
By default, the server MUST create any datasets necessary to store the entry, including multiple hierarchy levels. If NOCREATE is specified, the STORE command will fail with a NOEXIST error unless the parent dataset already exists.

UNCHANGEDSINCE
If the "modtime" of the entry is later than the unchangedsince time, then the store fails with a MODIFIED response code. Use of UNCHANGEDSINCE with a time of "00000101000000" will always fail if the entry exists. Clients writing to a shared dataset are encouraged to use UNCHANGEDSINCE when modifying an existing entry.

The server MUST either make all the changes specified in a single STORE command or make none of them. If successful, the server MUST update the "modtime" attribute for every entry which was changed.

It is illegal to list any metadata item within an attribute twice, any attribute within an entry twice or any entry path twice. The server MUST return a BAD response if this happens.

The server MAY re-order the strings in a multi-value on STORE and MAY remove duplicate strings. However, SEARCH MUST return multi-values and the associated size list metadata in a consistant order.

```
Example:    C: A342 STORE ("/addressbook/user/fred/ABC547"
                    "addressbook.TelephoneNumber" "555-1234"
                    "addressbook.CommonName" "Barney Rubble"
                    "addressbook.AlternateNames" ("value"
                    ("Barnacus Rubble" "Coco Puffs Thief"))
                    "addressbook.Email" NIL)
            S: A342 OK "Store completed"
            C: A343 STORE ("/addressbook/user/joe/ABD42"
                    UNCHANGEDSINCE "19970320162338"
                    "user.joe.hair-length" "10 inches")
            S: A343 NO (MODIFIED) "'ABD42' has been changed
                    by somebody else."
            C: A344 STORE ("/addressbook/group/Developers/ACD54"
                    "entry" NIL)
            S: A344 OK "Store completed"
            C: A345 STORE ("/option/~/common/SMTPserver"
                    "option.value" DEFAULT)
            S: A345 ENTRY "/option/~/common/SMTPserver"
```

```
                "option.value" "smtp.server.do.main"
            S: A345 OK "Store completed"
            C: A347 STORE ("/addressbook/~/" "dataset.inherit"
                "/addressbook/group/Developers")
            S: A347 OK "Store completed"
```

6.6.2. DELETEDSINCE Command

Arguments: dataset name
time

Data: intermediate response: DELETED

Result: OK - DELETEDSINCE completed
NO - DELETEDSINCE failure: can't read dataset
date too far in the past
BAD - command unknown or arguments invalid

The DELETEDSINCE command returns in intermediate DELETED replies the names of entries that have been deleted from the named dataset since the given time.

Servers may impose a limit on the number or age of deleted entry names they keep track of. If the server does not have information going back to the specified time, the command fails, returning a TOOOLD response code in the tagged NO response.

Example:
```
C: Z4S9 DELETEDSINCE "/folder/site/" 19951205103412
S: Z4S9 DELETED "blurdybloop"
S: Z4S9 DELETED "anteaters"
S: Z4S9 OK "DELETEDSINCE completed"
C: Z4U3 DELETEDSINCE "/folder/site/" 19951009040854
S: Z4U3 NO (TOOOLD) "Don't have that information"
```

6.6.3. DELETED Intermediate Response

Data: entry name

The intermediate DELETED response occurs as a result of a DELETEDSINCE command. It returns an entry that has been deleted from the dataset specified in the DELETEDSINCE command.

6.7. Access Control List Commands

The commands in this section are used to manage access control lists.

6.7.1. SETACL Command

Arguments: acl object
authentication identifier
access rights

Data: no specific data for this command

Result: OK - setacl completed
NO - setacl failure: can't set acl
BAD - command unknown or arguments invalid

The SETACL command changes the access control list on the specified object so that the specified identifier is granted the permissions enumerated in rights. If the object did not previously have an access control list, one is created.

Example: C: A123 SETACL ("/addressbook/~/public/") "anyone" "r"
S: A123 OK "Setacl complete"
C: A124 SETACL ("/folder/site/") "B1FF" "rwa"
S: A124 NO (PERMISSION ("/folder/site/")) "'B1FF' not permitted to modify access rights for '/folder/site/'"

6.7.2. DELETEACL Command

Arguments: acl object
optional authentication identifier

Data: no specific data for this command

Result: OK - deleteacl completed
NO - deleteacl failure: can't delete acl
BAD - command unknown or arguments invalid

If given the optional identifier argument, the DELETEACL command removes any portion of the access control list on the specified object for the specified identifier.

If not given the optional identifier argument, the DELETEACL command removes the ACL from the object entirely, causing access to be controlled by a higher-level default ACL. This form of the DELETEACL command is not permitted on the default ACL for a dataset and servers MUST return a BAD.

```
Example:    C: A223 DELETEACL ("/addressbook/~/public") "anyone"
            S: A223 OK "Deleteacl complete"
            C: A224 DELETEACL ("/folder/site")
            S: A224 BAD "Can't delete ACL from dataset"
            C: A225 DELETEACL ("/addressbook/user/fred"
                  "addressbook.Email" "barney")
            S: A225 OK "Deleteacl complete"
```

6.7.3. MYRIGHTS Command

```
Arguments:  acl object

Data:       intermediate responses: MYRIGHTS

Result:     OK - myrights completed
            NO - myrights failure: can't get rights
            BAD - command unknown or arguments invalid
```

The MYRIGHTS command returns the set of rights that the client has to the given dataset or dataset attribute.

```
Example:    C: A003 MYRIGHTS ("/folder/site")
            S: A003 MYRIGHTS "r"
            S: A003 OK "Myrights complete"
```

6.7.4. MYRIGHTS Intermediate Response

```
Data:       rights
```

The MYRIGHTS response occurs as a result of a MYRIGHTS command. The argument is the set of rights that the client has for the object referred to in the MYRIGHTS command.

6.7.5. LISTRIGHTS Command

```
Arguments:  acl object
            authentication identifier

Data:       untagged responses: LISTRIGHTS

Result:     OK - listrights completed
            NO - listrights failure: can't get rights list
            BAD - command unknown or arguments invalid
```

The LISTRIGHTS command takes an object and an identifier and returns information about what rights the current user may revoke or grant to that identifier in the ACL for that object.

```
Example:    C: a001 LISTRIGHTS ("/folder/~/") "smith"
            S: a001 LISTRIGHTS "xra" "w" "i"
            S: a001 OK Listrights completed
            C: a005 LISTRIGHTS ("/folder/site/archive/imap") "anyone"
            S: a005 LISTRIGHTS "" "x" "r" "w" "i"
            S: a005 OK Listrights completed
```

6.7.6. LISTRIGHTS Intermediate Response

```
Data:       required rights
            list of optional rights
```

The LISTRIGHTS response occurs as a result of a LISTRIGHTS command. The first argument is a string containing the (possibly empty) set of rights the identifier will always be granted on the dataset or attribute.

Following this are zero or more strings each containing a single right which the current user may revoke or grant to the identifier in the dataset or attribute.

The same right MUST NOT be listed more than once in the LISTRIGHTS response.

6.8. Quotas

The section defines the commands and responses relating to quotas.

6.8.1. GETQUOTA Command

```
Arguments:  dataset

Data:       untagged responses: QUOTA

Result:     OK - Quota information returned
            NO - Quota failure: can't access resource limit
                                no resource limit
            BAD - command unknown or arguments invalid
```

The GETQUOTA command takes the name of a dataset, and returns in an untagged QUOTA response the name of the dataset, quota limit in bytes that applies to that dataset and the quota usage within that limit. The scope of a quota limit is implementation dependent.

```
Example:    C: A043 GETQUOTA "/option/user/fred/common"
            S: * QUOTA "/option/user/fred/common" 1048576 2475
            S: A043 OK "Getquota completed"
```

6.8.3. QUOTA Untagged Response

```
Data:       dataset
            quota limit in bytes
            amount of quota limit used
            extension data
```

The QUOTA untagged response is generated as a result of a GETQUOTA command or MAY be generated by the server in response to a SEARCH or STORE command to warn about high usage of a quota. It includes the name of the applicable dataset, the quota limit in bytes, the quota usage and some optional extension data. Clients MUST tolerate the extension data as its use is reserved for a future extension.

6.9. Extensions

In order to simplify the process of extending the protocol, clients MUST tolerate unknown server responses which meet the syntax of response-extend. In addition, clients MUST tolerate unknown server response codes which meet the syntax of resp-code-ext. Availability of new commands MUST be announced via a capability on the initial greeting line and such commands SHOULD meet the syntax of command-extend.

Servers MUST respond to unknown commands with a BAD command completion result. Servers MUST skip over non-synchronizing literals contained in an unknown command. This may be done by assuming the unknown command matches the command-extend syntax, or by reading a line at a time and checking for the non-synchronizing literal syntax at the end of the line.

7. Registration Procedures

ACAP's usefulness comes from providing a structured storage model for all sorts of configuration data. However, for its potential to be achieved, it is important that the Internet community strives for the following goals:

(1) Standardization. It is very important to standardize dataset classes. The authors hope that ACAP achieves the success that SNMP has seen with the definition of numerous standards track MIBs.

(2) Community Review. In the absence of standardization, it is important to get community review on a proposal to improve its engineering quality. Community review is strongly recommended prior to registration. The ACAP implementors mailing list <ietf-acap@andrew.cmu.edu> should be used for this purpose.

(3) Registration. Registration serves a two-fold purpose. First it prevents use of the same name for different purposes, and second it provides a one-stop list which can be used to locate existing extensions or dataset classes to prevent duplicate work.

The following registration templates may be used to register ACAP protocol elements with the Internet Assigned Numbers Authority (IANA).

7.1. ACAP Capabilities

New ACAP capabilities MUST be registered prior to use. Careful consideration should be made before extending the protocol, as it can lead to complexity or interoperability problems. Review of proposals on the acap implementors mailing list is strongly encouraged prior to registration.

To: iana@iana.org
Subject: Registration of ACAP capability

Capability name:

Capability keyword:

Capability arguments:

Published Specification(s):

(Optional, but strongly encouraged)

Person and email address to contact for further information:

7.2. ACAP Response Codes

ACAP response codes are registered on a first come, first served basis. Review of proposals on the acap implementors mailing list is strongly encouraged prior to registration.

To: iana@iana.org
Subject: Registration of ACAP response code

Response Code:

Arguments (use ABNF to specify syntax):

Purpose:

Published Specification(s):

(Optional, but strongly encouraged)

Person and email address to contact for further information:

7.3. Dataset Classes

A dataset class provides a core set of attributes for use in a specified hierarchy. It may also define rules for the dataset hierarchy underneath that class. Dataset class specifications must be standards track or IESG approved experimental RFCs.

To: iana@iana.org
Subject: Registration of ACAP dataset class

Dataset class name/attribute prefix:

Purpose:

Published Specification(s):

(Standards track or IESG approved experimental RFC)

Person and email address to contact for further information:

7.4. Vendor Subtree

Vendors may reserve a portion of the ACAP namespace for private use. Dataset class names beginning with "vendor.<company/product name>." are reserved for use by that company or product. In addition, all attribute names beginning with "vendor.<company/product name>." are reserved for use by that company or product once registered. Registration is on a first come, first served basis. Whenever possible, private attributes and dataset classes should be avoided in favor of improving interoperable dataset class definitions.

To: iana@iana.org
Subject: Registration of ACAP vendor subtree

Private Prefix: vendor.<company/product name>.

Person and email address to contact for further information:

(company names and addresses should be included when appropriate)

8. 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.

The "initial-greeting" rule below defines the initial ACAP greeting from the server. The "command" rule below defines the syntax for commands sent by the client. The "response" rule below defines the syntax for responses sent by the server.

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

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

CHAR               = %x01-7F

DIGIT-NZ           = %x31-39
                       ; non-zero digits ("1" - "9")

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

QUOTED-SPECIALS    = <"> / "\"

SAFE-CHAR          = %x01-09 / %x0B-0C / %x0E-21 /
                       %x23-5B / %x5D-7F
                       ;; any TEXT-CHAR except 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-CHAR          = %x01-09 / %x0B-0C / %x0E-7F
                        ;; any CHAR except CR and LF

   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

   acl                = "(" [acl-identrights *(SP acl-identrights)] ")"
                        *(SPACE acl-identrights)] ")"

   acl-identifier     = string-utf8
                        ;; MUST NOT contain HTAB

   acl-identrights    = string-utf8
                        ;; The identifier followed by a HTAB,
                        ;; followed by the rights.

   acl-delobject      = "(" dataset SP attribute [SP entry-name] ")"

   acl-object         = "(" dataset [SP attribute [SP entry-name]] ")"

   acl-rights         = quoted

   atom               = ALPHA *1023ATOM-CHAR

   attribute          = string-utf8
                        ;; dot-separated attribute name
                        ;; MUST NOT contain "*" or "%"

   attribute-store    = attribute SP (value-nildef /
                        "(" 1*(metadata-write-q SP value-store) ")")
                        ;; MUST NOT include the same metadata twice

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

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

   command            = tag SP (command-any / command-auth /
                        command-nonauth) CRLF
                        ;; Modal based on state

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

   command-any        = "NOOP" / command-lang / "LOGOUT" /
                        command-extend

   command-auth       = command-delacl / command-dsince /
                        command-freectx / command-getquota /
                        command-lrights / command-myrights /
                        command-search / command-setacl /
                        command-store
                        ;; only valid in authenticated state

   command-delacl     = "DELETEACL" SP acl-delobject [SP acl-identifier]

   command-dsince     = "DELETEDSINCE" SP dataset SP time

   command-extend     = extend-token [SP extension-data]

   command-freectx    = "FREECONTEXT" SP context

   command-getquota   = "GETQUOTA" SP dataset

   command-lang       = "LANG" *(SP lang-tag)

   command-lrights    = "LISTRIGHTS" SP acl-object

   command-myrights   = "MYRIGHTS" SP acl-object

   command-nonauth    = command-authent
                        ;; only valid in non-authenticated state

   command-search     = "SEARCH" SP (dataset / context)
                        *(SP search-modifier) SP search-criteria
                        ;; MUST NOT include same search-modifier twice

   command-setacl     = "SETACL" SP acl-object SP acl-identifier
                        SP acl-rights

   command-store      = "STORE" SP store-entry-list
```

```
   comparator          = <"> comparator-name <">

   comparator-name     = ["+" / "-"] iana-token

   context             = string-utf8
                         ;; MUST NOT begin with slash ("/")

   dataset             = string-utf8
                         ;; slash-separated dataset name
                         ;; begins with slash

   entry               = entry-name / entry-path

   entry-name          = string-utf8
                         ;; entry name MUST NOT contain slash
                         ;; MUST NOT begin with "."

   entry-path          = string-utf8
                         ;; slash-separated path to entry
                         ;; begins with slash

   entry-relative      = string-utf8
                         ;; potentially relative path to entry

   extend-token        = atom
                         ;; MUST be defined by a standards track or
                         ;; IESG approved experimental protocol extension

   extension-data      = extension-item *(SP extension-item)

   extension-item      = extend-token / string / number /
                         "(" [extension-data] ")"

   iana-token          = atom
                         ;; MUST be registered with IANA

   initial-greeting    = "*" SP "ACAP" *(SP "(" init-capability ")") CRLF

   init-capability     = init-cap-context / init-cap-extend /
                         init-cap-implem / init-cap-sasl

   init-cap-context    = "CONTEXTLIMIT" SP string

   init-cap-extend     = iana-token [SP string-list]

   init-cap-implem     = "IMPLEMENTATION" SP string

   init-cap-sasl       = "SASL" SP string-list
```

```
   lang-tag           = <"> Language-Tag <">
                        ;; Language-Tag rule is defined in [LANG-TAGS]

   literal            = "{" number [ "+" ] "}" CRLF *OCTET
                        ;; The number represents the number of octets
                        ;; MUST be literal-utf8 except for values

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

   metadata           = attribute [ "(" metadata-type-list ")" ]
                        ;; attribute MAY end in "*" as wildcard.

   metadata-list      = metadata *(SP metadata)

   metadata-type      = "attribute" / "myrights" / "size" /
                        "count" / metadata-write

   metadata-type-q    = <"> metadata-type <">

   metadata-type-list = metadata-type-q *(SP metadata-type-q)

   metadata-write     = "value" / "acl"

   metadata-write-q   = <"> metadata-write <">

   nil                = "NIL"

   number             = *DIGIT
                        ;; A 32-bit unsigned number.
                        ;; (0 <= n < 4,294,967,296)

   nz-number          = DIGIT-NZ *DIGIT
                        ;; A 32-bit unsigned non-zero number.
                        ;; (0 < n < 4,294,967,296)

   position           = number
                        ;; "0" if context is not enumerated
                        ;; otherwise this is non-zero

   quota-limit        = number

   quota-usage        = number

   quoted             = <"> *QUOTED-CHAR <">
                        ;; limited to 1024 octets between the <">s
```

```
   response           = response-addto / response-alert / response-bye /
                        response-change / response-cont /
                        response-deleted / response-done /
                        response-entry / response-extend /
                        response-listr / response-lang /
                        response-mtimei / response-mtimeu /
                        response-myright / response-quota /
                        response-refer / response-remove / response-stat

   response-addto     = "*" SP "ADDTO" SP context SP entry-name
                        SP position SP return-data-list

   response-alert     = "*" SP "ALERT" SP resp-body CRLF
                         ;; Client MUST display alert text to user

   response-bye       = "*" SP "BYE" SP resp-body CRLF
                         ;; Server will disconnect condition

   response-change    = "*" SP "CHANGE" SP context SP entry-name
                        SP position SP position SP return-data-list

   response-cont      = "+" SP string

   response-deleted   = tag SP "DELETED" SP entry-name

   response-done      = tag SP resp-cond-state CRLF

   response-entry     = tag SP "ENTRY" SP entry SP return-data-list

   response-extend    = (tag / "*") SP extend-token [SP extension-data]

   response-lang      = "*" SP "LANG" SP lang-tag 1*(SP comparator)

   response-listr     = tag SP "LISTRIGHTS" SP acl-rights
                        *(SP acl-rights)

   response-mtimei    = tag SP "MODTIME" SP time

   response-mtimeu    = "*" SP "MODTIME" SP context SP time

   response-myright   = tag SP "MYRIGHTS" SP acl-rights

   response-quota     = "*" SP "QUOTA" SP dataset SP quota-limit
                        SP quota-usage [SP extension-data]

   response-refer     = tag SP "REFER" SP dataset
                        1*(SP <"> url-relative <">)
```

```
   response-remove    = "*" SP "REMOVEFROM" SP context SP
                        entry-name SP position

   response-stat      = "*" SP resp-cond-state CRLF

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

   resp-code          = "AUTH-TOO-WEAK" / "ENCRYPT-NEEDED" /
                        resp-code-inval / resp-code-mod /
                        resp-code-noexist / resp-code-perm / "QUOTA" /
                        resp-code-refer / resp-code-sasl /
                        resp-code-toomany / "TOOOLD" /
                        "TRANSITION-NEEDED" / "TRYFREECONTEXT" /
                        "TRYLATER" / "WAYTOOMANY" / resp-code-ext

   resp-code-ext      = iana-token [SP extension-data]
                        ;; unknown codes MUST be tolerated by the client

   resp-code-inval    = "INVALID" 1*(SP entry-path SP attribute)

   resp-code-mod      = "MODIFIED" SP entry-path

   resp-code-noexist  = "NOEXIST" SP dataset

   resp-code-perm     = "PERMISSION" SP acl-object

   resp-code-refer    = "REFER" 1*(SP <"> url-relative <">)

   resp-code-sasl     = "SASL" SP string

   resp-code-toomany  = "TOOMANY" SP nz-number

   resp-cond-state    = ("OK" / "NO" / "BAD") SP resp-body
                        ;; Status condition

   return-attr-list   = "(" return-metalist *(SP return-metalist) ")"
                        ;; occurs when "*" in RETURN pattern on SEARCH

   return-data        = return-metadata / return-metalist /
                        return-attr-list

   return-data-list   = return-data *(SP return-data)

   return-metalist    = "(" return-metadata *(SP return-metadata) ")"
                        ;; occurs when multiple metadata items requested

   return-metadata    = nil / string / value-list / acl
```

```
   searchkey-equal    = "EQUAL" SP attribute SP comparator SP value-nil

   searchkey-comp     = "COMPARE" SP attribute SP comparator SP value

   searchkey-prefix   = "PREFIX" SP attribute SP comparator SP value

   searchkey-range    = "RANGE" SP nz-number SP nz-number SP time

   searchkey-strict   = "COMPARESTRICT" SP attribute SP comparator
                        SP value

   searchkey-substr   = "SUBSTRING" SP attribute SP comparator SP value

   searchmod-depth    = "DEPTH" SP number

   searchmod-hard     = "HARDLIMIT" SP nz-number

   searchmod-limit    = "LIMIT" SP number SP number

   searchmod-make     = "MAKECONTEXT" [SP "ENUMERATE"]
                        [SP "NOTIFY"] SP context

   searchmod-ninh     = "NOINHERIT"

   searchmod-return   = "RETURN" SP "(" [metadata-list] ")"

   searchmod-sort     = "SORT" SP "(" sort-list ")"

   search-criteria    = "ALL" / searchkey-equal / searchkey-comp /
                        searchkey-strict / searchkey-range /
                        searchkey-prefix / searchkey-substr /
                        "NOT" SP search-criteria /
                        "OR" SP search-criteria SP search-criteria /
                        "AND" SP search-criteria SP search-criteria

   search-modifier    = searchmod-depth / searchmod-hard /
                        searchmod-limit / searchmod-make /
                        searchmod-ninh / searchmod-return /
                        searchmod-sort

   sort-list          = sort-item *(SP sort-item)

   sort-item          = attribute SP comparator

   store-entry        = "(" entry-path *(SP store-modifier)
                            *(SP attribute-store) ")"
                        ;; MUST NOT include same store-modifier twice
                        ;; MUST NOT include same attribute twice
```

```
   store-entry-list   = store-entry *(SP store-entry)
                        ;; MUST NOT include same entry twice

   store-modifier     = store-mod-unchang / store-mod-nocreate

   store-mod-nocreate = "NOCREATE"

   store-mod-unchang  = "UNCHANGEDSINCE" SP time

   string             = quoted / literal

   string-list        = string *(SP string)

   string-utf8        = quoted / literal-utf8

   tag                = 1*32TAG-CHAR

   time               = <"> time-year time-month time-day time-hour
                        time-minute time-second time-subsecond <">
                        ;; Timestamp in UTC

   time-day           = 2DIGIT ;; 01-31

   time-hour          = 2DIGIT ;; 00-23

   time-minute        = 2DIGIT ;; 00-59

   time-month         = 2DIGIT ;; 01-12

   time-second        = 2DIGIT ;; 00-60

   time-subsecond     = *DIGIT

   time-year          = 4DIGIT

   value              = string

   value-list         = "(" [value *(SP value)] ")"

   value-nil          = value / nil

   value-nildef       = value-nil / "DEFAULT"

   value-store        = value-nildef / value-list / acl

   url-acap           = "acap://" url-server "/" url-enc-entry
                        [url-filter] [url-extension]
                        ;; url-enc-entry interpreted relative to "/"
```

```
   url-attr-list      = url-enc-attr *("&" url-enc-attr)

   url-auth           = ";AUTH=" ("*" / url-enc-auth)

   url-achar          = uchar / "&" / "=" / "~"
                        ;; See RFC 1738 for definition of "uchar"

   url-char           = uchar / "=" / "~" / ":" / "@" / "/"
                        ;; See RFC 1738 for definition of "uchar"

   url-enc-attr       = 1*url-char
                        ;; encoded version of attribute name

   url-enc-auth       = 1*url-achar
                        ;; encoded version of auth-type-name above

   url-enc-entry      = 1*url-char
                        ;; encoded version of entry-relative above

   url-enc-user       = *url-achar
                        ;; encoded version of login userid

   url-extension      = *("?" 1*url-char)

   url-filter         = "?" url-attr-list

   url-relative       = url-acap / [url-enc-entry] [url-filter]
                        ;; url-enc-entry is relative to base URL

   url-server         = [url-enc-user [url-auth] "@"] hostport
                        ;; See RFC 1738 for definition of "hostport"
```

9. Multi-lingual Considerations

The IAB charset workshop [IAB-CHARSET] came to a number of conclusions which influenced the design of ACAP. The decision to use UTF-8 as the character encoding scheme was based on that work. The LANG command to negotiate a language for error messages is also included.

Section 3.4.5 of the IAB charset workshop report states that there should be a way to identify the natural language for human readable strings. Several promising proposals have been made for use within ACAP, but no clear consensus on a single method is apparent at this stage. The following rules are likely to permit the addition of multi-lingual support in the future:

(1) A work in progress called Multi-Lingual String Format (MLSF) proposes a layer on top of UTF-8 which uses otherwise illegal UTF-8 sequences to store language tags. In order to permit its addition to a future version of this standard, client-side UTF-8 interpreters MUST be able to silently ignore illegal multi-byte UTF-8 characters, and treat illegal single-byte UTF-8 characters as end of string markers. Servers, for the time being, MUST be able to silently accept illegal UTF-8 characters, except in attribute names and entry names. Clients MUST NOT send illegal UTF-8 characters to the server unless a future standard changes this rule.

(2) There is a proposal to add language tags to Unicode. To support this, servers MUST be able to store UTF-8 characters of up to 20 bits of data.

(3) The metadata item "language" is reserved for future use.

10. Security Considerations

The AUTHENTICATE command uses SASL [SASL] to provide basic authentication, authorization, integrity and privacy services. This is described in section 6.3.1.

When the CRAM-MD5 mechanism is used, the security considerations for the CRAM-MD5 SASL mechanism [CRAM-MD5] apply. The CRAM-MD5 mechanism is also susceptible to passive dictionary attacks. This means that if an authentication session is recorded by a passive observer, that observer can try common passwords through the CRAM-MD5 mechanism and see if the results match. This attack is reduced by using hard to guess passwords. Sites are encouraged to educate users and have the password change service test candidate passwords against a dictionary. ACAP implementations of CRAM-MD5 SHOULD permit passwords of at least 64 characters in length.

ACAP protocol transactions are susceptible to passive observers or man in the middle attacks which alter the data, unless the optional encryption and integrity services of the AUTHENTICATE command are enabled, or an external security mechanism is used for protection. It may be useful to allow configuration of both clients and servers to refuse to transfer sensitive information in the absence of strong encryption.

ACAP access control lists provide fine grained authorization for access to attributes. A number of related security issues are described in section 3.5.

ACAP URLs have the same security considerations as IMAP URLs [IMAP-URL].

ACAP clients are encouraged to consider the security problems involved with a lab computer situation. Specifically, a client cache of ACAP configuration information MUST NOT allow access by an unauthorized user. One way to assure this is for an ACAP client to be able to completely flush any non-public cached configuration data when a user leaves.

As laptop computers can be easily stolen and a cache of configuration data may contain sensitive information, a disconnected mode ACAP client may wish to encrypt and password protect cached configuration information.

11. Acknowledgments

Many thanks to the follow people who have contributed to ACAP over the past four years: Wallace Colyer, Mark Crispin, Jack DeWinter, Rob Earhart, Ned Freed, Randy Gellens, Terry Gray, J. S. Greenfield, Steve Dorner, Steve Hole, Steve Hubert, Dave Roberts, Bart Schaefer, Matt Wall and other participants of the IETF ACAP working group.

12. Authors' Addresses

Chris Newman
Innosoft International, Inc.
1050 Lakes Drive
West Covina, CA 91790 USA

Email: chris.newman@innosoft.com

John Gardiner Myers
Netscape Communications
501 East Middlefield Road
Mail Stop MV-029
Mountain View, CA 94043

Email: jgmyers@netscape.com

Appendices

A. References

[ABNF] Crocker, Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium, Demon Internet Ltd, November 1997.

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

[BASIC-URL] Berners-Lee, Masinter, McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox Coproration, University of Minnesota, December 1994.

<ftp://ds.internic.net/rfc/rfc1738.txt>

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

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

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

[IAB-CHARSET] Weider, Preston, Simonsen, Alvestrand, Atkinson, Crispin, Svanberg, "The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996", RFC 2130, April 1997.

<ftp://ds.internic.net/rfc/rfc2130.txt>

[IMAP4] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, University of Washington, December 1996.

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

[IMAP-ACL] Myers, J., "IMAP4 ACL extension", RFC 2086, Carnegie Mellon, January 1997.

<ftp://ds.internic.net/rfc/rfc2086.txt>

[IMAP-URL] Newman, "IMAP URL Scheme", RFC 2192, Innosoft, July 1997.

<ftp://ds.internic.net/rfc/rfc2192.txt>

[ISO-10646] ISO/IEC 10646-1:1993(E) "Information Technology-- Universal Multiple-octet Coded Character Set (UCS)." See also amendments 1 through 7, plus editorial corrections.

[ISO-C] "Programming languages -- C", ISO/IEC 9899:1990, International Organization for Standardization. This is effectively the same as ANSI C standard X3.159-1989.

[KEYWORDS] Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.

<ftp://ds.internic.net/rfc/rfc2119.txt>

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

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

[REL-URL] Fielding, "Relative Uniform Resource Locators", RFC 1808, UC Irvine, June 1995.

<ftp://ds.internic.net/rfc/rfc1808.txt>

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

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

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

[UNICODE-2] The Unicode Consortium, "The Unicode Standard, Version 2.0", Addison-Wesley, 1996. ISBN 0-201-48345-9.

[US-ASCII] "USA Standard Code for Information Interchange," X3.4. American National Standards Institute: New York (1968).

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

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

B. ACAP Keyword Index

C. Full Copyright Statement