Internet-Draft
6/16/98
draft-oleary-icap-04.doc   Expires 6 months from above date

Internet Calendar Access Protocol (ICAP)

Status of this Memo

Abstract

This Internet Calendar Access Protocol (ICAP) allows a client to access, manipulate and store Calendar information on a server. ICAP employs the iCalendar format [ICAL] for interchange of calendaring information.

ICAP includes operations for creating Calendar stores on a server, opening them and retrieving information about them. When a Calendar Store has been opened, it can be bounded by start and end times so that the client can act on a smaller subset of Calendar information for more efficient operation. ICAP allows users to store new Calendar Objects into their own Calendar store and Calendar stores owned by other users with a single operation.

ICAP supports searching iCalendar objects within a Calendar Store. Searches can be based on any iCalendar property and filtered by iCalendar Component type.

Table of Contents

1. PROTOCOL OVERVIEW

1.1. Conventions Used in this Document

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

The following terms are used in this document to signify the requirements of this specification.

1) MUST, or the adjective REQUIRED, means that the definition is an absolute requirement of the specification.

2) MUST NOT that the definition is an absolute prohibition of the specification.

3) SHOULD means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.

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

5) MAY, or the adjective OPTIONAL, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option.

"Can" is used instead of "may" when referring to a possible circumstance or situation, as opposed to an optional facility of the protocol.

"User" is used to refer to a human user, whereas "client" refers to the software being run by the user.

1.2. Link Level

The ICAP server assumes a reliable, stream oriented transport such as that provided by TCP/IP. When ICAP is used with TCP the server listens on TCP port 7668 (subject to change).

1.3. Commands and Responses
An ICAP 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 response.

All interactions transmitted by client and server are in the form of lines; that is, strings that end with a CRLF. The protocol receiver of an ICAP client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line.
The ICAP server states a greeting similar to the following:

```
S: * OK ICAP Server ready.
```

The greeting is an untagged response from the server which includes a list of the server's capabilities followed by an optional human-readable message. See below for the description of untagged responses. If the ICAP server supports multiple capabilities (see below) they must be presented using a parenthesized list. It is mandatory that the capability ICAP be presented and that it be first in the list of capabilities. If the capability ICAP is not presented, the client cannot proceed and must close the connection immediately.

The server must present an OK response if it is ready to accept a client connection. If the server is not ready to accept such a connect, it must present a BYE response.

More examples of valid connection responses:

```
S: * OK (ICAP X-PigLatin) Server ready.
S: * OK ICAP It's a wonderful day today!
S: * BYE Connection refused. Please try again later.
```

1.3.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with a identifier (typically a short alphanumeric string, e.g. A0001, A0002, etc.) called a "tag". A different tag is generated by the client for each command.

There are two cases in which a line from the client does not represent a complete command. In one case, a command argument is quoted with an octet count (see the description of literal in String under Data Formats); in the other case, the command arguments require server feedback (see the AUTHENTICATE command). In some of these cases, the server sends a command continuation request response if it is ready for the octets (if appropriate) and the remainder of the command. This response is prefixed with the token "+".

Note: If, instead, the server detected an error in the command, it sends a BAD completion response with tag matching the command

(as described below) to reject the command and prevent the client from sending any more of the command.

It is also possible for the server to send a completion 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 protocol receiver of an ICAP server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result response.

1.3.2. 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 completion request is prefixed with the token "+".

A command completion result response indicates the success or
failure of the operation. It is tagged with the same tag as the client
command which began the operation. Thus, if more than one
command is in progress, the tag in a server completion response
identifies the command to which the response applies. There are
three possible server completion responses: OK (indicating success),
NO (indicating failure), or BAD (indicating protocol error such as
unrecognized command or command syntax error).

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 ICAP 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.
Due to the obviously time-critical nature of applications which may use ICAP,
an
ICAP server implementation should attempt to keep
connected clients "current" by sending updates to the client when a
selected Calendar Store is updated.

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

1.4. Data Formats

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

1.4.1. Atom

An atom consists of one or more non-special characters.

1.4.2. Number

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

1.4.3. String

A string is in one of two forms: literal and quoted string. The literal form is the general form of string. The quoted string form is an alternative that avoids the overhead of processing a literal at the cost of 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. In the case of literals transmitted from client to server, the client must wait to receive a command continuation request (described later in this document) before sending the octet data (and the remainder of the command).

A quoted string is a sequence of zero to 1024 7-bit characters, excluding CR and LF, with double quote (<">) characters at each end.

The empty string is respresented as either "" (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 literal must wait to receive a command continuation request.

1.4.4. Parenthesized Lists

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.

1.4.5. NIL

The special atom "NIL" represents the non-existence of a particular data item that is represented as a string or parenthesized list, as distinct from the empty string "" or the empty parenthesized list ().

1.4.6. Dates

All dates given in this document are in a compact form of the ISO **8601 date and time format [ISO-TIME]: YYYYMMDD 'T'** HHMMSS. Hours are always given using the 24 hour clock. A "Z" may be appended to indicate UTC or "Zulu" time (that's GMT to most people). The TZID property parameter MUST NOT be applied to dates whose time values are specified in UTC. The use of local time in a DATE-TIME or TIME value without the TZID property parameter is to be interpreted as a local time value, in the time zone of the selected calendar. Clients can check the time zone of the selected calendar by using the ATTRIBUTES command (see below.)

Note: ICAP servers do not support ISO 8601's week of the year notation. Before storing in an ICAP server, these dates must be converted to the above format.

Examples of valid dates:

```
DTSTART:19980927T0700                          'Sept. 27, 1998
in the local time zone
DTSTART:19980927T1300Z                     'Sept. 27, 1998
in UTC time
DTSTART;TZID=America-NYC:20000101T0000  'New Year's Eve in New York City
```

The form of date and time with UTC offset MUST NOT be used. For example, the following is not valid for a date-time value:

```
DTSTART:19980119T230000-08               'Invalid time
format
```

1.4.7. Periods of Time

A PERIOD value type is used to identify values that contain a precise period of time. The description of this data type has been

borrowed from the iCalendar [ICAL] document. A PERIOD is
defined by the following notation:

```
period     = period-explicit / period-start

period-explicit = date-time "/" date-time
; [ISO 8601] complete representation basic format for a period of
time consisting of a start and end.
; The start MUST be before the end.

period-start = date-time "/" duration
; [ISO 8601] complete representation basic format for a period of
time consisting of a start and
; positive duration of time.
```

If the property permits, multiple "period" values can be using a
COMMA character (US-ASCII decimal 44) separator character.
There are two forms of a period of time. A period of time identified
by its start and its end. This format is expressed as the [ISO 8601]
complete representation, basic format for "DATE-TIME" start of the
period, followed by a SOLIDUS character (US-ASCII decimal 47),
followed by the "DATE-TIME" of the end of the period. The start of
the period MUST be before the end of the period. A period of time
can also be defined by a start time and a positive duration. The
format is expressed as the [ISO 8601] complete representation, basic
format for the "DATE-TIME" start of the period, followed by a
SOLIDUS character (US-ASCII decimal 47), followed by the [ISO
8601] basic format for "DURATION" of the period.

Examples of valid periods of time:

The period starting at 18:00:00 UTC, on January 1, 1999 and ending
at 07:00:00 UTC on January 2, 1999 would be:

```
19990101T180000Z/19970102T070000Z
```

The period start at 18:00:00 on January 1, 1999 and lasting 5 hours
and 30 minutes would be:

```
19990101T180000Z/PT5H30M
```

No additional content value encoding (i.e., BACKSLASH character
encoding) is defined for this value type.

1.5. 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.

1.6. Autologout Timer

If a server has an inactivity autologout timer, that timer MUST be of
at least 10 minutes' duration. The receipt of ANY command from
the client during that interval should suffice to reset the autologout
timer.

1.7. Multiple Commands in Progress

The client is not required to wait for the completion result response
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.

1.8. Calendar Store

The primary purpose of the ICAP protocol is to allow access to, and
the modification of, Calendar Stores. A Calendar Store is defined as
one set of Calendar Objects that are organized chronologically and
given a name. A Calendar Object is one discrete item that may be
posted to a calendar. In ICAP, Calendar Objects are represented in
iCalendar [ICAL] format and can consist of one or more Calendar
Components as described in the iCalendar specification.

Objects within a Calendar Store MUST meet one of the two
following requirements:

? Every Date-Time property of every Object within the Calendar
Store MUST contain a UTC value or a UTC relative value. All
Objects within the Calendar Store MUST be sorted by UTC
using the per-Component rules specified in section 1.9 below.
If all Objects within a Calendar Store are contained within the same
time zone and their time values can all be converted to UTC
using the same TIMEZONE component, then TIMEZONE
components and UTC relative information can be omitted from

individual Objects within the Calendar Store. All Objects within the Calendar Store MUST be sorted by using the per-Component rules specified in section 1.9 below.

1.9. Calendar Objects and Components

iCalendar Objects as specified in [ICAL] consist of a sequence of calendar properties and one or more calendar Components. An ICAP implementation MUST support all valid iCalendar Objects and their Components, with the following qualifiers:

? Calendar Properties, as identified in section 5.4.7 of [ICAL] can be omitted from any Object in a Calendar Store if those Properties are common to all Objects in that Calendar Store. In this case, those Calendar Properties common to every Object in the Calendar Store MUST be returned via the ATTRIBUTES command. If the Calendar Properties for Objects in a Calendar Store can vary from Object to Object, every Object contained within a Calendar Store MUST contain Calendar Properties appropriate to that Object.
? FREEBUSY Components MUST be returned only via the FREEBUSY command.
? Event, To-Do and Journal Components can be intermixed within the same Calendar Store. Event components are sorted according to their DTSTART value. To-Do components are sorted according to their DTSTART value. Journal Components are sorted according to their DTSTART value.
? An Object within a Calendar Store can contain at most ONE Event, To-Do or Journal Component.

1.10. Unique Identifiers

Each ICAP server, Calendar store and Calendar Object must have a unique identifier ("unique ID" or "UID") associated with it. This unique ID must persist across sessions. Unique ID's in ICAP consist of alphanumeric characters only, are exactly 16 characters in length and are case sensitive. Nothing about the structure of the unique ID must be assumed: they are not guaranteed to represent numeric values, ascending in value, etc.

A Calendar store UID need only be unique within the current server and is referred to hereafter as the Calendar Store ID (CSID). A Calendar Object UID need only be unique within its Calendar store and is referred to as the Calendar Object ID (COID).

The exact method for ensuring that UID's are unique is implementation dependent.

Note that the iCalendar specification [ICAL] defines an attribute called "UID" for calendar Objects which must be globally unique. This value can be created by concatenating the server's host name then the CSID and COID.

1.11. Calendar Store Naming

Calendar names can be any string of alphanumeric characters and the characters "_" (underscore), "." (period), "-" (hyphen) and "'" (apostrophe). Calendar names can contain spaces, in which case the whole name must be delimited by double quote characters (see below). Calendar names are case sensitive and must be distinct from all other Calendar stores.

Calendar names can be hierarchical: the hierarchy is read from left (highest level in the hierarchy) to right and delimited by the "/" (forward slash) character. If a hierarchical name is quoted, the entire name must be quoted. An ICAP implementation is NOT required to

support hierarchical naming.

Examples of valid names:

```
"Pete's Calendar"
Product_Calendar
Project1
SpinalTapPerformanceSchedule
Projects/Pete/ICAP
"Projects/Pete O'Leary/ICAP Specification Schedule"
```

Calendar names can contain a user's name delimited by angle braces "<" and ">". Empty angle braces "<>" are meant to refer to the currently authenticated user. This specification refers to "users" and "user names", although these concepts can apply to things other than human beings. For instance, a "user" with their own Calendar store may actually be a resource such as a conference room which exists outside of the Calendar server itself.

The important distinction between user names and store names is that user names very often have meaning outside the implementation of the current server. For instance, a user name may be an SMTP mail address or a Distinguished Name that may be looked up using a

directory service like LDAP. An ICAP implementation must not
assume anything about the structure of a user name.

A user's name by itself, used as a Calendar Store name, must
represent the default Calendar Store (see below) for that user. The
user's name must also be the root by which other Calendar Stores the
user has created can be found (see the LIST command below). A
user name must always be at the leftmost position in the hierarchy of
a Calendar Store name. It is invalid to have a Calendar name with
more than one user name in it.

See the description of the SELECT and LIST commands below for
more discussion about user names and their use.

Valid names with user/resource names:

```
<Pete>
<Pete>/Project_Schedule
<Large_Conference_Room>
"<Palo Alto/Research Building/Large Conference Room>"
```

Invalid names:

```
Users/<Pete>
<Pete>/<Paul>
"Palo Alto/Research Building/<Large Conference Room>"
```

1.12. Default Calendar

Every user local to a calendar server should have a "default
calendar". This is the Calendar store that is most likely to contain up-
to-the-minute information about a person's calendar. The exact
definition of this concept is implementation-dependent. The default
calendar, which can be selected using only the user's name, can be
used by clients to look up free and busy time information for that
user.

1.13. Access Control

ICAP servers should allow for different levels of access control on
user's Calendar stores. The exact definition of this access control is
implementation dependent. A good default choice would be to allow
read-only access to a user's default calendar store via the EXAMINE

command to allow for free and busy time searches.

The server should give a NO response any time a client requests an operation which is not permitted by access control.

For example, a server that allows anonymous read-only browsing of Calendar stores may enforce the following rules:

1. **The client is only shown user's default Calendars when** performing a LIST command.
2. **The client is only allowed to select Calendar stores via the** EXAMINE command.
3. **The STORE command always returns a NO response and allows** no updates of the Calendar store. In this case, the server could return a response to the client indicating where to send a meeting invitation via e-mail to request a meeting with the desired user.
4. **The FETCH command will return a NO response if the user** requests anything more than the flags and summary information of a Calendar Object. This would allow the anonymous user to browse the Calendar of another user in a company which had an "open calendar" policy for all users.
5. **Optionally, for a higher level of security, the server may return a** NO response for an attempted FETCH and allow only the use of the FREEBUSY command. The FREEBUSY command does not return any specific information about the Objects of a user's calendar.

1.14. Server States

An ICAP server is in one of four states. Most commands are valid in only certain states. It is a protocol error for the client to attempt a command while the command is in an inappropriate state. In this case, a server will respond with a BAD or NO (depending upon server implementation) command completion result.

When a command is valid in more than one server state, the description below will list the "Valid States" for that command.

1.13.1. 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.

1.13.2. 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.

1.13.3. Selected State

In selected state, the user has chosen a particular calendar store (or calendar stores) and can perform operations on them.

1.13.4. 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.

2. Scheduling Operations

This section discusses different scheduling operations and how ICAP enables those operations. This section also presents scheduling operations which ICAP does not enable and gives a short discussion of why.

2.1 Operations Supported by ICAP

For illustration purposes, the following is an incomplete list of the scheduling operations that ICAP is intended to support:

2.1.1. Calendar Browsing

ICAP supports the ability to open and browse Calendar Stores via the SELECT and FETCH commands. A client may obtain a list of Calendar Stores available using the LIST command. A user can browse a Calendar that belongs the them or to another user, subject to access control restrictions. The SELECT command actually allows

multiple users' Calendars to be viewed simultaneously.

2.1.2. Freetime Search

ICAP supports the ability to obtain free and busy information about a user in two ways:

1. **The user's default Calendar Store can be browsed as described above.**
2. **The FREEBUSY command can be used to obtain a "snapshot"**
of users' scehdule during a given period of time.

2.1.3. Creating, Deleting and Modifying Calendar Objects

A user can create, delete and modify Objects either in their own Calendar Stores or, subject to access control restrictions, in another user's Calendar store.

The APPEND command is used to create new Calendar Objects in any accessible Calendar Store. The STORE command is used to modify or mark for deletion Calendar Objects in the currently selected Calendar store.

2.1.4. Group Scheduling

ICAP supports the so-called "direct-book" mechanism of creating group meetings by allowing a user to actually place a shared Calendar Object into the Calendar Stores of multiple users. This is not the only way that group scheduling can be performed (see below under "Meeting Invitations").

An ICAP implementation may choose not to support direct-book group scheduling by enforcing access control on users' Calendar Stores.

2.2. Operations Not Supported By ICAP

The following is partially complete list of the scheduling operations that ICAP is NOT intended to support:

2.2.1. Meeting Invitations

ICAP contains no mechanisms for sending so-called "meeting invitations" to Calendar users. Meeting invitations are one means by which group scheduling can be accomplished. This operations may be accomplished by sending iCalendar objects encapsulated as MIME [RFC 1521] content in an SMTP [RFC 821] mail message, as described in the iTIP documents [ITIP].

ICAP contains no mechanisms for allowing access to meeting invitations that have been received by a user. This should be accomplished via message access protocols like IMAP4 [RFC 1730].

2.2.2. Directory Services

ICAP contains no mechanisms for locating a user or obtaining personal information about a user. This operation should be accomplished via LDAP [RFC 1731].

3. ICAP Commands - Any State

3.1. CAPABILITY Command

```
Arguments:      None.

Data:           Mandatory untagged response: CAPABILITY

Result: OK - Command completed
                NO - Command failed
                BAD - Improperly formed command, invalid arguments
```

The CAPABILITY command requests a listing of capabilities that the server supports. The server MUST send a single untagged CAPABILITY response with "ICAP" as one of the listed capabilities before the (tagged) OK response. This listing of capabilities is not dependent upon connection state or user. It is therefore not necessary to issue a CAPABILITY command more than once in a session.

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism. See [RFC 1731] for a discussion of authentication mechanisms that can be used with ICAP. All authentication names are, by definition, part of this specification. For example, the authorization capability for an experimental "blurdybloop" authenticator would be "AUTH=X-BLURDYBLOOP" and not "X-AUTH=BLURDYBLOOP" or "X-AUTH=X-BLURDYBLOOP". Other capability names refer to extensions, revisions, or amendments to this specification. See the documentation of the CAPABILITY response additional information. No capabilities are enabled without explicit client action to invoke the capability. See the section entitled "X-(Atom) Experimental Commands" for information about the form of site or implementation-specific capabilities.

Examples:

```
C: a001 CAPABILITY
S: * CAPABILITY ICAP
S: a001 OK CAPABILITY completed

C: a001 CAPABILITY
S: * CAPABILITY ICAP X-Vegomatic AUTH=X-Secret-Decoder-Rings
S: a001 OK CAPABILITY completed
```

```
C: a001 CAPABILITY
S: * CAPABILITY ICAP AUTH=KERBEROS_V4
S: a001 OK CAPABILITY completed
```

3.2. NOOP Command

```
Arguments:      None

Data:           Optional untagged responses.

Result: OK - Command completed
                BAD - Improperly formed command, arguments
supplied which are not allowed
```

An ICAP server must support this command. The NOOP command always succeeds. It does nothing.

Since any command can return a status update as untagged data, the NOOP command can be used as a periodic poll during a period of inactivity. The NOOP command can also be used to reset any inactivity autologout timer on the server. The ICAP server implementation should attempt to ensure that the NOOP commands completes in as little time as possible.

Examples:

```
C: A001 NOOP
S: A001 OK NOOP Completed.

C: A002 NOOP
S: * 42 EXISTS
S: * 1 RECENT
S: A002 OK NOOP Completed.
```

3.4. LOGOUT Command

```
Arguments:      None

Data:           None

Result: OK - Command completed.
                NO - Command failed, this would indicate that
     something is seriously wrong.
                BAD - Improperly formed command, arguments
     supplied which are not allowed
```

An ICAP server must support this command, closing all open selected Calendars and disconnecting. If a NO response is returned by this command, the server should return some human-readable information describing why the Logout cannot occur and what the user can do to correct the situation. The server must send an

untagged BYE response before the connection is closed and the
command completes.

Example:

```
C: A001 LOGOUT
S: * BYE ICAP Server logging out.
S: A001 OK LOGOUT Completed.
```

3.5. X-(Atom) Experimental Commands

Arguments: implementation defined

Data: implementation defined

Result: OK - command completed
NO - failure
BAD - command unknown or arguments invalid

Any command prefixed with an "X-" is an experimental command.
Commands which are not part of this specification MUST use the
"X-" prefix.

Any added untagged responses issued by an experimental command
MUST also be prefixed with an X. Server implementations MUST
NOT send any such untagged responses, unless the client requested it
by issuing the associated experimental command.

Example:

```
C: a441 CAPABILITY
S: * CAPABILITY ICAP AUTH=KERBEROS_V4 X-PIG-LATIN
S: a441 OK CAPABILITY completed
C: A442 X-PIG-LATIN
S: * XPIG-LATIN ow-nay eaking-spay ig-pay atin-lay
S: A442 OK X-PIG-LATIN ompleted-cay
```

4. ICAP Commands - Non-Authenticated State

4.1. AUTHENTICATE Command

Arguments: Authentication mechanism name

Data: None.

```
Result: OK - Command completed, in Authenticated state
                NO - Command failed, still in Non-Authenticated state
                BAD - Improperly formed command, invalid arguments,
      still Non-Authenticated.
```

The AUTHENTICATE command indicates an authentication

mechanism, such as described in [RFC 1731], to the server. If the server supports the requested authentication mechanism, it performs an authentication protocol exchange to authenticate and identify the

client. It MAY also negotiate an OPTIONAL protection mechanism for subsequent protocol interactions. If the requested authentication mechanism is not supported, the server SHOULD reject the AUTHENTICATE command by sending a tagged NO response.

The authentication protocol exchange consists of a series of server challenges and client answers that are specific to the authentication mechanism. A server challenge consists of a command continuation request response with the "+" token followed by a BASE64 encoded string. The client answer consists of a line consisting of a BASE64 encoded string. If the client wishes to cancel an authentication exchange, it issues a line with a single "*". If the server receives such an answer, it MUST reject the AUTHENTICATE command by sending a tagged BAD response.

A protection mechanism provides integrity and privacy protection to the protocol session. If a protection mechanism is negotiated, it is applied to all subsequent data sent over the connection. The protection mechanism takes effect immediately following the CRLF that concludes the authentication exchange for the client, and the CRLF of the tagged OK response for the server. Once the protection mechanism is in effect, the stream of command and response octets is processed into buffers of ciphertext. Each buffer is transferred over the connection as a stream of octets prepended with a four octet field in network byte order that represents the length of the following data. The maximum ciphertext buffer length is defined by the protection mechanism.

Authentication mechanisms are OPTIONAL. Protection mechanisms are also OPTIONAL; an authentication mechanism MAY be implemented without any protection mechanism. If an AUTHENTICATE command fails with a NO response, the client MAY try another authentication mechanism by issuing another

AUTHENTICATE command, or MAY attempt to authenticate by
using the LOGIN command. In other words, the client MAY request
authentication types in decreasing order of preference, with the
LOGIN command as a last resort.

Example:

```
S: * OK ICAP KerberosV4 Server
C: A001 AUTHENTICATE KERBEROS_V4
S: + AmFYig==
C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25
C: a4DT+nZImJjn TNHJUtxAA+o0KPKfHEcAFs9a3CL5Oebe/ydHJUwYFd
C: WwuQ1MWiy6IesKvjL5rL9WjXUb9MwT9bpObYLGOKi1Qh
S: + or//EoAADZI=
```

```
C: DiAF5A4gA+oOIALuBkAAmw==
S: A001 OK Kerberos V4 authentication successful
```

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

4.2. LOGIN Command

Arguments: User name for login.
Optional password.

Data: None.

Result: OK - Command completed, in Authenticated state
                NO - Command failed, still in Non-Authenticated state
                BAD - Improperly formed command, invalid arguments,
     still in Non-Authenticated state.

The LOGIN command identifies the client to the server and carries
the plaintext password authenticating this user.

This is accomplished by allowing a LOGIN command specifying a
user name of "Anonymous" and any password. The anonymous user
should be allowed to enter the Authenticated state, with access
control restrictions. It is recommended that anonymous users be
given read-only permission to users' default Calendar stores to allow
free busy time searches.

Example:

```
C: a001 LOGIN Pete Mumblefrtoz
```

S: a001 OK LOGIN completed

5. ICAP Commands - Authenticated State

5.1. SELECT Command

```
Arguments:      Name of the Calendar store to select.
                Optional date range

Data:           Mandatory untagged response: FLAGS, EXISTS
Optional untagged responses: RECENT

Result: OK - now in Selected state
NO - no such Calendar store, can't access Calendar store
                BAD - Invalid arguments

Valid states:   Authenticated, Selected
```

The SELECT command selects the Calendar store for the current user so that Calendar Objects can be queried and retrieved. Multiple

Calendar stores can be selected by reissuing the SELECT command. In this way, a composite view of the Calendar stores can be created.

This composite calendar can then be used to allow browsing of group calendars and creating of group meetings (see below). It is invalid to select the same Calendar store more than once.

The name parameter can be the name of a Calendar and optionally a user name. If a user name is given, then it is bracketed with angle braces "<" and ">" and added before the Calendar store name. The default Calendar store of a user can be selected by using only the user's name bracketed by angles. The default calendar store of the current user can be selected by using angles"<>". All commands which take a Calendar store name as a parameter can accept a user name in this manner. Which Calendar store is selected as a default is implementation dependent. It is recommended that the default store be whichever Calendar store most accurately represents the user's actual schedule, so that it can be queried to find freetime (see the FREEBUSY command below). Any Calendar store name given that does not include an explicit username must be assumed to belong to the current user. In other words, a prefix of "<>/" can be implicitly added before any Calendar store name that does not give an explicit user name.

The user name may be specified as <"username"@"hostname">. If

"hostname" is the name of the current host machine, then
"hostname" may be omitted and the form is: <"username">. If
"hostname" is included, and it is different from the current host
machine name, the server can either reject the name with a NO
response or attempt to connect to the specified host machine on
behalf of the user and issue an OK response if successful. If a NO
response is given by the server because the given user's calendar
information is not located on the host but the server knows where, a
reference name in angle brackets may be included as part of the
response.
The EXISTS response should return the total number of Objects
currently selected. When multiple Calendar stores are selected, this is
the combined total of all Objects selected in all the Calendar stores.

Optionally, a date range may be specified as part of the SELECT
command. When a date range is provided, the SELECT command
MUST limit the selection to entries that fall between the given range.
If a date range is specified the server MUST resolve any recurring
rules in the calendar's entries and present a distinct entry for each
instance of the recurring set. For example, suppose a calendar
contained only one entry which was a recurring event that repeated
every week on Wednesdays. If a SELECT command were given
which specified a date range of 4 weeks, the server MUST return 4
entries as part of the selection.

All Objects in the selected Calendar stores must be presented by the
server in chronological order from 1 to the number of Objects
selected by reissuing the SELECT command and the server cannot
support presenting multiple Calendars in chronological order the
server must issue a NO response for additional Calendars as they are
selected.

Note for readers familiar with IMAP4: The ICAP protocol does not
require any correlation between Object numbers and unique ID's as
IMAP4 does. UID's are not required to be strictly ascending.
Furthermore, UID's in ICAP cannot change between sessions as in
IMAP4 (per the UIDVALIDITY response).

The RECENT response should be given if new Calendar Objects
have appear in the selected Calendar since it was last selected. This
may occur when someone else's Calendar store is selected or
possibly when someone else - an assistant perhaps - modifies the
user's Calendar

The FLAGS response should list the flags supported by this Calendar store. Note that when multiple Calendar stores are selected, the FLAGS supported should be the intersection of those supported by all the Calendars.

Flags current supported are:

\Deleted - Object is marked for deletion.
\Recent - Object has been added since the last time that this Calendar store was selected.
\Repeating - Object is one of a repeating set of Objects.
\Tentative - The Object is marked as being "tentative" - not yet confirmed - by the Calendar's owner.
\Seen - The user has already seen this Object. Set by default when the user creates an Object in their own store.

Example interactions:

```
C: A001 SELECT
S: * 123 EXISTS
S: * FLAGS (\Deleted \Recent \Repeating)
S: A001 OK SELECT Completed
C: A002 RANGE 19970101T000000 19970130T235959
```

The sequence above selects the current user's default Calendar store. It then sends a Selected state command called RANGE (see below).

```
C: A001 SELECT <>/Section1
S: * 45 EXISTS
S: * FLAGS (\Deleted \Recent \Repeating)
S: A001 OK SELECT completed
C: A002 RANGE 19970101T000000 19970130T235959
```

The sequence above selects the current user's Calendar store called "Section1". It then sends a Selected state command called RANGE.

```
C: A001 SELECT <>
S: * 23 EXISTS
S: * FLAGS (\Deleted \Recent \Repeating)
S: A001 OK SELECT completed
C: A002 SELECT <Bob>
S: * 56 EXISTSS: * FLAGS (\Deleted \Recent \Repeating)
S: A002 OK SELECT completed
C: A003 SELECT <Sally>
S: * 123 EXISTS
S: * FLAGS (\Deleted \Recent \Repeating)
```

```
S: A003 OK SELECT completed
```

This sequence uses multiple SELECT commands to open the current user's default Calendar store plus the default Calendar stores of Bob and Sally. Note that the EXISTS response from the command contains the number of Calendar Objects in all of the Calendars currently selected.

```
C: A001 SELECT <jyip@clearblue.com>
S: A001 NO SELECT No such user.

C: A001 SELECT <paul@broadbase.com>
S: * 27 EXISTS
S: * FLAGS (\Deleted \Recent \Repeating)
S: A001 OK SELECT completed.

C: A001 SELECT <poleary@clearblue.com>
S: A001 NO SELECT Try <poleary@yosemite.clearblue.com>
```

These sequences demonstrate how a server might handle a SELECT command where the given user's Calendar store is not on the current host. In the second example, the server knows the location of the user's Calendar host and supplies that information to the client.

5.2. EXAMINE Command

```
Data:           None.
Result: OK - Command completed
                NO - Command failed
                BAD - Improperly formed command, invalid
arguments

Valid states:   Authenticated, Selected
```

This command is identical to SELECT except that the selected Calendar store is returned read only. EXAMINE and SELECT

command can be given in combination to select multiple calendars for browsing. The operation is identical in all respects, regardless of which command is used, except that a Calendar store selected via EXAMINE cannot be modified or updated in any way.

5.3. CREATE Command

```
Arguments:      Name of Calendar store to create.
```

Data: None.

Result: OK - Calendar store created.
                NO - Calendar store not created.
                BAD - Invalid arguments.

Valid states: Authenticated, Selected

Creates a new Calendar store with the given name. Calendar store names must begin with an alphabetic character and consist of alphanumeric characters. Calendar names are not case sensitive. It is illegal to create more than one Calendar store with the same name. CREATE does not automatically select the Calendar store created.

ICAP servers are NOT required to support hierarchical names. If a client attempts to create a Calendar Store with a hierarchical name on a server which does not support hierarchical names, the server MUST return a response of NO to the CREATE command.

If the Calendar name is suffixed with the hierarchy separator "/", this is a declaration that the client intends to create Calendar names under this name in the hierarchy. Server implementations that do not require this declaration MUST ignore it.

If the hierarchy separator character appears elsewhere in the name, the server SHOULD create any superior hierarchical names that are needed for the CREATE command to complete successfully. In other words, an attempt to create "foo/bar/zap" on a server SHOULD create foo/ and foo/bar/ if they do not already exist.

Example:

```
C: A001 CREATE Projects/
S: A001 OK CREATE Completed
C: A001 CREATE Projects/Purple
S: A001 OK CREATE Completed
C: A001 CREATE Projects/Green
S: A001 OK CREATE Completed
```

The above example creates two Calendar stores for the default user below the hierarchical name "Projects".

5.4. DELETE Command

Arguments: Name of Calendar store to delete.

```
Data:           None.

Result: OK - Calendar store deleted.
                NO - Calendar store not deleted.
                BAD - Invalid arguments.

Valid states:   Authenticated, Selected
```

Deletes the Calendar store with the given name. If this command is given from the Selected state, a currently selected Calendar cannot be deleted.

Example:

```
C: A001 DELETE Projects/Purple
S: A001 OK DELETE Completed.
```

5.5. RENAME Command

```
Arguments:      Name of Calendar store to rename.
                New Calendar store name.

Data:           None.
Result: OK - Calendar store renamed
                NO - Calendar store not renamed.
                BAD - Invalid arguments

Valid states:   Authenticated, Selected
```

The RENAME command changes the name of a Calendar store. A tagged OK response is returned only if the Calendar has been renamed. It is an error to attempt to rename from a Calendar name that does not exist or to a Calendar name that already exists. Any error in renaming will return a tagged NO response.

If the hierarchy separator character appears in the new Calendar store name, the server SHOULD create any superior hierarchical names that are needed for the RENAME command to complete successfully. In other words, an attempt to rename a Calendar to "foo/bar/zap" on a server SHOULD create foo/ and foo/bar/ if they do not already exist.

Example:

```
C: A001 RENAME Projects/Purple Projects/Orange
S: A001 OK RENAME Completed.
C: A001 RENAME Projects/Green Completed/Green
S: A001 OK RENAME Completed.
```

5.6. LIST Command

```
Arguments:      Store name with optional wildcard

Data:           Untagged responses: LIST

Result: OK - List complete.
                NO - List failed, no stores found that match the given
                     pattern.
                BAD - What did that pattern mean anyway?

Valid states:   Authenticated, Selected
```

The LIST command returns an untagged LIST response for each Calendar store that matches the given reference and store name. The "*" character is a wildcard and can be used only in the right-most position in the given store name. The "*" character matches any length string of valid Calendar Store name characters.

ICAP uses the "/" character to delimit levels of hierarchy: if the Calendar store name returned by the LIST command ends with a "/" character, then a level of hierarchy exists below that store name. If that store name cannot be selected, the LIST response must include the \Noselect flag, as described below in the LIST response.

The server must hide all Calendar stores that the current user does not have access to.

These reference names should be interpreted in the following manner:

```
* - all top-level Calendar stores accessible to the current user
<*> - all users accessible by the server
<> - the currently authenticated user
```

Examples:

```
C: A001 LIST <*>
S: * LIST () <Ann>
S: * LIST () <Bob>
S: * LIST () <Pete>
S: * LIST () <Fred>
C: A002 LIST <Ann>/*
S: * LIST () <Ann>/Project_1
S: * LIST () <Ann>/Project_2
S: A001 OK LIST Completed.
```

```
C: A001 LIST *
S: * LIST () Business
S: * LIST () Private
```

```
S: * LIST () CorporateEvents
S: * LIST (\Noselect) Projects/
S: A001 OK LIST Completed.
C: A002 LIST Projects/*
S: * LIST () Projects/ICAP_Spec
S: * LIST () "Projects/Vacation Plans"
S: A002 OK LIST Completed.
```

5.7. LSUB Command

```
Arguments:      Store name with optional wildcard

Data:           Untagged responses: LIST

Result: OK - List complete.
                NO - List failed, no stores found that match the
                     given pattern.
                BAD - What did that pattern mean anyway?

Valid states:   Authenticated, Selected
```

The LSUB command is identical to the LIST command except that it
returns Calendar names from those that the current user has
subscribed to.

5.8. SUBSCRIBE Command

```
Arguments:      Calendar name

Data:           None

Result: OK - Subscribe complete.
                NO - Subscribe failed, no stores found that match
name.
                BAD - Invalid arguments

Valid states:   Authenticated, Selected
```

The SUBSCRIBE command adds the given Calendar store name to
the list of subscribed stores that will be returned by the LSUB
command.

Example:

```
C: A001 SUBSCRIBE Corporate_Calendar/Barbecues
S: A001 OK SUBSCRIBE Completed.
```

5.9. UNSUBSCRIBE Command

Arguments: Calendar name

```
Data:           None

Result: OK - Unsubscribe complete.
                NO - Unsubscribe failed, no stores found that
                     match name.
                BAD - Invalid arguments

Valid states:   Authenticated, Selected
```

The UNSUBSCRIBE command removes the given Calendar store
name from the list of subscribed stores that will be returned by the
LSUB command.

Example:

```
C: A001 UNSUBSCRIBE <Corporate_Calendar>/Barbecues
S: A001 OK UNSUBSCRIBE Completed.
```

5.10. APPEND Command

```
Arguments:      Calendar store name list or NIL for currently
selected stores
                Flags list
                Calendar Object literal

Data:           None.

Result: OK - Command completed
                NO - Command failed, no Objects were added to
                     any calendar store
                BAD - Improperly formed command, invalid
                     arguments, no Objects added

Valid states:   Authenticated, Selected
```

The APPEND command creates a new Calendar Object in the given

Calendar Stores. If the destination Calendar Store does not exist, the server must return a NO response.

In the Selected state, a NIL atom may be given for the list of Calendar store names to operate on.

The server can send an optional untagged response for each user or calendar that is specified. The NO response can be used to indicate that the store failed in a certain user's calendar with a response code of "REFUSED" followed by the name of the calendar refusing. The server can optionally return a response code of "MAILTO" followed by the calendar name and a mail address that can accept an invitation request for the given calendar. The server may return OK responses for selected Calendars to indicate that the STORE completed successfully in that Calendar but that some special condition exists.

The server may send a response code of "TENTATIVE" to indicate that a new Calendar Object was created marked as \Tentative. The server may send a response code of "SENT" to indicate that a meeting invitation was sent to the owner of the Calendar store.

The \StoreAll flag should be given when the client is creating a new Calendar Object and wants to guarantee that the Object will be created in all of the selected Calendar stores simultaneously. If the server cannot store to at least one of the selected Calendars, it must not store to any of them and must return a NO response indicating that the command failed. The server may still return untagged responses indicating which Calendar stores failed.

The \NoConflict should be given when the Object being appended to the Calendar Stores specified cannot have a time conflict with any Object on any of the Calendar Stores. If a time conflict exists, the server MUST return a NO response and MUST not modify any of the specified Calendar stores.

New Calendar Objects added to a Calendar store with the APPEND command MUST contain all required iCalendar properties. If a required property is missing the server MUST return a NO response and MUST not modify any of the specified Calendar stores.

Examples:

```
C: A001 APPEND Personal_Calendar () {88}
C: BEGIN: VCALENDAR
C: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
C: VERSION: 2.0
```

```
C: BEGIN:VTIMEZONE

C: TZID:America-New_York
C: LAST-MODIFIED:19870101T000000Z
C: TZURL:http://zones.stds_r_us.net/tz/America-New_York
C: BEGIN:STANDARD
C: DTSTART:19671029T020000
C: RRULE:FREQ=YEARLY;BYDAY=-1SU;BYMONTH=10
C: TZOFFSETFROM:-0400
C: TZOFFSETTO:-0500
C: TZNAME:EST
C: END:STANDARD
C: BEGIN:DAYLIGHT
C: DTSTART:19870405T020000
C: RRULE:FREQ=YEARLY;BYDAY=1SU;BYMONTH=4
C: TZOFFSETFROM:-0500
C: TZOFFSETTO:-0400
C: TZNAME:EDT
C: END:DAYLIGHT
C: END:VTIMEZONE
C: BEGIN: VEVENT
```

```
C: DTSTART;TZID=America-New_York: 19980606T140000
C: DTEND;TZID=America-New_York: 19980606T150000
C: SUMMARY: Meeting with Pete
C: END: VEVENT
C: END: VCALENDAR
C:
S: A001 OK APPEND completed

C: A001 APPEND (<Ann> <Bob> <Fred>) (\Tentative) {88}
C: BEGIN: VCALENDAR
C: PRODID:-//Amplitude Software//NONSGML Reserve 1.5//EN
C: VERSION: 2.0
C: BEGIN: VEVENT
C: DTSTART: 19980927T163000Z
C: DTEND:19980927T190000Z
C: SUMMARY: Liz's Birthday Party
C: END: VEVENT
C: END: VCALENDAR
C:
S: A001 OK APPEND completed
```

5.11. ATTRIBUTES Command

Arguments:	Calendar store name Name of attributes to fetch
Data:	Untagged FETCH response.

Result: OK - ATTRIBUTES completed
NO - ATTRIBUTES failed, no attributes were returned
BAD - Improperly formed command, invalid arguments

Valid states: Authenticated, Selected

The ATTRIBUTES command returns information about the specified Calendar store. This command is similar in operation to the FETCH command (see below) except that it acts on Calendar stores instead of the Objects in them.

The ATTRIBUTES command currently supports fetching the following attributes:

FLAGS	Returns the flags supported by this Calendar store, as when the store is selected
TYPE	Flags which represents this Calendar store's type. See below.
CSID	The unique identifier of this Calendar.
COMPONENTS	An iCalendar object, see below

TIMEZONE	An iCalendar object containing a
TIMEZONE	Component, see below.

The TYPE flags currently supported are:

\Default	This container is a user's default Calendar store
\Resource	This container is actually owned by a resource.

The COMPONENTS attribute can be used by a client to determine which Calendar Components that can be stored within a Calendar

Store, along with the names and types of the properties of those Calendar Components. The server MUST return an iCalendar object which contains a "model" of all the Components the Calendar Store supports. The "model" Components MUST contain all of the properties that the Component can store. The Properties should specify a VALUE property parameter that identifies the data-type of the property. The property value MUST be an null string.

The server does not have to return a TIMEZONE Component to
indicate that this Component is supported. The server MUST return
an ALARM Component if this Component is supported.
The TIMEZONE attribute can be used by the client to determine
time zone information about a specific Calendar Store. The server
can return an Object containing zero or more TIMEZONE
Components. If no TIMEZONE Components are returned, then
every object fetched from the Calendar Store MUST contain at least
one TIMEZONE Component. If one or more TIMEZONE
Components are returned, then those Components apply to every
Object fetched from the Calendar Store. The client MUST NOT
expect to be able to store an Object that contains a TIMEZONE
Component other than those returned by this attribute.

Examples:

```
C: A001 ATTRIBUTES <Pete> (FLAGS TYPE CSID)
S: * FETCH (FLAGS (\Deleted \Seen \Recent) TYPE \Default CSID
1234123412341234)
S: A001 OK ATTRIBUTES Completed.
```

In the above example, the client queries the server for the flags
supported by the specified Calendar Store, the Calendar Store type
and its UID.

```
C: A001 ATTRIBUTES <Pete> COMPONENTS
S: * FETCH COMPONENTS {200}
S: BEGIN: VCALENDAR
S: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
S: VERSION: 2.0
S: BEGIN: VEVENT
S: ATTACH; VALUE=URL:
S: DESCRIPTION; VALUE=TEXT:
```

```
S: DTSTART; VALUE=DATE-TIME:
S: DTEND; VALUE=DATE-TIME:
S: STATUS:
S: SUMMARY; VALUE=TEXT:
S: UID:
S: X-BILLING_CODE, VALUE=INTEGER:
S: X-CLIENT_NAME; VALUE=TEXT:
S: END: VEVENT
S: END: VCALENDAR
S: A001 OK ATTRIBUTES Completed.
```

In the above example, the client queries the server for the type of Components supported by the Calendar Store. The server returns an Object which specifies that only Event Components can be stored in the Calendar Store. The Event Components can store the required Event Component properties, plus the standard properties ATTACH, SUMMARY, STATUS and UID. Also, the non-standard properties X-BILLING_CODE and X-CLIENT_NAME can be stored. This appears to be a lawyer's calendar...

5.12. FREEBUSY Command

```
Arguments:      Calendar store name list or NIL for currently
selected stores
Start of search range in ISO8601 format
                End of search range in ISO8601 format

Data:           Mandatory untagged response: FETCH

Result: OK - Command completed
                NO - Command failed, no freetime found
                BAD - Improperly formed command, invalid
                        Arguments

Valid states:   Authenticated, Selected
```

The FREEBUSY command searches the currently selected Calendars, bounded by a start and end time, and returns a list of intervals during which an event of the given length of time can be scheduled. See below under "Example Sessions" for and example of the use of the FREEBUSY command.

As is the case in the RANGE command, the start time given is included in the search range of the FREEBUSY command and the end time is excluded.

In the Selected state, a NIL atom may be given for the list of Calendar store names to operate on.

The freetime data is returned in an untagged FETCH response in iCalendar FREEBUSY format. The FETCH attribute CSNAME must

be returned along with the FETCH response if more than one Calendar Store name was specified. An ATTENDEE property within the FREEBUSY object return may also contain the name of the person corresponding to the FREEBUSY FETCH result. If every

FREEBUSY component within the FETCH responses returned by
the command contains an ATTENDEE property, then CSNAME is
not required. The DTSTART and DTEND properties of the
FREEBUSY object MUST match the start and end dates specified.

Example:

```
C: A001 FREEBUSY (<Ann> <Bob>) 1998930T0900Z 1998930T1700Z
S: * 1 FETCH (CSNAME <Ann> ICAL {88}
S: BEGIN: VCALENDAR
S: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
S: VERSION: 2.0
S: BEGIN: VFREEBUSY
S: ATTENDEE: Ann
S: DTSTART: 1998930T0900
S: DTEND: 1998930T1700
S: FREEBUSY; VALUE=PERIOD-START: 1998930T1000/PT1H, 1998930T1200/PT1H
S: END: VFREEBUSY
S: END: VCALENDAR
S: )
S: A001 OK FREEBUSY
```

In the example above, only two busy periods were found for the
given time range: Ann is busy from 10 to 11 on 19970903 and also
busy from 12 to 1 o'clock on the same day.

6. ICAP Commands - Selected State

6.1. CLOSE Command

```
Arguments:      Optional name of user and or Calendar store.

Data:           Optional untagged EXISTS response.

Result: OK - Command completed successfully
                NO - Calendar name or user is not selected.
                BAD - Invalid argument, calendar name is invalid
```

The CLOSE closes the currently selected Calendar store or one of
the Calendar stores currently selected. When no parameter is
supplied, all currently selected Calendars are closed. When a
parameter is supplied, it should be of the same form for Calendar
names given in the SELECT command. If the user has not previously
issued a SELECT command with the exact name given in the
CLOSE command, a NO response is returned.

Examples:

```
C: A001 CLOSE
S: A001 OK CLOSE Completed.

C: A001 CLOSE <Bob>
S: * 11 EXISTS
S: A001 CLOSE Completed.
```

6.2. RANGE Command

```
Arguments:      Start date/time in ISO8601 format.
                End date/time in ISO8601 format.

Data:           Mandatory untagged responses: EXISTS

Result: OK - Command completed successfully, date range
                  has been set
                NO - An error occurred while setting the date
                  range
                BAD - Bad date format
```

The RANGE command sets a date/time range for the currently selected Calendars and returns a EXISTS response with the number of items in the range.

Either the start time or end time can be replaced with the wildcard character "*" in which case lower or upper bound, respectively, is placed on the current date range.

The start time given must be included in the range. The end time given must be excluded from the range.

If any Object in the selected Calendar store contains a set of recurrence rules that resolve to dates within the specified date range, then that Object MUST appear once for each date resolved within the specified range. In other words, an Object may count for more than one Object in the EXISTS response returned by the RANGE command if it is a recurring Object.

Example:

```
C: A001 RANGE 19971230T0900 19971230T1700
S: * 9 EXISTS
S: A001 OK RANGE
```

The above example selects all Calendar Objects from 0900 to 1700 on 1997 December 30 in the slected calendar's local time zone.

```
C: A001 RANGE 19970730Z 19970731Z
S: * 12 EXISTS
```

```
S: A001 OK RANGE
```

The above example selects all Calendar Objects on 1997 July 30, GMT.

```
C: A001 RANGE 19970101 19980101
S: * 56 EXISTS
S: A001 OK RANGE
```

The above example selects all Calendar Objects in 1997 in the
selected calendar's local time.

6.3. CHECK Command

Arguments: None

Data: None.

```
Result: OK - Command completed
                NO - Command failed
                BAD - Improperly formed command, invalid
                     arguments
```

An ICAP server given the CHECK command should perform any
housekeeping operations that ensure the integrity of the currently
selected Calendar stores. It is expected that the CHECK command
may take some time to complete.

Example:

```
C: A001 CHECK
S: A001 OK CHECK
```

6.4. EXPUNGE Command

Arguments: None

Data: Mandatory untagged response: EXPUNGE

```
Result: OK - Command completed
                NO - Command failed, no items were removed
                BAD - Improperly formed command, no items
     were removed
```

The EXPUNGE command permanently removes from the currently
selected Calendar stores all Objects that have the \Deleted flag set.
Before returning an OK to the client, an untagged EXPUNGE

response is sent for each Object that is removed.

Example:

```
C: A001 EXPUNGE
S: * 3 EXPUNGE
S: * 3 EXPUNGE
```

```
S: * 5 EXPUNGE
S: * 8 EXPUNGE
S: A001 OK EXPUNGE completed
```

Note: in this example, Objects 3, 4, 7, and 11 had the \Deleted flag set. See the description of the EXPUNGE response for further explanation.

6.5. FETCH Command

Arguments:	Set of calendar Objects to fetch. Item names.
Data:	Untagged responses: FETCH
Result:	OK - fetch completed NO - fetch error, no items were fetched BAD - command unknown or invalid arguments

The FETCH command retrieves information about the specified Objects. Objects can addressed for retrieval in 3 different ways:

Index number - A number from 1 to the number of Objects in the current selection as returned in the last EXISTS response from the server.

Multiple index - numbers MAY be specified enclosed in parenthesis.

Unique ID - The unique ID of the Object may be specified as described in the UID command below.

Search criteria - A set of search criteria, as defined in the SEARCH command below. The search criteria MUST be enclosed in curly braces: "{" and "}".

The information to be fetched is specified using the following item names:

FLAGS   The flags associated with this calendar

Object	
ICAL	iCalendar object format. All properties MUST be returned.
ICAL.SIZE	The size of the iCalendar iCalendar, in octets.
ICAL.REQUIRED	Only required iCalendar attribute information in iCalendar format.
ICAL.BRIEF	Only DTSTART, DTEND and SUMMARY attributes in iCalendar format.
ICAL.propertyName	A specific iCalendar property.
UID	The unique ID of the calendar Object (the COID).

CSID	The unique ID of the Calendar Store that contains the fetched Object.
CSNAME	The name of the Calendar Store that contains the fetched Object.

The FLAGS item returns any flags that are currently set for the given calendar Object.

The ICAL item returns the full iCalendar Object specified, including any time zone and access control information. Additionally, if any non-standard properties are associated with the Object (preceded with X-) these properties MUST be returned as well.

The ICAL.SIZE item returns the size of the iCalendar object in octets.

The ICAL.REQUIRED item returns only the iCalendar properties defined as required in [ICAL] for the corresponding component type. Optional and non-standard properties MUST not be returned.

The ICAL.BRIEF item returns only the DTSTART, DTEND and SUMMARY properties for the specified Objects.

The ICAL.propertyName item returns only the specified property for each Object requested. For example, "ICAL.DTSTART" will return only the DTSTART property.

If more than one ICAL item is specified, the server should return only ONE Object per FETCH response. The server MUST return the union of the ICAL items specified.

The UID item returns the unique identifier of the calendar Object.

The CSID item returns the unique calendar store identifier of the Calendar which contains the specified Object. The client can use this value to distinguish Objects when multiple Calendar Stores are selected.

The CSNAME item returns the name of the Calendar which contains the specified Object. The client can use this value to distinguish Objects when multiple Calendar Stores are selected.

Multiple items MAY be specified by surrounding them in parenthesis. All information requested by the client is returned via a FETCH response from the server. See the description of the FETCH response below.

Note that items returned by FETCH should always be returned in ascending chronological order, even when multiple Calendar stores are selected.

Example:

```
C: A001 FETCH 1 FLAGS
S: * 1 FETCH FLAGS  (\Deleted \Seen)
S: A001 OK FETCH

C: A001 FETCH 1 (FLAGS ICAL.REQUIRED)
S: * 1 FETCH (FLAGS  (\Deleted \Seen) ICAL.REQUIRED {96}
S: BEGIN: VCALENDAR
S: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
S: VERSION: 2.0
S: BEGIN: VEVENT
S: DTBEGIN: 1998701T0300Z
S: DTEND: 1998701T0400Z
S: SUMMARY: Test meeting
S: END: VEVENT
S: END: VCALENDAR
S: )
S: A001 OK FETCH

C: A001 FETCH 1:4 UID
S: * 1 FETCH UID 1234123412341234
S: * 2 FETCH UID 5678567856785678
S: * 3 FETCH UID 2345234523452345
S: * 4 FETCH UID abcdabcdabcdabcd
S: A001 OK FETCH
```

6.6. STORE Command

```
Arguments:      Calendar Object set
                Calendar Object data item name
                Calendar Object data item value

Data:           None.

Result: OK - Command completed
                NO - Command failed, no Objects were added to
                     any calendar store
                BAD - Improperly formed command, invalid
                     arguments, no Objects added
```

Calendar data items:

+FLAGS

Set the flag list of the given Calendar Objects. In the STORE command, one additional flag can be given: \StoreAll. See below for the use of this flag.

-FLAGS

Remove the flag argument from the flags of the given Calendar Objects.

ICAL

Updates the iCalendar data associated with this Calendar Object. When this form of the STORE command is used, the Calendar data item must be supplied as a literal. The data must be a iCalendar object.

If a value of "0" is given for the Calendar Object set, then a new Object is created. This functionality is similar to performing an APPEND command except that it allows the client to check the availability of the Calendar stores before attempting to create new Objects. If there is more than one Calendar store selected in the given Selected object, then the STORE command will add the new Object to all of the Calendar stores. This will cause the EXISTS count for the current selection to increase by 1 for each currently selected Calendar store.

The server can send an optional untagged response for each user or calendar that is currently selected. The NO response can be used to indicate that the store failed in a certain user's calendar with a response code of "REFUSED" followed by the name of the calendar refusing. The server can optionally return a response code of "MAILTO" followed by the calendar name and a mail address that can accept an invitation request for the given calendar. See the example below. The server may return OK responses for selected Calendars to indicate that the STORE completed successfully in that Calendar but that some special condition exists. The server may send a response code of "TENTATIVE" to indicate that a new Calendar Object was created marked as \Tentative. The server may send a response code of "SENT" to indicate that a meeting invitation was sent to the owner of the Calendar store. In the case of a "TENTATIVE" response from the server, the EXISTS count for the selected Calendars MUST be increased. When a SENT response is given, the EXISTS count MUST NOT be increased.

The \NoConflict should be given when the Object being appended to the Calendar Stores selected cannot have a time conflict with any Object on any of the Calendar Stores. If a time conflict exists, the server MUST return a NO response and MUST not modify any of the selected Calendar stores.

The \StoreAll flag should be given when the client is creating a new Calendar Object and wants to guarantee that the Object will be created in all of the selected Calendar stores simultaneously. If the server cannot store to at least one of the selected Calendars, it must not store to any of them and must return a NO response indicating

that the command failed. The server may still return untagged responses indicating which Calendar stores failed.

New Calendar Objects added to a Calendar store must contain all required iCalendar properties. Updates to existing Calendar Objects need only contain the actual data to be updated. Duplicate attributes names are not allowed, whenever a value is given for a attribute name that already exists, the new value replaces the old value. If the new value is a null string (""), {0} or a CRLF immediately following the ":") then the old attribute is deleted.

In this first example, the users is creating a new Object in their own calendar. The operation succeeds:

```
C: A001 STORE 0 (+FLAGS \Repeating ICAL {102}
C: BEGIN: VCALENDAR
C: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
C: VERSION: 2.0
C: BEGIN:VTIMEZONE
C: TZID:America-California
C: LAST-MODIFIED:19870101T000000Z
C: TZURL:http://zones.stds_r_us.net/tz/America-New_York
C: BEGIN:STANDARD
C: DTSTART:19671029T020000
C: RRULE:FREQ=YEARLY;BYDAY=-1SU;BYMONTH=10
C: TZOFFSETFROM:-0700
C: TZOFFSETTO:-0800
C: TZNAME:EST
C: END:STANDARD
C: BEGIN:DAYLIGHT
C: DTSTART:19870405T020000
C: RRULE:FREQ=YEARLY;BYDAY=1SU;BYMONTH=4
C: TZOFFSETFROM:-0800
C: TZOFFSETTO:-0700
C: TZNAME:EDT
C: END:DAYLIGHT
C: END:VTIMEZONE
C: BEGIN: VEVENT
C: DTSTART;TZID=US-California: 19980606T140000
C: DTEND;TZID=US-California: 19980606T150000
C: SUMMARY: Meeting with Pete
C: END: VEVENT
C: END: VCALENDAR
C: )
S: A001 OK STORE completed
```

In the following example, the user has two default Calendar stores selected, one for <Alice>, one for <Bruce> and one for <Cindy>. The client attempts to schedule a meeting in all Calendars by using the STORE command. <Alice> refuses, <Bruce> requests a meeting invitation be sent and <Cindy> accepts. Note that the terms "accept"

and "refuse" to not imply an intervention on the part of a real person: whether the server accepts or refuses a STORE request should be based on access control.

```
C: A001 STORE 0 ICAL {102}
C: BEGIN: VCALENDAR
C: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
```

```
C: VERSION: 2.0
C: BEGIN: VEVENT
C: DTSTART: 19980606T140000
C: DTEND: 19980606T150000
C: SUMMARY: Meeting with Pete.
C: END: VEVENT
C: END: VCALENDAR
C:
S: * NO [REFUSED <Alice>] Alice doesn't like you.
S: * NO [MAILTO <Bruce> <Bruce@Iris.com>] Please send me a meeting
S: invitation.
S: A001 OK STORE completed
```

The following sequence modifies Calendar Object number 23 with a new start and end date/time.

```
C: A001 STORE 23 ICAL {102}
C: BEGIN: VCALENDAR
C: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
C: VERSION: 2.0
C: BEGIN: VEVENT
C: DTSTART: 19980806T140000Z
C: DTEND: 19980806T150000Z
C: END: VEVENT
C: END: VCALENDAR
C:
S: A001 OK STORE completed
```

6.7. COPY Command

6.8. MOVE Command

```
Arguments:      Calendar Object set
                Name of Calendar to move or copy to

Data:           None.

Result: OK - Calendar Objects moved.
                NO - Calendar Objects not moved.
                BAD - Bad date format given.
```

COPY and MOVE transfer a given set of Objects to a different Calendar store. In the case of the MOVE command, the Objects are removed from the current Calendar. MOVE must fail if the user does not have permission to remove an Object from the selected Calendar.

The Calendar store to move or copy to MUST exist before the operation is initiated.

Example:

```
C: A001 COPY 1 Section1
S: A001 OK COPY Completed.
```

6.9. UID Command

```
Arguments:      Command

Data:           Untagged responses: FETCH, SEARCH.

Result: OK - UID command completed.
                NO - UID command failed.
                BAD - Invalid command or UID given.
```

In its first form the UID command takes a COPY, MOVE, FETCH
or STORE command as its arguments. These commands are
processed as usual, except that unique ID's must be given instead of
Object numbers.

In the second form, the UID command takes a SEARCH command
with SEARCH command arguments. The interpretation of the
arguments is the same as with SEARCH; however, the numbers
returned in a SEARCH response for a UID SEARCH command are
unique identifiers instead of sequence numbers.

The number after the "*" in an untagged FETCH response is always
a sequence number, not a unique identifier, even for a UID command
response. However, server implementations MUST implicitly
include the UID data item as part of any FETCH response caused by
a UID command, regardless of whether UID was specified as a data
item to the FETCH.

Example:

```
C: A001 UID FETCH 1234123412341234 (FLAGS ICAL.REQUIRED)
S: * 12 FETCH (UID 1234123412341234 FLAGS  (\Deleted \Seen)
S: ICAL.REQUIRED {96}
S: BEGIN: VCALENDAR
S: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
S: VERSION: 2.0
S: BEGIN: VEVENT
S: DTBEGIN: 19980701T0300Z
S: DTEND: 19980701T0400Z
S: SUMMARY: Test meeting
S: END: VEVENT
S: END: VCALENDAR
S: )
```

S: A001 OK UID FETCH

6.10. SEARCH Command

Arguments: One or more search criteria

Data: Untagged responses: SEARCH.

Result: OK - search completed.
                NO - search error: can't search that criteria.
                BAD - command unknown or invalid arguments.

The SEARCH command searches the Calendar store for Objects that match the given searching criteria. Searching criteria consist of one or more search keys. The untagged SEARCH response from the server contains a listing of Object numbers corresponding to those Objects that match the searching criteria.

When multiple keys are specified, the result is the intersection (AND function) of all the messages that match those keys. A search key can also be a parenthesized list of one or more search keys (e.g. for use with the OR and NO keys).

<Object set> Objects with sequence numbers corresponding to the specified message sequence number set

ALL
All Objects in the mailbox; the default initial key for ANDing.

COMPONENT <component_name>
Objects which contain at least one of the specified Components. Valid Components are: VEVENT, VTODO, VJOURNAL, VALARM. Note that Objects which have a VALARM Component embedded anywhere within them MUST match this key.

DELETED
Objects with the \Deleted flag set.

NEW
Objects that have the \Recent flag set but not the \Seen flag. This is functionally equivalent to "(RECENT UNSEEN)".

NOT <search-key>
Objects that do not match the specified search key.

OR <search-key1> <search-key2>
Objects that match either search key.

RECENT
Objects that have the \Recent flag set.

SEEN
Objects that have the \Seen flag set.

TENTATIVE
Objects that have the \Tentative flag set.

UID <message set>
Objects with unique identifiers corresponding to the specified unique identifier set.

UNDELETED
Objects that do not have the \Deleted flag set.

UNSEEN
Objects that do not have the \Seen flag set.

ICAL <property name> <comparison operator> < property value>
Objects where the specified comparison operation is true.

The valid comparison operators are:

"="   Equals. True if the given property matches the given property value. String values are assumed to be case insensitive. Valid for all property types.

"contains"   Valid only for type TEXT, RFC822-ADDRESS and URL. True if the given value is a substring of the given property. String values are assumed to be case insensitive.

">"   Greater than. True if the given property is greater than the given property value. Valid only for the property types DATE, TIME, DATE-TIME, PERIOD, DURATION, INTEGER, FLOAT and UTC-OFFSET.

"<"   Less than. True if the given property is less than the given property value. Valid only for the property types DATE, TIME, DATE-TIME, PERIOD, DURATION, INTEGER, FLOAT and UTC-OFFSET.

The comparison value is assumed to have the same type as the specified Object property. The Object property type can be queried using the ATTRIBUTES command. If the property value specified in the comparison operation cannot be readily converted to the Object property type, the comparison operation is false.

Comparisons of DATE and DATE-TIME values are valid only under the following conditions. If the currently selected Calendar Store contains Object having different Time Zone information, as described in the section titled "Calendar Stores" above, then all comparison values for DATE and DATE-TIME types MUST be either UTC values or be convertible to UTC. Only Calendar Stores

whose objects fall within the same Time Zone information may perform comparisons based on local time values.

DATE-TIME values which contain only a date component (e.g. 19970106Z) MUST match any DATE-TIME which falls on the given date when used with the "=" operator.

If multiple properties with the same property name exist within the same Object, the comparison operator is true if ANY of the properties meet the specified criteria. If multiple properties with different property types exist within an object, the result of the SEARCH operation is undefined.

If a COMPONENT key is specified, a property name given in an ICAL search request is assumed to refer to a field within any specified COMPONENT.

Examples:

```
C: A001 SEARCH 1:20 ICAL DUE = 19981001
S: * SEARCH 4 9 12 19
S: A001 OK SEARCH Completed.

C: A001 SEARCH UNSEEN ICAL PRIORITY > 3
S: * SEARCH 3 12 45
S: A001 OK SEARCH Competed.

C: A001 SEARCH ICAL PRIORITY > 3 ICAL DUE < 19981112Z
S: * SEARCH 3 45
S: A001 OK SEARCH Competed.

C: A001 SEARCH COMPONENT VALARM ICAL DTSTART =
19980808Z
S: * SEARCH 14 18 19
S: A001 OK SEARCH Competed.
```

The above SEARCH operation finds all of the Objects which contain an alarm set to go off on August 8th of 1998 in the Pacific Time Zone.

7. Server Responses

Server responses are in three forms: status responses, server data, and command continuation request.

The client MUST be prepared to accept any response at all times.

Status responses that are tagged indicate the completion result of a client command, and have a tag matching the command.

Some status responses, and all server data, are untagged. An

untagged response is indicated by the token "*" instead of a tag. Untagged status responses indicate server greeting, or server status that does not indicate the completion of a command. For historical reasons, untagged server data responses are also called "unsolicited data", although strictly speaking only unilateral server data is truly "unsolicited".

Certain server data MUST be recorded by the client when it is received; this is noted in the description of that data. Such data conveys critical information which affects the interpretation of all subsequent commands and responses (e.g. updates reflecting the creation or destruction of Calendar Objects).

Other server data SHOULD be recorded for later reference; if the client does not need to record the data, or if recording the data has no obvious purpose (e.g. a SEARCH response when no SEARCH command is in progress), the data SHOULD be ignored.

Command continuation request responses use the token "+" instead of a tag. These responses are sent by the server to indicate acceptance of an incomplete client command and readiness for the remainder of the command.

7.1. Server Responses - Status Responses

Status responses MAY include an OPTIONAL response code. A response code consists of data inside square brackets in the form of an atom, possibly followed by a space and arguments. The response code contains additional information or status codes for client software beyond the OK/NO/BAD condition, and are defined when there is a specific action that a client can take based upon the additional information.

The currently defined response codes are:

ALERT - The human-readable text contains a special alert that MUST be presented to the user in a fashion that calls the user's attention to the message.

PERMANENTFLAGS - Followed by a parenthesized list of flags, indicates which of the known flags that the client can change permanently. Any flags that are in the FLAGS untagged response, but not the PERMANENTFLAGS list, can not be set permanently. If the client attempts to STORE a flag that is not in the PERMANENTFLAGS list, the server will either reject it with a NO reply or store the state for the remainder of the current session only.

MAILTO - Returned from a STORE or APPEND command to indicate that an specific user requests Meeting Invitations to be sent to them via SMTP mail. Returned with NO response only.

READ-ONLY - The Calendar is selected read-only, or its access while selected has changed from read-write to read-only.

READ-WRITE - The Calendar is selected read-write, or its access while selected has changed from read-only to read-write.

REFUSED - Returned from a STORE or APPEND command to indicate that an specific user does not allow scheduling requests from other users. Returned with NO response only.

SENT - Returned from a STORE or APPEND command to indicate that a meeting invitation was sent by the server via e-mail rather than creating an Object in a user's Calendar.

TENTATIVE - Returned from a STORE or APPEND command to indicate that a Calendar Object marked as \Tentative was created in the specified users Calendar.

Additional response codes defined by particular client or server implementations SHOULD be prefixed with an "X-" until they are added to a revision of this protocol. Client implementations SHOULD ignore response codes that they do not recognize.

7.1.1. OK Response
7.1.2. NO Response
7.1.3. BAD Response

Data: Optional response code
Optional human-readable text.

The OK, NO and BAD responses are intended to give the client information about a command's completion status or information about the operation of the server. When given in a tagged response, these responses indicates completion of the command with the associated tag. Untagged responses of this kind are always informational messages. In both cases, a message based on the response code and text MAY be presented to the user.

The untagged form is also used as one of three possible greetings (along with PREAUTH and BYE) at session startup. It indicates that the session is not yet authenticated and that a LOGIN command is needed.

Examples of the OK, NO and BAD response can be found within many of the examples given above.

7.1.4. PREAUTH Response

Data: Optional response code
Human-readable text.

The PREAUTH response is always untagged, and is one of three possible greetings (along with OK and BYE) at session startup. It indicates that the session has already been authenticated by external means and thus no LOGIN command is needed.

Example:

```
S: * PREAUTH ICAP server logged in as Smith
```

7.1.5. BYE Response

Data: Optional response code

Human-readable text.

The BYE response is always untagged, and indicates that the server is about to close the connection. The human-readable text MAY be displayed to the user in a status report by the client. The BYE response is sent under one of four conditions:

1) as part of a normal logout sequence. The server will close the connection after sending the tagged OK response to the LOGOUT command.

2) as a panic shutdown announcement. The server closes the connection immediately.

3) as an announcement of an inactivity autologout. The server closes the connection immediately.

4) as one of three possible greetings at session startup (along with OK and PREAUTH), indicating that the server is not willing to accept a session from this client. The server closes the connection immediately.

The difference between a BYE that occurs as part of a normal LOGOUT sequence (the first case) and a BYE that occurs because of a failure (the other three cases) is that the connection closes immediately in the failure case.

Example:

```
S: * BYE Autologout; idle for too long
```

7.2. Server Responses - Data Responses

These responses are always untagged. This is how server data are transmitted from the server to the client, often as a result of a command with the same name.

7.2.1. CAPABILITY Response

Data: capability listing

The CAPABILITY response occurs as a result of a CAPABILITY command. The capability listing contains a space-separated listing of capability names that the server supports. The capability listing MUST include the atom "ICAP".

A capability name which begins with "AUTH=" indicates that the server supports that particular authentication mechanism.

Other capability names indicate that the server supports an extension, revision, or amendment to the ICAP protocol. Server responses MUST conform to this document until the client issues a command that uses the associated capability.

Capabilities not defined in this document MUST be prefixed with "X-".

Client implementations SHOULD NOT require any capability name other than "ICAP", and MUST ignore any unknown capability names.

Example:

```
S: * CAPABILITY ICAP AUTH=KERBEROS_V4 X-PocketToaster
```

7.2.2. LIST Response

Data: Name attributes list
Calendar name or hierarchical name, possible prefixed by an owner name

The LIST response is given in response to a LIST command.

Four name attributes are defined:

\Noinferiors   It is not possible for any child levels of hierarchy to exist under this name; no child levels exist now and none can be created in the future.

\Noselect   It is not possible to use this name as a selectable Calendar.

\Marked   The Calendar has been marked "interesting" by the server; the Calendar probably contains Objects that have been added since the last time the Calendar was selected.

\Unmarked   The Calendar does not contain any additional Objects since the last time the Calendar was selected.

If it is not feasible for the server to determine whether the Calendar is "interesting" or not, or if the name is a \Noselect name, the server SHOULD NOT send either \Marked or \Unmarked.

The hierarchy delimiter "/" is used to delimit levels of hierarchy in a

Calendar name. A client can use it to create child Calendars, and to search higher or lower levels of naming hierarchy.

The name represents an unambiguous left-to-right hierarchy, and MUST be valid for use as a reference in LIST and LSUB commands. Unless \Noselect is indicated, the name MUST also be valid as an argument for commands, such as SELECT, that accept Calendar names.

Example:

```
S: * LIST () Section1
S: * LIST () <Ann>
S: * LIST () <Bob>
S: * LIST (\Noselect) Projects/
S: * LIST () Projects/Project_Purple
S: * LIST () "Project 1/From Bill"
```

7.2.3. LSUB Response

The LSUB command is identical to the LIST command except that it is given in response to the LSUB command.

Example:

```
S: * LSUB () Section1
S: * LSUB() Projects/Project_Purple
S: * LSUB() "Project 1/From Bill"
```

7.2.4. EXISTS Response

Data: None.

The EXISTS response reports the number of Objects in the Calendar. This response occurs as a result of a SELECT, EXAMINE or RANGE command, and if the size of the Calendar changes (e.g. new Calendar Objects).

The update from the EXISTS response MUST be recorded by the client.

Example:

```
S: * 23 EXISTS
```

7.2.5. RECENT Response

Data: None.

The RECENT response reports the number of Calendar Objects that have been posted since the previous time a SELECT command was done on this Calendar. This response occurs as a result of a SELECT or EXAMINE command, and if the size of the Calendar changes (e.g. new Calendar Objects).

The update from the RECENT response MUST be recorded by the client.

Example:

```
S: * 1 RECENT
```

7.2.6. EXPUNGE Response

Data: None.

The EXPUNGE response reports that the specified Calendar Object sequence number has been permanently removed from the Calendar. The sequence number for each successive Object in the Calendar is immediately decremented by 1, and this decrement is reflected in sequence numbers in subsequent responses (including other untagged EXPUNGE responses).

As a result of the immediate decrement rule, sequence numbers that appear in a set of successive EXPUNGE responses depend upon whether the Objects are removed starting from lower numbers to higher numbers, or from higher numbers to lower numbers. For example, if the last 5 Objects in a 9-Object Calendar are expunged; a "lower to higher" server will send five untagged EXPUNGE responses for sequence number 5, whereas a "higher to lower server" will send successive untagged EXPUNGE responses for sequence numbers 9, 8, 7, 6, and 5.

An EXPUNGE response MUST NOT be sent when no command is in progress; nor while responding to a FETCH, STORE, or SEARCH command. This rule is necessary to prevent a loss of synchronization of sequence numbers between client and server.

The update from the EXPUNGE response MUST be recorded by the client.

Example:

```
S: * 1 EXPUNGE
```

7.2.7. FETCH Response

Data: Calendar Object data.

The FETCH response returns data about a Calendar Object to the client. The data are pairs of data item names and their values in parentheses. This response occurs as the result of a FETCH, STORE, ATTRIBUTES and FREEBUSY commands, as well as by unilateral server decision (e.g. flag updates). See the description of the commands for a list of data items.

```
S: * 1 FETCH ICAL.REQUIRED {96}
S: BEGIN: VCALENDAR
S: PRODID:-//Amplitude Software//NONSGML EventCenter v1.5//EN
S: VERSION: 2.0
S: BEGIN: VEVENT
S: DTBEGIN: 19980701T0300Z
S: DTEND: 19980701T0400Z
S: SUMMARY: Test meeting
S: END: VEVENT
S: END: VCALENDAR
S: )
```

7.2.8. FLAGS Response

Data: Calendar Object flags.

The FLAGS response occurs as a result of a SELECT or EXAMINE command. The flag parenthesized list identifies the flags (at a minimum, the system-defined flags) that are applicable for this Calendar. Flags other than the system flags can also exist, depending on server implementation.

The update from the FLAGS response MUST be recorded by the client.

Example:

```
S: * 1 FLAGS (\Deleted)
```

7.2.9. SEARCH Response

Data: zero or more numbers

The SEARCH response occurs as a result of a SEARCH or UID SEARCH command. The number(s) refer to those Objects that match the search criteria. For SEARCH, these are sequence numbers; for UID SEARCH, these are unique identifiers. Each number is delimited by a space.

Example:

```
S: * SEARCH 2 3 6
```

7.3.   Server Responses - Command Continuation Request

The command completion request response is indicated by a "+" token instead of a tag. This form of response indicates that the server is ready to accept the continuation of a command from the client. The remainder of this response is a line of text.

This response is used in the AUTHORIZATION command to transmit server data to the client, and request additional client data. This response is also used if an argument to any command is a literal.

The client is not permitted to send the octets of the literal unless the server indicates that it expects it. This permits the server to process commands and reject errors on a line-by-line basis. The remainder of the command, including the CRLF that terminates a command, follows the octets of the literal. If there are any additional command arguments the literal octets are followed by a space and those arguments.

Example:

```
C: A001 LOGIN {11}
S: + Ready for additional command text
C: FRED FOOBAR {7}
S: + Ready for additional command text
C: fat man
S: A001 OK LOGIN completed
```

8. Formal Syntax

This will be included in a later draft of this specification.

9. Example Sessions

Here is an example of a client using guest access to query the freetime of three individuals:

```
S: * OK ICAP Server ready.
C: A001 LOGIN anonymous pete@amplitude.com
S: A001 OK LOGIN Anonymous access granted.
C: A002 EXAMINE <Liz>
S: * 12 EXISTS
S: A002 OK EXAMINE
C: A003 EXAMINE <Bob>
S: * 22 EXISTS
S: A003 OK EXAMINE
C: A004 EXAMINE <Purna>
```

```
S: * 36 EXISTS
S: A004 OK EXAMINE
C: A005 FREEBUSY "" 19980701T0300Z 19980701T1900Z
S: * 1 FETCH ICAL {200}
S: BEGIN: VCALENDAR
S: PRODID:-//hacksw/handcal//NONSGML v1.6//EN
S: VERSION: 2.0
S: BEGIN: VFREEBUSY
S: ATTENDEE: Liz
S: DTBEGIN: 19980701T0300Z
S: DTEND: 19980701T1900Z
C: FREEBUSY; VALUE=PERIOD-START: 19980701T0800Z/PT3H,
C: 19980701T1500Z/PT2H
S: END: VFREEBUSY
S: BEGIN: VFREEBUSY
S: ATTENDEE: Bob
S: DTBEGIN: 19980701T0300Z
S: DTEND: 19980701T1900Z
C: FREEBUSY; VALUE=PERIOD-START: 19980701T1000Z/PT1H,
C: 19980701T1500Z/PT1H
S: END: VFREEBUSY
S: BEGIN: VFREEBUSY
S: ATTENDEE: Purna
S: DTBEGIN: 19980701T0300Z
S: DTEND: 19980701T1900Z
C: FREEBUSY; VALUE=PERIOD-START: 19980701T0900/PT2H
S: END: VFREEBUSY
S: END: VCALENDAR
S: A005 OK FREEBUSY
C: A006 LOGOUT
S: * BYE ICAP Server terminating connection.
S: A006 OK LOGOUT
```

10. Open Issues/Work in Progress

How should a user's Calendar server be located? For example, given a mail address like <pete@amplitude.com> how should a client locate the Calendar server. This is still not clearly defined.

11. Changes From Previous Draft Version

1. The section on the Dates data type has been updated to reflect changes to the iCalendar definition [ICAL] to better support timezone information.
2. A section to explain the use of periods of time has been added.
3. An optional date range has been added to the SELECT comand.

4. Examples that included iCalendar objects have been updated to reflect changes to the iCalendar definition [ICAL] with reguard to timeszones, dates, the summary field (previously the description field) and the now mandatory product id field.
5. The RANGE command no longer supports wildcard characters.
6. The FETCH command has been modified and additional description has been added.

12. References

[RFC 1730] Crispin, M. "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4", Dec 1994,

[ICAL] Dawson, F., Stenerson, D.,"Internet Calendaring and Scheduling Core Object Specification (iCalendar)", 05/11/1998, http://www.imc.org/draft-ietf-calsch-ical-main

[RFC 1521] Borenstein, N., and Freed, N., "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," Bellcore, Innosoft, September 1993.

[RFC 821] Postel, Jonathan B. "Simple Mail Transfer Protocol," STD 10, USC/Information Sciences Institute, August 1982.

[RFC 1731] Myers, J., "IMAP4 Authentication Mechanism,"
Carnegie-Mellon University, December 1994.

[ISO-TIME] Kuhn, M., "A Summary of the International Standard
Date and Time Notation",
http://www.ft.uni-erlangen.de/~mskuhn/iso-time.html

[ITIP] Silverberg, S., Mansour, S., Dawson, F., Hopson, R.,"
iCalendar Transport-Independent Interoperability Protocol",
http://www.imc.org/draft-ietf-calsch-itip-part1,
http://www.imc.org/draft-ietf-calsch-itip-part2,
http://www.imc.org/draft-ietf-calsch-itip-part3

13. Security Considerations

The user name and password arguments of the LOGIN command are
sent in clear text over most transport protocols. Consult [RFC 1731]
for a discussion of authentication mechanisms used by IMAP4 and
by ICAP.

Servers should implement and enforce access control mechanisms
for Calendar stores. This specification contains no provisions for
defining and maintaining access control.

14. Author's Notes

This spec is based in very large part on the operation, commands and
concepts of IMAP4 [RFC 1730]. In the spirit of "not reinventing the
wheel" I have incorporated parts of the IMAP4 specification into this
work. My thanks to the authors of the IMAP4 specification for their
excellent work.

Author's Address:

Peter O'Leary
Amplitude Software Corp
185 Berry Street
San Francisco, CA 94107
http://www.amplitude.com/
(415) 659-3500
E-mail: mailto:pete@amplitude.com