Network Working Group
Internet-Draft
Expires: December 29, 2002

D. Royer
INET-Consulting
G. Babics
Steltor
P. Hill
MIT
S. Mansour
AOL/Netscape

June 30, 2002

Calendar Access Protocol (CAP)
draft-ietf-calsch-cap-08

Status of this Memo

Copyright Notice

Abstract

The Calendar Access Protocol (CAP) is an Internet protocol described in this memo that permits a Calendar User (CU) to utilize a Calendar User Agent (CUA) to access an [iCAL] based Calendar Store (CS).

The CAP definition is based on requirements identified by the Internet Engineering Task Force (IETF) Calendaring and Scheduling

(CALSCH) Working Group. More information about the IETF CALSCH Working Group activities can be found on the IMC web site at http://www.imc.org/ietf-calendar and at the IETF web site at http://www.ietf.org/html.charters/calsch-charter.html [1]. Refer to the references within this memo for further information on how to access these various documents.

Table of Contents

1. Introduction

This document specifies how a Calendar CUA interacts with a CS to manage calendar information. In particular, it specifies how to query, create, modify, and delete iCalendar components (e.g., events, to-dos, or daily journal entries). It further specifies how to search for available busy time information. Synchronization with CUAs is not covered.

CAP is specified as a BEEP "profile". As such, many aspects of the protocol (e.g., authentication and privacy) are provided within [BEEP]. The protocol data units leverage the standard iCalendar format [iCAL] to convey calendar related information.

CAP can also be used to store and fetch [iTIP] objects and when those objects are used in this memo, they mean exactly the same as defined in [iTIP]. When iCalendar objects are transfered between the calendar user agent and a calendar server, some additional properties and parameters may be added and the calendar user agent is responsible for correctly generating iCalendar objects to non CAP processes.

The definition of new components, properties, parameter's, and value types are broken into two parts. The first part summarizes and defined the new objects. The second part provides the detail and any ABNF for those objects.

1.1 Formatting Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Calendaring and scheduling roles are referred to in quoted-strings of text with the first character of each word in upper case. For example, "Organizer" refers to a role of a "Calendar User" (CU) within the protocol defined by [iTIP]. Calendar components defined by [iCAL] are referred to with capitalized, quoted-strings of text. All iCalendar components should start with the letter "V". For example, "VEVENT" refers to the event calendar component, "VTODO" refers to the to-do component and "VJOURNAL" refers to the daily journal component.

Scheduling methods defined by [iTIP], are referred to with capitalized, quoted-strings of text. For example, "REPLY" refers to the method for replying to a "REQUEST".

CAP commands are referred to by upper-case, quoted-strings of text,

followed by the word "command". For example, "CREATE" command refers to the command for creating a calendar entry, "SEARCH" command refers to the command for reading calendar components. CAP Commands are named using the "CMD" property.

Properties defined by this memo are referred to with capitalized, quoted-strings of text, followed by the word "property". For example, "ATTENDEE" property refers to the iCalendar property used to convey the calendar address that has been invited to a "VEVENT" or "VTODO" component.

Property parameters defined by this memo are referred to with capitalized, quoted-strings of text, followed by the word "parameter". For example, "PARTSTAT" parameter refers to the iCalendar property parameter used to specify the participation status of an attendee. Enumerated values defined by this memo are referred to with capitalized text, either alone or followed by the word "value".

In tables, the quoted-string text is specified without quotes in order to minimize the table length.

1.2 Related Documents

Implementers will need to be familiar with several other memos that, along with this one, describe the Internet calendaring and scheduling standards. These documents are:

[iCAL] - (RFC2445) Which specifies the objects, data types, properties and property parameters used in the protocols, along with the methods for representing and encoding them.

[iTIP] - (RFC2446) Which specifies an interoperability protocol for scheduling between different installations.

[iMIP] - (RFC2447) Which specifies the Internet email binding for [iTIP].

[GUIDE] - (draft/rfc...), a guide to implementers and describes the elements of a calendaring system, how they interact with each other, how they interact with end users, and how the standards and protocols are used.

This memo does not attempt to repeat the specification of concepts and definitions from these other memos. Where possible, references are made to the memo that provides for the specification of these concepts and definitions.

1.3 Definitions

BOOKED - An entry in the calendar store has one of three conceptual states. It is "UNPROCESSED", "BOOKED" or marked as "DELETED". How the implementation stores the state of any object is not a protocol issues and is not discussed. An object can be said to be booked, unprocessed, or marked for delete.

1. A "UNPROCESSED" scheduling entry has been stored in the calendar store but has not been acted on by a Calendar User (CU) or Calendar User Agent (CUA). All scheduled entries are [iTIP] objects. All [iTIP] objects in the store are not booked. To retrieve any [iTIP] object, simply do a query asking for any objects that were stored with its state set to "UNPROCESSED".

2. A booked entry is stored with the CREATE command. It is an entry that has been acted on by a CU or CUA and there has been a decision to store an object. To retrieve any booked object, simply do a query asking for any objects that were stored with its state set to "BOOKED".

3. A marked for delete component has its state set to DELETE. To retrieve any deleted object, simply do a query asking for any objects that were stored with its state set to "DELETED". By default objects marked for delete are not returned. The CUA must specifically ask for marked for delete objects.

Calendar - A collection of logically related objects or entities each of which may be associated with a calendar date and possibly time of day. These entities can include calendar properties or components. In addition, a calendar might be related to other calendars with the "RELATED-TO" property. A calendar is identified by its unique calendar identifier. The [iCAL] defines the initial calendar properties, calendar components and properties that make up the contents of a calendar.

Calendar Access Protocol (CAP) - The standard Internet protocol that permits a CUA to access and manipulate calendars residing on a Calendar Store. (this memo)

Calendar Access Rights (VCAR) - The mechanism for specifying the CAP operations ("PERMISSION") that a particular calendar user ("UPN") is granted or denied permission to perform on a given calendar object ("SCOPE"). The calendar access rights are specified with the "VCAR" calendar components within a CS and calendar.

Calendar Address - Also See Calendar URL - they are one in the same for CAP addresses. The calendar address can also be the value to the "ATTENDEE" and "ORGANIZER" properties as defined in [iCAL].

Calendar URL - A calendar URL is a URL defined in this memo that specifies the address of a CS or Calendar.

Component- Any object that conforms to the iCalendar object format and that is either defined in an internet draft, registered with IANA, or is an experimental object that is prefixed with "x-". Some types of components include calendars, events, to-dos, journals, alarms, and time zones. A component consists of properties and possibly other contained components. For example, an event may contain an alarm component.

Properties - An attribute of a particular component. Some properties are applicable to different types of components. For example, the "DTSTART" property is applicable to "VEVENT", "VTODO", "VJOURNAL" components. Other components are applicable only to an individual type of calendar component. For example, the "TZURL" property may only applicable to "VTIMEZONE" components.

Calendar Identifier (CalID) - A globally unique identifier associated with a calendar. Calendars reside within a CS. See Qualified Calendar Identifier and Relative Calendar Identifier. All CalIDs start with "cap:".

Calendar Policy - A CAP operational restriction on the access or manipulation of a calendar. These may be outside of the scope of the CAP protocol. An example of an implementation or site policy is, "events MUST BE scheduled in unit intervals of one hour".

Calendar Property - An attribute of a calendar ("VAGENDA"). The attribute applies to the calendar, as a whole. For example, the "CALSCALE" property specifies the calendar scale (e.g., the "GREGORIAN" value) for the whole calendar.

Calendar Server - An implementation of a Calendar Store that manages one or more calendars.

Calendar Store (CS) - The data and service model definition for a Calendar Store as defined in this memo. This memo does not specify how the CS is implemented.

Calendar Store Identifier (CSID) - The globally unique identifier for an individual CS. A CSID consists of the host and port portions of a "Common Internet Scheme Syntax" part of a URL, as

defined by [RFC1738]. The CSID excludes any reference to a specific calendar.

Calendar Store Components - Components maintained in a CS specify a grouping of calendar store-wide information.

Calendar Store Properties - Properties maintained in a Calendar Store calendar store-wide information.

Calendar User (CU) - An entity (often biological) that uses a calendaring system.

Calendar User Agent (CUA) - The client application that a CU utilizes to access and manipulate a calendar.

CAP Session - An open communication channel between a CUA and a Calendar Server. If the CAP session is authenticated, the the CU is "authenticated" and it is an "authenticated CAP session".

Contained Component / Contained Properties - A component or property that is contained inside of another component. A "VALARM" component for example may be contained inside of a "VEVENT" component. And a "TRIGGER" property could be a contained property of a "VALARM" component.

Delegate - A calendar user (sometimes called the delegatee) who has been assigned participation in a scheduled component (e.g., VEVENT) by one of the attendees in the scheduled component (sometimes called the delegator). An example of a delegate is a team member told to go to a particular meeting in place of another Attendee who is unable to attend.

Designate - A calendar user who is authorized to act on behalf of another calendar user. An example of a designate is an assistant.

Experiential - The CUA and CS may implement experimental extensions to the protocol. They also might have experimental components, properties, and parameters. These extensions MUST start with "x-" (or "X-") and should include a vendor prefix (such as "x-myvendor-"). There is no guarantee that these experimental extensions will interoperate with other implementations. There is no guarantee that they will not interact in unpredictable ways with other vendor experimental extensions. Implementations should limit sending those extensions to other implementations.

Object - A generic name for any component, property, parameter, or value type to be used in iCalendar.

Overlapped Booking - A policy which indicates whether or not components with a "TRANSP" property not set to "TRANSPARENT-NOCONFLICT" or "OPAQUE-NOCONFLICT" value can overlap one another. When the policy is applied to a calendar it indicates whether or not the time span of any component (VEVENT, VTODO, ...) in the calendar can overlap the time span of any other component in the same calendar. When applied to an individual entry, it indicates whether or not any other component's time span can overlap that individual component. If the CS does not allow overlapped booking, then the CS is unwilling to allow any overlapped bookings within any calendar in the CS.

Owner - One or more CUs or UGs that are listed in the "OWNER" property in a calendar. There can be more than one owner. The "

Qualified Calendar Identifier (Qualified CalID) - A CalID in which both the scheme and csid of the CAP URI are present.

Realm - A collection of calendar user accounts, identified by a string. The name of the Realm is only used in UPNs. In order to avoid namespace conflict, the Realm SHOULD be postfixed with an appropriate DNS domain name. (e.g., the foobar Realm could be called foobar.example.com).

Relative Calendar Identifier (Relative CalID) - An identifier for an individual calendar in a calendar store. It MUST BE unique within a calendar store. A Relative CalID consists of the "URL path" of the "Common Internet Scheme Syntax" portion of a URL, as defined by [RFC396] and [RFC2718].

Session Identity - A UPN associated with a CAP session. A session gains an identity after successful authentication. The identity is used in combination with VCAR to determine access to data in the CS.

User Group (UG) - A collection of Calendar Users and/or User Groups. These groups are expanded by the CS and may reside either locally or in an external database or directory. The group membership may be fixed or dynamic over time.

Username - A name which denotes a Calendar User within a Realm. This is part of a UPN.

User Principal Name (UPN) - A unique identifier that denotes a CU or a group of CU. A UPN is a RFC 822 compliant email address, with exceptions listed below, and in most cases it is deliverable to the CU. In some cases it is identical to the CU's well known email address. A CU's UPN MUST never be an e-mail address that is

deliverable to a different person as there is no requirement that a person's UPN MUST BE their e-mail address. A UPN is formatted as a user name followed by "@" followed by a Realm in the form of a valid, and unique, DNS domain name. The user name MUST BE unique within the Realm. In it's simplest form it looks like "user@example.com".

In certain cases a UPN will not be RFC 822 compliant. When anonymous authentication is used, or anonymous authorization is being defined, the special UPN "@" will be used. When authentication MUST BE used, but unique identity MUST BE obscured, a UPN of the form @DNS-domain-name may be used. For example, "@example.com".

2. Additions to iCalendar

Several new components, properties, parameters, and value types are added in CAP. This section summarizes those new objects.

This memo extends the properties that can go into 'calprops' as defined in [iCAL] section 4.6 page 51 to allow iTIP objects transmitted between a CAP aware CUA and the CS to contain the "TARGET" and "CMD" properties. This memo does not address how a CUA transmits iTIP or iMIP objects to non CAP programs.

```
calprops   = 2*(

           ; 'prodid' and 'version' are both REQUIRED,
           ; but MUST NOT occur more than once.
           ;
           prodid /version /

           ; These are optional, but MUST NOT occur
           ; more than once.
           ;
           calscale        /
           method          /
           target          /
           iana-prop       /
           cmd             /

           ; These are optional, and may occur more
           ; than once.
           ;
           x-prop
```

In addition a problem exists with the control of "VALARM" components and their "TRIGGER" properties. A CU may wish to set their own alarm (local alarms) on components. These local alarms are not to be forwarded to other CUs, CUAs, or CSs as are the "SEQUENCE" property and the "ENABLE" parameter. So for the protocol between a CUA and a CS, the following changes apply to the CAP protocol from [iCAL] section "4.6.6" page 67:

```
alarmc     = "BEGIN" ":" "VALARM" CRLF
             alarm-seq
             iana-prop
             (audioprop / dispprop / emailprop / procprop)
             "END" ":" "VALARM" CRLF

alarm-seq  = "SEQUENCE" alarmseqparam ":" integer CRLF

alarmseqparam = *( ";" xparam)
              / ";" local-param
```

The CUA adds a "SEQUENCE" property to each "VALARM" component as it books the component. This property along with the "LOCAL" and "ENABLE" parameters allow the CUA to uniquely identify any VALARM in any component. The CUA should remove those before forwarding to non CAP aware CUAs (including iMIP CUAs).

In addition, if a CUA wished to ignore a "TRIGGER" property in a "VALARM" that was supplied to it by the ORGANIZER, the CUA needs a common way to tag that trigger as disabled. So for the protocol between a CUA and a CS, the following is a modification to [iCAL] section "4.8.6.3" page 127:

```
trigger    = "TRIGGER" 1*(";" enable-param) (trigrel / trigabs)
```

Section 6.2.1 and Section 6.2.2.

These additions will be transmitted between a CS and a CAP aware CUA. So the VERSION value will remain at "2.0" as no existing iTIP or iMIP implementation will be effected.

2.1 New Value Types (summary)

UPN The UPN value type is text value type restricted to only UPN values. (Section 6.1.2)

UPN-FILTER Like the UPN value type, but also includes filter rules that allow wildcards. (Section 6.1.3)

CALQUERY The "CAL-QUERY" (Section 6.1.1) value type is a query syntax that is used by the CUA to specify the rules that apply to a CAP command. In the case of "SEARCH", the query language is used to fetch objects from the CS. When used with "DELETE", the selected objects are deleted from the CS. "CAL-QUERY" can also be used with "MOVE" and "MODIFY".

2.1.1 New Parameters (summary)

ENABLE -

The "ENABLE" parameter in CAP is used to tag a "TRIGGER" property in a component as disabled or enabled. This is used when a scheduling request arrives and the CU wishes to ignore the trigger time included. (Section 6.2.1).

Formal Definition: The "ENABLE" parameter is defined by the following notation:

```
enable-param  = "ENABLE" "=" ("TRUE" / "FALSE")
```

LOCAL -

The "LOCAL" parameter in CAP is used to tag a "SEQUENCE" property in a "VALARM" to signify that a VALARM is local or to be distributed. (Section 6.2.2).

For example, when inviting others to an event, the ORGANIZERs booked VEVENT might contain VALARMs, and those VALARMS might be 'alarm be 5 minutes before the meeting'. However other ATTENDEEs, may have to set their own VALARMs for the same event (assuming they reply that they will be attending). So, by tagging the VALARM as local the CUA MUST never forward those local VALARMs to other CS's or CUAs.

The CUA can not simply delete any VALARMs from components where the CU is not the ORGANIZER. If it did, any [iTIP] "COUNTER" would result in the ORGANIZER thinking that the ATTENDEE wished to also counter with removing those VALARMs. And in addition, any update to an existing component would re-create those VALARMs in the ATTENDEEs CS.

Formal Definition: The "LOCAL" parameter is defined by the following notation:

```
local-param  = "LOCAL" "=" ("TRUE" / "FALSE")
```

2.1.2 New Properties (summary)

ALLOW-CONFLICT - Some entries in a calendar might not be valid if other entries were to allowed to overlap the same time span. Renting a car for example. It would not make sense to allow two reservations for the same car at the same time. The "ALLOW-CONFLICT" property takes a boolean value. If FALSE, then conflicts are not allowed. (Section 7.1)

CSID - Each CS needs its own unique identifier. The "CSID" property is the official unique identifier for the CS. If the BEEP 'serverName' attribute was suppled in the BEEP 'start' message, then the CSID will be mapped to the virtual host name supplied and the host name part of the CSID MUST BE the same as the 'serverName' value. This allows one CS implementation to service multiple virtual hosts. CS's are not required to support virtual hosting. If a CS does not support virtual hosting then it must ignore the BEEP 'serverName'. (Section 7.5)

CALID - Each calendar within a CS needs to be uniquely identifiable. The "CALID" property identifies a unique calendar within a CS. It can be a full CALID or a relative CALID. (Section 7.2)

CALMASTER - The "CALMASTER" property specifies the contact information for the CS. (Section 7.3)

CARID - Access rights can be saved and fetched by unique ID - the "CARID". (Section 7.4)

CMD - The enumerated list of CAP commands and the options for those commands, as well as replies are transmitted using the "CMD" property. (Section 9.1)

DECREED - Some access rights are not changeable by the CUA. When that is the case, the "DECREED" property value in the "VCAR" will be TRUE. (Section 7.6)

DEFAULT-CHARSET - The list of charsets supported by the CS. The first entry MUST BE the default for the CS. (Section 7.7)

DEFAULT-LOCALE - The list of locales supported by the CS. The first entry in the list is the default locale. (Section 7.8)

DEFAULT-TZID - This is the list of known timezones supported. The first entry is the default. (Section 7.9)

DEFAULT-VCARS - A list of the CARIDs that will be used to create new calendars. (Section 7.10)

DENY - The UPNs listed in the "DENY" property of a "VCAR" will denied access as described in the "VRIGHT" component. (Section 7.11)

EXPAND - This property tells the CS if the query reply should expand components into multiple instances. The default is FALSE. (Section 7.12)

GRANT - The UPNs listed in the "GRANT" property of a "VCAR" will allowed access as described in the "VRIGHT" component. (Section 7.13)

MAXDATE - The maximum date supported by the CS. (Section 7.14)

MINDATE - The minimum date supported by the CS. (Section 7.15)

NAME - Several storeable components such as "VCAR" and "VQUERY" may have the "NAME" property contained in them to describe in a various locals the purpose of the component. Components may have multiple "NAME" properties. (Section 7.16)

OWNER - Each calendar has at least one "OWNER". (xref target="OWNER"/>) Related to the "OWNER()" (Section 6.1.3) query clause.

PERMISSION - This property specifies the permission being granted or denied. Examples are "READ" and "MODIFY". (Section 7.18)

QUERY - Use to hold the CAL-QUERY (Section 7.19) for the component.

QUERYID - A unique id for a stored query. (Section 7.20)

REQUEST-STATUS - The [iCAL] "REQUEST-STATUS" property is extended to include new error numbers. (Section 7.21)

RESTRICTION - In the final check when granting calendar access requests, the CS test the results to the value of the "RESTRICTION" property in the corresponding "VRIGHT" component to determine if the access meets that restriction. (Section 7.22)

SCOPE - The "SCOPE" property is used in "VRIGHT"s component to select the subset of data that may be acted upon when checking access rights. (Section 7.23)

TARGET - The new VCALENDAR property "TARGET" (Section 7.24) used to specify which calendar(s) will be the subject of the CAP command.

TRANSP - This is a modification the the [iCAL] TRANSP property and

it allows more values. (Section 7.25)

2.1.3 New Components (summary)

VAGENDA - CAP allows the fetching and storing of the entire contents of a calendar. The "VCALENDAR" object is not sufficient to encapsulate all of the needed data that describes a calendar. The VAGENDA object is the encapsulating object for an entire calendar. (Section 8.1)

VCALSTORE - Each CS contains one or more calendars (VAGENDAs), the VCALSTORE object is the encapsulating object that can hold all of the "VAGENDA"s along with any components and properties that are unique to the store level. (Section 8.2)

VCAR - Calendar Access Rights are specified and encapsulated in the new iCalendar "VCAR" (Section 8.3) component. The "VCAR" component holds some new properties and at least one "VRIGHT" component.

VRIGHT - (Section 8.4) This component encapsulates a set of instructions to the CS that define the rights or restrictions needed.

VREPLY - (Section 8.5) This component encapsulates a set of data that can consist of an arbitrary amounts of properties and components. Its contents is dependent on the command that was issued.

VQUERY - The search operation makes use of a new component, called "VQUERY" (Section 8.6) and a new value type "CAL-QUERY" (Section 6.1.1). "VQUERY" is used to fetch objects from the CS.

2.2 Relationship of RFC-2446 (ITIP) and CAP

[iTIP] describes scheduling methods which result in indirect manipulation of components. In CAP, the "CREATE" command is used to deposit entities into the store. Other CAP commands such as "DELETE", "MODIFY" and "MOVE" provide direct manipulation of components. In the CAP calendar store model, scheduling messages are conceptually kept separate from other components by their state.

All scheduling operations and are as define in [iTIP]. This memo makes no changes to any of the workflow described in [iTIP]. In this memo when referring to the presence of the "METHOD" property in an object is the same as saying an [iTIP] object.

A CUA may create a "BOOKED" entry by depositing a iCalendar object into the store. This is done by depositing an object that does not have a "METHOD" property. The CS then knows to set the state of the object to "BOOKED". If the object has a "METHOD" property then the object is stored in the "UNPROCESSED" state.

If existing "UNPROCESSED" objects exist in the CS for the same UID then a CUA may wish to consolidate the objects in to one "BOOKED" object. The CUA would fetch the "UNPROCESSED" objects for that UID and process them in the CUA as described in [iTIP]. Then if the CUA wished to book the UID, then the CUA would issue a "CREATE" command to create the new "BOOKED" object in the CS, followed by a "DELETE" command to remove any related old [iTIP] objects from the CS. And it might also involve having the CUA send some [iMIP] objects or contacting other CS's and performing CAP operations on those CSs.

The CUA could also decide not to book the object. In which case the "UNPROCESSED" objects could be deleted from the CS. Or the CUA could set those object to the marked for delete.

The marked for delete state is used to keep the object around so that the CUA can process duplicate requests automatically. If a duplicate [iTIP] object is deposited into the CS and there exists identical marked for delete objects, then a CUA acting on behalf of the "OWNER" can silently drop those duplicate entries.

Another purpose for the marked for delete state is so that when a CU decides they do not wish to have the object show in their calendar, the CUA can book the object, changing the PARTSTAT parameter to "DECLINED" in the "ATTENDEE" property that corresponds to their UPN. Perform an iTIP processing such as sending back a decline. Then mark that object as marked for delete. Their CUA might be configurable to automatically drop any updates for that object knowing the CU has already declined.

When synchronizing with multiple CUAs, the marked for delete state could be used to inform the synchronization process that an object is to be deleted. How synchronization is done is not specified in this memo.

Several "UNPROCESSED" entries can be in the CS for the same UID. However once consolidated, then only one entry exists in the CS and that is the booked object. The others MUST BE removed, or have their state changed to "DELETED".

There MUST NOT BE more than one "BOOKED" entry in a calendar for the same "UID".

For example, if you were on vacation, you could have a REQUEST to attend a meeting and several updates to that meeting. Your CUA would have to "SEARCH" them out of the CS using CAP, process them, determine what the final state of the object from a possible combination of user input and programmed logic. Then the CUA would instruct the CS to create a new booked entry from the consolidated results. Finally, the CUA could do a "DELETE" or change their state to "DELETED" for all of these now old scheduling requests in the CS. See [iTIP] for details on resolving multiple [iTIP] scheduling entries.

3. CAP Design

3.1 System Model

The system model describes the high level components of a calendar system and how they interact with each other.

CAP is used by a "Calendar User Agent" (CUA) to send commands to and receive responses from a "Calendar Server" (CS).

The CUA prepares a [MIME] encapsulated command, sends it to the CS, and receives a [MIME] encapsulated response. The calendaring related information within these messages are represented by iCalendar objects.

There are two distinct protocols in operation to accomplish this exchange. [BEEP] is the transport protocol is used to move these encapsulations between a CUA and a CS. CAP's [BEEP] profile defines the application protocol where the content and semantics of the messages sent between the CUA and the CS are specified.

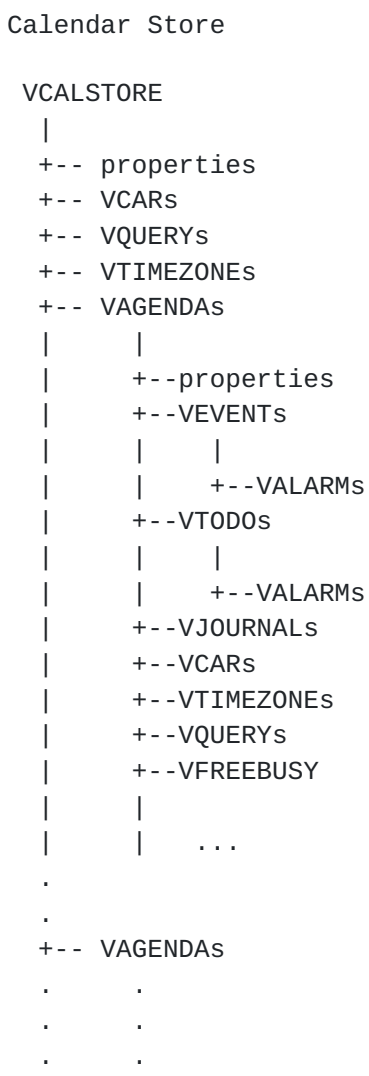
3.2 Calendar Store Object Model

[iCAL] describes components such as events, todos, alarms, and timezones. [CAP] requires additional object infrastructure. In particular, detailed definitions of the containers for events and todos (calendars), access control objects, and a query language.

The conceptual model for a calendar store is shown below. The calendar store (VCALSTORE - Section 8.2) contains "VCAR"s, "VQUERY"s, "VTIMEZONE"s, "VAGENDA"s and calendar store properties.

Calendars (VAGENDAs) contain "VEVENT"s, "VTODO"s, "VJOURNAL"s, "VCAR"s, "VTIMEZONE"s, "VFREEBUSY", "VQUERY"s and calendar properties.

The component "VCALSTORE" is used to denote the a root of the calendar store and contains all of the calendars.



Calendars within a Calendar Store are identified by their unique Relative CALID.

3.3 Protocol Model

CAP uses beep as the transport and authentication protocol.

The initial charset MUST BE UTF-8 for the session in an unknown locale. If the CS supplied the BEEP 'localize' attribute in the BEEP 'greeting' then the CUA may tell the CS to switch locales for the session by issuing the "SET-LOCALE" CAP command and supplying one of the locales supplied by the BEEP 'localize' attribute. If supplied the first locale supplied in the BEEP 'localize' attribute MUST BE the default locale of the CS. The locale is switched only after a successful reply.

The "DEFAULT-CHARSET" property of the CS contains the list of charsets supported by the CS with the first value being the default for new calendars. If the CUA wishes to switch to one of those charsets for the session, the CUA issues the "SET-LOCALE" CAP command. The CUA would have to first perform a "GET-CAPABILITY" command on the CS to get the list of charsets supported by the CS. The charset is switched only after a successful reply.

The CUA may switch locales and charsets as needed. There is no requirement that a CS support multiple locales or charsets.

3.3.1 Use of BEEP, MIME and iCalendar

CAP uses the BEEP application protocol over TCP. (refer to [BEEP] and [BEEPTCP] for more information). The default port that the Calendar Server listens for connections is on user port 1026.

The BEEP data exchanged in CAP is a iCalendar MIME content that fully conforms to [iCAL] iCalendar format.

This example tells the CS to generate and return 10 UIDs to be used by the CUA. (Note throughout this memo, 'C:' refers to what the CUA sends, 'S:' refers to what the CS sends, 'I:' refers to what the initiator sends, and 'L:' refers to what the listener sends. Where initiator and responder are used as defined in [BEEP].)

```
C: MSG 1 2 . 432 62
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: PRODID:-//someone's prodid
C: CMD;ID=unique-per-cua-123;OPTIONS=10:GENERATE-UID
C: END:VCALENDAR
C: END
```

NOTE: The following examples will not include the BEEP header and footer information. Only the iCalendar objects that are sent between the CUA and CS will be shown as the BEEP payload boundaries are independent of CAP.

The commands listed below are used to manipulate or access the data on the calendar store:

ABORT - Sent to halt the processing of any command except ABORT. (Section 9.1.2)

CONTINUE - Sent to continue processing a command that has had its specified timeout time reached. (Section 9.1.3)

CREATE - Create a new object on the CS. This can be implied for iTIP objects. Initiated by the CUA only. (Section 9.1.4)

SET-LOCALE - Tell the CS to use any named locale and charset supplied. Initiated by the CUA only. (Section 9.9)

DELETE - Delete objects from the CS. Initiated by the CUA only. Can also be used to mark a object for deletion. (Section 9.1.5)

GENERATE-UID - Generate one or more unique ids. Initiated by the CUA only. (Section 9.2)

GET-CAPABILITY - Query the capabilities the other end point of the session. (Section 9.3)

IDENTIFY - Set a new identity for the session. Initiated by the CUA only. (Section 9.4)

MODIFY - Modify components. Initiated by the CUA only. (Section 9.5)

MOVE - Move components to another container. Initiated by the CUA only. (Section 9.6)

REPLY - When replying to a command, the "CMD" value will be set to "REPLY" so that it will not be confused with a new command. (Section 9.7)

SEARCH - Search for components. Initiated by the CUA only. (Section 9.8)

TIMEOUT - Sent when a specified amount of time has lapsed and a command has not finished. (Section 9.10)

4. Security Model

The BEEP transport performs all session authentication.

4.1 Calendar User and UPNs

A Calendar User (CU) is an entity that can be authenticated. It is represented in CAP as a UPN, which is a key part of access rights. The UPN representation is independent of the authentication mechanism used during a particular CUA/CS interaction. This is because UPNs are used within VCARs. If the UPN were dependent on the authentication mechanism, a VCAR could not be consistently evaluated. A CU may use one mechanism while using one CUA but the same CU may use a different authentication mechanism when using a different CUA, or while connecting from a different location.

The user may also have multiple UPNs for various purposes.

Note that the immutability of the user's UPN may be achieved by using SASL's authorization identity feature. (The transmitted authorization identity may be different than the identity in the client's authentication credentials.) [SASL, section 3]. This also permits a CU to authenticate using their own credentials, yet request the access privileges of the identity for which they are proxying SASL. Also, the form of authentication identity supplied by a service like TLS may not correspond to the UPNs used to express a server's access rights, requiring a server specific mapping to be done. The method by which a server determines a UPN, based on the authentication credentials supplied by a client, is implementation specific. See [BEEP] for authentication details; [BEEP] relies on SASL.

4.1.1 UPNs and Certificates

When using X.509 certificates for purposes of CAP authentication, the UPN should appear in the certificate. Unfortunately there is no single correct guideline for which field should contain the UPN.

From RFC-2459, section 4.1.2.6 (Subject):

> If subject naming information is present only in the subjectAlt-Name extension (e.g., a key bound only to an email address or URI), then the subject name MUST be an empty sequence and the subjectAltName extension MUST BE critical.
>
> Implementations of this specification MAY use these comparison rules to process unfamiliar attribute types (i.e., for name chaining). This allows implementations to process certificates

with unfamiliar attributes in the subject name.

In addition, legacy implementations exist where an RFC 822 name is embedded in the subject distinguished name as an EmailAddress attribute. The attribute value for EmailAddress is of type IA5String to permit inclusion of the character '@', which is not part of the PrintableString character set. EmailAddress attribute values are not case sensitive (e.g., "fanfeedback@redsox.com" is the same as "FANFEEDBACK@REDSOX.COM").

Conforming implementations generating new certificates with electronic mail addresses MUST use the rfc822Name in the subject alternative name field (see sec. 4.2.1.7 of [RFC 2459]) to describe such identities. Simultaneous inclusion of the EmailAddress attribute in the subject distinguished name to support legacy implementations is deprecated but permitted.

Since no single method of including the UPN in the certificate will work in all cases, CAP implementations MUST support the ability to configure what the mapping will be by the CS administrator. Implementations MAY support multiple mapping definitions, for example, the UPN may be found in either the subject alternative name field, or the UPN may be embedded in the subject distinguished name as an EmailAddress attribute.

Note: If a CS or CUA is validating data received via iMIP, if the "ORGANIZER" or "ATTENDEE" property said (e.g.) "ATTENDEE;CN=Joe Random User:MAILTO:juser@example.com" then the email address should be checked against the UPN. This is so the "ATTENDEE" property cannot be changed to something misleading like "ATTENDEE;CN=Joe Rictus User:MAILTO:jrictus@example.com" and have it pass validation. Note that it is the email addresses that miscompare, the CN miscompare is irrelevant.

4.1.2 Anonymous Users and Authentication

Anonymous access is often desirable. For example an organization may publish calendar information that does not require any access control for viewing or login. Conversely, a user may wish to view unrestricted calendar information without revealing their identity.

4.1.3 User Groups

A User Group is used to represent a collection of CUs or other UGs that can be referenced in VCARs. A UG is represented in CAP as a UPN. The CUA cannot distinguish between a UPN that represents a CU or a UG.

UGs are expanded as necessary by the CS. The CS MAY expand a UG (including nested UGs) to obtain a list of unique CUs. Duplicate UPNs are filtered during expansion.

How the UG expansion is maintained across commands is implementation specific. A UG may reference a static list of members, or it may represent a dynamic list. Operations SHOULD recognize changes to UG membership.

CAP does not define commands or methods for managing UGs.

4.2 Access Rights

Access rights are used to grant or deny access to calendars, components, properties, and parameters in a CS to a CU. CAP defines a new component type called a Calendar Access Right (VCAR). Specifically, a "VCAR" component grants, or denies, UPNs the right to read and write components, properties, and parameters on calendars within a CS.

The VCAR model does not put any restriction on the sequence in which the object and access rights are created. That is, an object associated with a particular VCAR might be created before or after the actual VCAR is defined. In addition, the VCAR and VEVENT definition might be created in the same iCalendar object and passed together in a single object.

All rights MUST BE denied unless specifically granted.

If two rights specified in VCAR components are in conflict, the right that denies access always takes precedence over the right that grants access. Any attempt to create a VCAR that conflicts with an immutable VCAR must fail.

4.2.1 Access Control and NOCONFLICT

The TRANSP property can take on values "TRANSPARENT-NOCONFLICT" and "OPAQUE-NOCONFLICT" that prohibit other components from overlapping it. This setting overrides access. The "ALLOW-CONFLICT" CS, Calendar or component setting may also prevent overlap, returning an error code "6.3".

4.2.2 Calendar Access Right (VCAR)

Access rights within CAP are specified with the "VCAR" component, "RIGHTS" value type and the "GRANT", "DENY" and "CARID" properties.

Properties within an iCalendar object are unordered. This also is

the case for the "VCAR" properties.

4.2.3 Predefined VCARs

Predefined calendar access CARIDs that MUST BE implemented are:

CARID:READBUSYTIMEINFO - Specifies the "GRANT" and "DENY" rules that allow UPNs to read "VFREEBUSY" components. An example definition for this VCAR is:

```
BEGIN:VCAR
CARID:READBUSYTIMEINFO
BEGIN:VRIGHT
GRANT:*
PERMISSION:READ
SCOPE:SELECT * FROM VFREEBUSY
END:VRIGHT
END:VCAR
```

CARID:REQUESTONLY - Specifies the "GRANT" and "DENY" rules to UPNs other than the owner of the calendar the ability to write new objects with the property "METHOD" property set to the "REQUEST" value. This CARID allows the owner to specify which UPNs are allowed to make scheduling requests. An example definition for this VCAR is:

```
BEGIN:VCAR
CARID:REQUESTONLY
BEGIN:VRIGHT
GRANT:NON OWNER()
PERMISSION:CREATE
RESTRICTION:SELECT * FROM VCALENDAR WHERE METHOD = 'REQUEST'
END:VRIGHT
END:VCAR
```

CARID:UPDATEPARTSTATUS - Grants to authenticated users the right to modify the instances of the "ATTENDEE" property set to one of their calendar addresses in any components for any booked component containing a "ATTENDEE" property. This allows (or denies) a CU the ability to update their own participation status

in a calendar where they might not otherwise have MODIFY access. They are not allowed to change the "ATTENDEE" property value. An example definition for this VCAR is (This example only effects the "VEVENT" components):

```
BEGIN:VCAR
CARID:UPDATEPARTSTATUS
BEGIN:VRIGHT
GRANT:*
PERMISSION:MODIFY
SCOPE:SELECT ATTENDEE FROM VEVENT
 WHERE ATTENDEE = SELF()
 AND ORGANIZER = CURRENT-TARGET()
 AND STATE() = 'BOOKED'
RESTRICTION:SELECT * FROM VEVENT
 WHERE ATTENDEE = SELF()
END:VRIGHT
END:VCAR
```

CARID:DEFAULTOWNER - Grants to any owner the permission they have for the target. An example definition for this VCAR is:

```
BEGIN:VCAR
CARID:DEFAULTOWNER
BEGIN:VRIGHT
GRANT:OWNER()
PERMISSION:*
SCOPE:SELECT * FROM VAGENDA
END:VRIGHT
END:VCAR
```

4.2.4 Decreed VCARs

A CS MAY choose to implement and allow persistent immutable VCARs that may be configured by the CS administrator. A reply from the CS may dynamically create VCARs that are decreed depending on the implementation. To the CUA any "VCAR" component with the "DECREED" property set to "TRUE" can not be changed by the currently authenticated UPN, and depending on the implementation and other

VCARs; might not be able to be changed by any UPN using CAP, and never when the CUA gets a "DECREED:TRUE" VCAR.

When a user attempts to modify or override a decreed VCAR an error will be returned indicating that the user has insufficient authorization to perform the operation. The reply to the CUA MUST BE the same as if a non-decreed VCAR caused the failure.

The CAP protocol does not define the semantics used to initially create a decreed VCAR. This administrative task is outside the scope of the CAP protocol.

For example; an implementation or a CS administrator may wish to define a VCAR that will always allow the calendar owners to have full access to their own calendars.

Decreed VCARs MUST BE readable by the calendar owner in standard VCAR format.

4.3 CAP Session Identity

A BEEP session has an associated set of authentication credentials, from which is derived a UPN. This UPN is the identity of the CAP session, and is used to determine access rights for the session.

The CUA may change the identity of a CAP session by calling the "IDENTIFY" command. The Calendar Server only permits the operation if the session's authentication credentials are good for the requested identity. The method of checking this permission is implementation dependent, but may be thought of as a mapping from authentication credentials to UPNs. The "IDENTIFY" command allows a single set of authentication credentials to choose from multiple identities, and allows multiple sets of authentication credentials to assume the same identity.

For anonymous access the identity of the session is "@". A UPN with a null Username and null Realm is anonymous. A UPN with a null Username, but non-null Realm, such as "@foo.com" may be used to mean any identity from that Realm, which is useful to grant access rights to all users in a given Realm. A UPN with a non-null Username and null Realm, such as "bob@" could be a security risk and MUST NOT be used.

Since the UPN includes Realm information it may be used to govern calendar store access rights across Realms. However, governing access rights across Realms is only useful if login access is available. This could be done through a trusted server relationship or a temporary account. Note that trusted server relationships are

outside the scope of [CAP].

The "IDENTIFY" command also provides for a weak group implementation. By allowing multiple sets of authentication credentials belonging to different users to identify as the same UPN, that UPN essentially identifies a group of people, and may be used for group calendar ownership, or the granting of access rights to a group.

5. CAP URL and Calendar Address

The CAP URL scheme is used to designate calendar stores and calendars accessible using the CAP protocol.

The CAP URL scheme conform to the generic URL syntax, defined in RFC 2396, and follows the Guidelines for URL Schemes, set forth in RFC 2718.

A CAP URL begins with the protocol prefix "cap" and is defined by the following grammar.

```
capurl   = "cap://" csid [ "/" relcalid ]
csid     = hostport   ; As defined in Section 3.2.2 of RFC 2396
relcalid = *uric      ; As defined in Section 2 of RFC 2396
```

'relcalid' is an identifier that uniquely identifies a calendar on a particular calendar store. There is no implied structure in a Relative CALID. It may refer to the calendar of a user or of a resource such as a conference room. It MUST BE unique within the calendar store.

Examples:

```
cap://cal.example.com
cap://cal.example.com/Company/Holidays
cap://cal.example.com/abcd1234Usr
```

Relative CAP URLs are permitted and are resolved according to the rules defined in Section 5 of RFC 2396.

Examples of valid relative CAP URLs:

```
opqaueXzz123String
UserName/Personal
```

A Calendar addresses can be described as qualified or relative CAP URLs.

For a user currently authenticated to the CS on cal.example.com, these two example calendar addresses refer to the same calendar:

```
cap://cal.example.com/abcd1234USR
abcd1234USR
```

6. New Components, Properties, Parameters, and Values

The following sections contains new components, properties, parameters, and value definitions.

The purpose of these is to extend the iCalendar objects in a compatible way so that existing iCalendar VERSION 2.0 parsers can still parse the objects without modification.

6.1 Property Value Data Types

6.1.1 CAL-QUERY Value Type

Subject: Registration of text/calendar MIME value type CAL-QUERY

Value Name: CAL-QUERY

Value Type Purpose: This value type is used to identify values and contains query statements targeted at locating those values.

This is based on [SQL92] and [SQLCOM].

1. For the purpose of a query, all components should be handled as tables, and the properties of those components, should be handled as columns.

2. All VAGENDAs and CS's look like tables for the purpose of a QUERY. And all of their properties look like columns in those tables.

3. You CAN NOT do any cross component-type joins. And that means you can ONLY have one component, OR one VAGENDA OR one VCALSTORE in the the FROM clause.

4. Everything in the SELECT and WHERE clauses MUST BE from the same component type, or VAGENDA OR VCALSTORE in the FROM clause.

5. When multiple QUERY properties are supplied in a single VQUERY component, the results returned are the same as the results returned for multiple VQUERY components having each a single QUERY property and the results are return in the same order as the VQUERYs were specified in the original command.

6. The '.' is used to separate the table name (component) and column name (property or component) when selecting a property that is contained inside of a component that is targeted in the TARGET property.

7. A contained component without a '.' is not the same as "component-name.*". If given as "component-name" (no dot) the the encapsulating BEGIN/END statement will be supplied for "component-name".:

In this example the '.' is used to separate the TRIGGER property from its contained component (VALARM) which is contained in any VEVENT in the selected TARGET (relcalid). All TRIGGER values in any VEVENT in relcalid would be returned.

```
TARGET:relcalid
QUERY:SELECT VALARM.TRIGGER FROM VEVENT
```

```
SELECT VALARM FROM VEVENT WHERE UID = "123"
```

This return one BEGIN/END VALARM for each VALARM in the VEVENT as there is no '.' (dot) in the VALARM:

```
BEGIN:VALARM
TRIGGER;RELATED=END:PT5M
REPEAT:4
...
END:VALARM
BEGIN:VALARM
TRIGGER;RELATED=START:PT5M
DURATION:PT10M
...
END:VALARM
...
...
```

If provided as "component-name.*", then only the properties and any contained components will be returned:

SELECT VALARM.* FROM VEVENT WHERE UID = "123"

Will return the properties in each VALARM in the VEVENT:

```
TRIGGER;RELATED=END:PT5M
REPEAT:4
...
TRIGGER;RELATED=START:PT5M
DURATION:PT10M
...
...
```

(a) SELECT VEVENT.<a-property-name> FROM VEVENT

(b) SELECT VALARM FROM VEVENT

(c) SELECT VALARM.* FROM VEVENT

(d) SELECT * FROM VEVENT

(e) SELECT * FROM VEVENT WHERE
VALARM.TRIGGER < '20020201T000000Z'
AND VALARM.TRIGGER > '20020101T000000Z'

Note: (a) Selects all instances of <a-property-name> from all VEVENT components.

(b) and (c) Select all VALARM components from all VEVENT components. (b) would return then in BEGIN/END VALARM tags. (c) would return all of the properties without BEGIN/END VALARM tags.

(d) Selects every property and every component that is in any VEVENT component.

(e) Selects all properties and all contained components in all VEVENT components that have a VALARM with a TRIGGER property value between the provided dates and times.

NOT VALID:

(f) SELECT VEVENET.VALARM.TRIGGER FROM VEVENT

(g) SELECT DTSTART,UID FROM VEVENT WHERE

```
                VTODO.SUMMERY = "Fix typo in CAP"
```

Note: (f) Is NOT valid because it contains two '.' characters in the SELECT clause.

(g) Is NOT valid because it mixes VEVENT and VTODO properties in the same VQUERY.

Formal Definition: The value type is defined by the following notation:

```
comp-name  = "VEVENT"    / "VTODO"   / "VJOURNAL"
           / "VTIMEZONE" / "VALARM"  / "VFREEBUSY"
           / "VAGENDA"   / "VCAR"    / "VCALSTORE"
           / "VQUERY"    / iana-name / x-comp

querycomp  = queries

             ; These next three property types
             ; may be in any order.
             ;
           / ( queryid *(name) queries)

             ; Only when using an existing stored query
             ; can query or queries be omitted.
             ;
           / queryid


queries    = query
           / queries query

             ; NOTE: There is exactly one space separating
             ; the various parts of cal-query
             ;
cal-query  = "SELECT"   SP   cap-cols  SP
             "FROM"     SP   comp-name SP
             *(cauprops SP / capcprops SP)
             "WHERE"    SP   cap-expr

           / "SELECT" SP cap-cols SP
             "FROM"   SP comp-name

uprop-list  = (cap-col SP cap-local)
            / uprop-list SP cap-col SP cap-local
```

```
cprop-list  = (cap-comp cap-local)
            / cprop-list SP cap-col SP cap-local

cap-col     = ; Any property name found in the component
              ; named in the comp-tbl used in the FROM clause.
              ;
              ;   SELECT ORGANIZER FROM VEVENT ...
              ;
              ; OR
              ;
              ; A component name of an existing component contained
              ; inside of the cmp-tbl used in the FROM clause.
              ;
              ;   SELECT VALARM FROM VEVENT ...

              ; NOTE: there is NO space around the "," on
              ; the next line
cap-cols    = cap-col / ( cap-cols "," cap-col)
              / "*"
            /
cap-param   = ; Any parameter that may be contained in the cap-col
              ; in the supplied PARAM() function

cap-local   = ; Any string that is composed of the characters
              ; that could be a cap-col name, but is not any
              ; cap-col name. It is suggested that the
              ; string start with "my-" to ensure it does not
              ; conflict with any existing or future cap-col name.
              ; This name MUST BE defined in the cap-using and
              ; can only be used in cap-expr of the same query.
              ; And this name is only known and valid for the
              ; provided query and only for the lifetime of
              ; the query. If multiple QUERY properties exist
              ; in the same component, it is only valid and usable
              ; in the same QUERY property where it was supplied.

col-value   = col-literal
            / "STATE()"
            / "SELF()"
            / "CAL-OWNERS()"
            / "CAL-OWNERS(" cal-address ")"
            / "CURRENT-TARGET()"

cal-address = ; A CALID as define by CAP

col-literal = "'" literal-data "'"

literal-data = ; Any data that matches the value type of the
```

```
                    ; column that is being compared. That is you can
                    ; not compare PRIORITY to "some string" because
                    ; PRIORITY has a value type of integer. If it is
                    ; not preceded by the LIKE element, any '%' and '_'
                    ; characters in the literal data are not treated as
                    ; wildcard characters and do not have to be backslash
                    ; escaped.
                    ;
                    ; OR
                    ;
                    ; If the literal-data is preceded by the LIKE
                    ; element it may also contain the '%' and '_'
                    ; wildcard characters. And if the literal data
                    ; that is comparing contains any '%' or '_'
                    ; characters, they MUST BE backslash escaped as
                    ; described in the notes below in order for them not
                    ; to be treated as wildcard characters.

      cap-ucol    = cap-col / cap-local

      cap-expr    = "(" cap-expr ")"
                  / cap-term

      cap-term    = cap-expr SP cap-logical SP cap-expr
                  / cap-factor

      cap-factor  = cap-colval SP cap-oper SP col-value
                  / cap-colval SP "NOT LIKE" SP col-value
                  / cap-colval SP "LIKE" SP col-value
                  / cap-colval SP "IS NULL"
                  / cap-colval SP "IS NOT NULL"
                  / col-value SP "NOT IN" cap-colval"
                  / col-value SP "IN" cap-colval"

      cap-colval     = cap-ucolq
                  / "PARAM(" cap-ucol "," cap-param ")"

      cap-oper    = "="
                  / "!="
                  / "<"
                  / ">"
                  / "<="
                  / ">="

      cap-logical = "AND" / "OR"

      SP          = ; A single white space ascii character
                    ; (value in HEX %x20).
```

```
CRLF       = ; As defined in RFC 2445.

xparam     = ; As defined in RFC 2445.

x-prop     = ; As defined in RFC 2445.

x-comp     = ; As defined in RFC 2445.
```

6.1.1.1 CAL-OWNERS()

This function returns the list of "OWNERS" for the named calendar.

If called as 'CAL-OWNERS()', it is equivalent to the comma separated list of all of the owners of the current "TARGET" calendar. If the target is a "VAGENDA", it returns the "CALMASTER" value.

If called as 'CAL-OWNERS(cal-address)', then it is the equivalent to the comma separated list of owners for the named calendar id.

6.1.1.2 CURRENT-TARGET()

Is equivalent to the value of the "TARGET" property in the current command. Used in a CAL-QUERY 'WHERE' clause.

6.1.1.3 [NOT] OWNER()

Returns true if the current UPN is an owner of the current "TARGET". Used in a CAL-QUERY 'WHERE' clause and in the UPN-FILTER.

6.1.1.4 SELF()

Used in a CAL-QUERY 'WHERE' clause. Returns the UPN of the currently authenticated CU.

6.1.1.5 STATE()

Returns one of three values, 'BOOKED', 'UNPROCESSED', or 'DELETED' depending on the state of the object. Used in a CAL-QUERY 'WHERE' clause.

6.1.1.6 Ordering of Results

Sorting will take place in the order the columns are supplied in the QUERY command. The CS MUST sort at least the first column. The CS MAY sort additional columns.

Float and integer values MUST BE sorted by their numeric value. This means the result of a sort on an integer value type will be:

1, 2, 100, 1000

and not

1, 100, 1000, 2

This means the result of a sort on an float value type will be:

1.1, 2.23, 100.332, 1000.12

and not

1.1, 100.332, 1000.12, 2.23

Date and date time values will be sorted by their equivalent value in UTC. No matter what the returned time zone in the result set returns. This is so that if multiple components are returned each in a unique time zone, the results will be sorted in UTC. This does not mean the values MUST BE converted to UTC in the data returned to the CUA. It means the CS must do the sort in UTC.

All other values are sorted according to the locale sorting order as specified in the command. Or the calendar locale if known, or the CS locale if the calendar does not have any locale set. And the locale to use for the sort is determined in that order.

6.1.1.7 Date sorting order

If the cap-cols is only "*" and nothing else and the result set has a DTSTART, then:

If EXPAND=FALSE sorting will be by the DTSTART value ascending as if it were in UTC.

If EXPAND=TRUE sorting will be by the RECURRENCE-ID value ascending as if it were in UTC.

If one or more DTSTART or RECURRENCE-ID components have exactly the same value, the order for those matching components is unspecified.

If the selected component(s) do not contain a DTSTART or a

RECURRENCE-ID, then the order is unspecified.

If an instance does not have a RECURRENCE-ID and the query compares RECURRENCE-IDs (comparing a RECURRENCE-ID to the date or date/time of a single instance object), then the CS MUST compare the DTSTART value as if it were a RECURRENCE-ID even for single instance objects that do not contain a RECURRENCE-ID.

A component with a DATE and no TIME value is returned before objects with both a DATE and TIME value when the dates of those two (or more) objects are the same, sorted by date.

6.1.1.8 Use of single quote

All literal values are surrounded by single quotes ('), not double quotes (") , and not without any quotes. If the value contains quotes or any other ESCAPED-CHAR, they MUST BE backslash escaped as described in section "4.3.11 Text" of RFC2445. Any LIKE wildcard characters that are part of any literal data that is preceded by a LIKE clause and is not intended to mean wildcard search, MUST BE escaped as described in note (7) below.

6.1.1.9 Comparing DATE and DATE-TIME values

When comparing DATE-TIME to DATE value types and when comparing DATE to DATE-TIME value types, the result will be true if the DATE value is on the same day as the DATE-TIME value. And they are compared in UTC no matter what time zone the data may actual have been stored in.

VALUE-1	VALUE-2	Compare Results
20020304 (in UTC-3)	20020304T123456 (in UTC-3)	TRUE
20020304 (in UTC)	20020304T003456 (in UTC-4)	FALSE
20020304T003456Z (in UTC-0)	20020205T003456 (in UTC-7)	FALSE

When the DATE or DATE-TIME value is not associated with a time zone, then the CS will compare the value assuming that the no time zone values are in the calendars default time zone.

When comparing DATE and DATE-TIME values with the LIKE clause the comparison will be done as if the value is a RFC2445 DATE or DATE-

TIME string value.

LIKE '2002%' will match anything in the year 2002.

LIKE '200201%' will match anything in January 2002.

LIKE '%T000000' will match anything at midnight.

LIKE '____01__T%' will match anything for any year or time that is in January. (Four '_', '01', two '_' 'T%').

Using a LIKE value of "%00%, would return any value that contained two consecutive zeros.

Again all comparisons will be done in UTC.

6.1.1.10 DTEND and DURATION

When a QUERY contains a DTEND value, then the CS MUST also evaluate any existing DURATION property value and determine if it has an effective end time that matches the QUERY supplied DTEND value or any range of values supplied by the QUERY.

When a QUERY contains a DURATION value, then the CS MUST also evaluate any existing DTEND property value and determine if it has an effective duration that matches the QUERY supplied DURATION value or any range of values supplied by the QUERY.

As DTEND is the first time that is excluded from a components time range, any DURATION supplied by the QUERY that is exactly one second less than DTEND MUST match the QUERY. And if the DURATION ends exactly at the computed DTEND it MUST NOT match.

Any DTEND supplied by the QUERY that is exactly one second more than an end time computed from a DURATION MUST match the QUERY. Any end time that is computed from a DURATION that exactly matches the supplied DTEND MUST NOT match.

Given a meeting room reserved with a component that contains (date-time-example-1):

```
DTSTART:20020127T000000Z
DTEND:20020127T010000Z

The reservation is really from:

        January 27th, 2002 00:00:00
To:

        January 27th, 2002,00:59:59
```

Given another meeting room reserved with a component that contains (date-time-example-2):

```
DTSTART:20020127T000000Z
DURATION:P59M59S

The reservation is really from:

       January 27th, 2002 00:00:00

To:

       January 27th, 2002,00:59:59
```

A QUERY that contains:

```
... VEVENT.DTSTART = '20020127T00000Z'
AND VEVENT.DTEND = '20020127T010000Z'
```

MUST match both (date-time-example-1) and (date-time-example-2)

A QUERY that contains:

```
... VEVENT.DTSTART = '20020127T00000Z'
AND DURATION = 'P59M59S'
```

MUST match both (date-time-example-1) and (date-time-example-2)

6.1.1.11 [NOT] LIKE

The pattern matching characters are the '%' that matches zero or more characters, and '_' that matches exactly one character (where character does not always mean octet).

LIKE pattern matches always cover the entire string. To match a pattern anywhere within a string, the pattern must start and end with a percent sign.

To match a '%' or '_' in the data and not have it interpreted as a wildcard character, they MUST BE backslash escaped. That is to search for a '%' or '_' in the string:

```
LIKE '%\%%'    Matches any string with a '%' in it.
LIKE '%\_%'    Matches any string with a '_' in it.
```

Strings compared using the LIKE clause MUST BE performed using case in-sensitive comparisons when the locale allows. (Example: in US-ASCII the compare assumes 'a' = 'A').

If LIKE is preceded by 'NOT' then there is a match when the string compare fails.

Some property values (such as the 'recur' value type), contain commas and are not multi valued. The CS must understand the objects being compared and understand how to determine how any multi valued or multi instances properties or parameter values are separated, quoted, and backslash escaped and perform the comparisons as if each value existed by itself and not quoted or backslash escaped when comparing using the IN element.

And see the examples in the next section (IN).

6.1.1.12 Empty vs. NULL

When used in a CAL-QUERY value, "NULL" means that the property or parameter is not present in the object.

If the property (or parameter) exists, but has no value then "NULL" MUST NOT match.

If the property (or parameter) exists, but has no value then it matches the empty string '' (quote quote).

6.1.1.13 [NOT] IN

This is similar to the LIKE element, except it does value matching and not string comparison matches.

Some iCalendar objects can be multi instance and multi valued. The IN operator will return a match if the literal value supplied as part of the 'IN' clause is contained in the value of any instance of the named property or parameter, or is in any of the multiple values in the named property or parameter. Unlike the 'LIKE' clause, the '%' and '_' matching characters are not used with the 'IN' clause and have no special meaning.

```
          BEGIN:A-COMPONENT
     a    property:value1,value2        One property, two values.
     b    property:"value1,value2"      One property, one value.
     c    FOO:parameter=1,2:x           One parameter, two values.
     d    FOO:parameter="1,2",3:y       One parameter, one value.
     e    FOO:parameter=",":z           One parameter, one value.
     f    property:x,y,z                One property, three values
          END:A-COMPONENT

          'value1' IN property          would match (a) only.
          'value1,value2' IN property   would match (b) only.
          'value%'  IN property         would NOT match any.
          ',' IN property               would NOT match any.
          '%,%' IN property             would NOT match any.
          'x' IN property               would match (f) and (c).
          '2' IN parameter              would match (c) only.
          '1,2' IN parameter            would match (d) only.
          ',' IN parameter              would match (e) only.
          '%,%' IN parameter            would NOT match any.

          property  LIKE 'value1%'      would match (a) and (b)
          property  LIKE 'value%'       would match (a) and (b)
          property  LIKE 'x'            would match (f) and (c).
          parameter LIKE '1%'           would match (c) and (d)
          parameter LIKE '%2%'          would match (c) and (d)
          parameter LIKE ','            would match (e) only.
```

Some property values (such as the 'recur' value type), contain commas and are not multi valued. The CS must understand the objects being compared and understand how to determine how any multi valued or multi instances properties or parameter values are separated, quoted, and backslash escaped and perform the comparisons as if each value existed by itself and not quoted or backslash escaped when comparing

using the IN element.

If IN is preceded by 'NOT' then there is a match when the value does not exist in the property or parameter value.

6.1.1.14 DATE-TIME and TIME values in a WHEN clause

All DATE-TIME and TIME literal values supplied in a WHEN clause MUST BE terminated with 'Z'. That means that the CUA MUST supply the values in UTC.

Valid:

```
WHERE alarm.TRIGGER < '20020201T000000Z'
AND alarm.TRIGGER > '20020101T000000Z'
```

Not valid and it is a syntax error and the CS MUST reject the QUERY.

```
WHERE alarm.TRIGGER < '20020201T000000'
AND alarm.TRIGGER > '20020101T000000'
```

6.1.1.15 Multiple contained components

All comparisons MUST BE done from the same instance of a contained component or property and repeated for each instance. As in the following example that uses a VALARM component contained in a VEVENT. If any instance of VALARM in VEVENT matches the query and the rest of the query is satisfied, then the UID, SUMMARY, and DESCRIPTION from the VEVENT will be returned. If there were two VALARMs in a VEVENT, then both VALARMs are tested and in this example only when the VEVENT state is booked:

```
BEGIN:VQUERY
EXPAND:TRUE
QUERY:SELECT UID,SUMMARY,DESCRIPTION FROM VEVENT
 WHERE VALARM.TRIGGER >= '20000101T030405Z'
 AND VALARM.TRIGGER <= '20001231T235959Z'
 AND STATE() = 'BOOKED'
END:VQUERY
```

6.1.1.16 Example, Query by UID

The following example would match the entire content of a "VEVENT" or "VTODO" component with the "UID" property equal to "uid123" and not expand any multiple instances of the component. If the CUA does not know if "uid123" was a "VEVENT", "VTODO", "VJOURNAL", or any other component, then all components that the CUA supports MUST BE supplied in a QUERY property. This example assumes the CUA is only interested in "VTODO" and "VEVENT" components.

If the results were empty it could also mean that "uid123" was a property in a component other than a VTODO or VEVENT.

```
BEGIN:VQUERY
QUERY:SELECT * FROM VTODO WHERE UID = 'uid123'
QUERY:SELECT * FROM VEVENT WHERE UID = 'uid123'
END:VQUERY
```

6.1.1.17 Query by Date-Time range

This query selects the entire content of every booked VEVENT that has an instance greater than or equal to July 1st, 2000 00:00:00 UTC and less than or equal to July 31st, 2000 23:59:59 UTC. This includes single instance VEVENT objects that do no explicitly contain a RECURRENCE-ID.

```
BEGIN:VQUERY
EXPAND:TRUE
QUERY:SELECT * FROM VEVENT
 WHERE RECURRENCE-ID >= '20000801T000000Z'
 AND RECURRENCE-ID <= '20000831T235959Z'
 AND STATE() = 'BOOKED'
END:VQUERY
```

6.1.1.18 Query for all Unprocessed Entries

The following example selects the entire contents of all non-booked "VTODO" and "VEVENT" components with their state of 'UNPROCESSED". The default for EXPAND is FALSE, so the recurrence rules will not be expanded.

```
BEGIN:VQUERY
QUERYID:Fetch VEVENT and VTODO iTIP components
QUERY:SELECT * FROM VEVENT WHERE
 STATE() = 'UNPROCESSED'
QUERY:SELECT * FROM VTODO WHERE
 STATE() = 'UNPROCESSED'
END:VQUERY
```

The following example fetches all "VEVENT" and "VTODO" components that are booked from the CS.

```
BEGIN:VQUERY
QUERYID:Fetch All Booked VEVENT and VTODO components
QUERY:SELECT * FROM VEVENT WHERE STATE() = 'BOOKED'
QUERY:SELECT * FROM VTODO WHERE STATE() = 'BOOKED'
END:VQUERY
```

The following fetches the UID for all VEVENT and VTODO components that have been marked for delete).

```
BEGIN:VQUERY
QUERYID:Fetch UIDs of marked for delete VEVENTs and VTODOs
QUERY:SELECT UID FROM VEVENT WHERE STATE() = 'DELETE'
QUERY:SELECT UID FROM VTODO WHERE STATE() = 'DELETE'
END:VQUERY
```

In the examples above they were bunched into groups of similar queries. They could be performed all at once by having all of the QUERY property in one BEGIN/END VQUERY component.

6.1.1.19 Query with Subset of Properties by Date/Time

In this example only the named properties will be selected and all booked and non-booked components will be selected that have a DTSTART from February 1st to February 10th 2000 (in UTC).

```
BEGIN:VQUERY
QUERY:SELECT UID,DTSTART,DESCRIPTION,SUMMARY FROM VEVENT
 WHERE DTSTART >= '20000201T000000Z'
 AND DTSTART <= '20000210T235959Z'
END:VQUERY
```

6.1.1.20 Query with Components and Alarms In A Range

This example fetches all booked "VEVENT" components with an alarm that triggers within the specified time range. In this case only the "UID", "SUMMARY", and "DESCRIPTION" properties will be selected for all booked "VEVENTS" components that have an alarm between the two date-times supplied.

```
BEGIN:VQUERY
EXPAND:TRUE
QUERY:SELECT UID,SUMMARY,DESCRIPTION FROM VEVENT
 WHERE VALARM.TRIGGER >= '20000101T030405Z'
 AND VALARM.TRIGGER <= '20001231T235959Z'
 AND STATE() = 'BOOKED'
END:VQUERY
```

6.1.2 UPN Value Type

Value Name: UPN

Purpose: This value type is used to identify values that contain user principal name of CU or group of CU.

Formal Definition: The value type is defined by the following notation:

```
upn        = "@"
           / [ dot-atom-text ] "@" dot-atom-text

             ; dot-atom-text is defined in RFC 2822
```

Description: This data type is an identifier that denotes a CU or a group of CU.

Example:

The following is a UPN for a CU:

```
jdoe@acme.com
```

The following is a UPN for a group of CU:

staff@acme.com

The following is a UPN for an anonymous CU belonging to a specific realm:

@acme.com

The following is a UPN for an anonymous CU:

@

6.1.3 UPN-FILTER Value

Value Name: UPN-FILTER

Purpose: This value type is used to identify values that contain a user principal name filter.

Formal Definition: The value type is defined by the following notation:

```
upn-filter    = "OWNER()" /
                "NOT OWNER()" /
                "*" /
                [ "*" / dot-atom-text ] "@" ( "*" / dot-atom-text )

                ; dot-atom-text is defined in RFC 2822
```

Description: The value is used to match user principal names (UPNs). For "OWNER()" and "NOT OWNER()", see Section 6.1.1.3.

*	Matches all UPNs.
@	Matches the UPN of anonymous CUs belonging to the null realm
@*	Matches the UPN of anonymous CUs belonging to any non-null realm
@realm	Matches the UPN of anonymous CUs belonging to the specified realm.
@	Matches the UPN of non-anonymous CUs belonging to any non-null realm
*@realm	Matches the UPN of non-anonymous CUs belonging to the specified realm
user@realm	Matches the UPN of the specified CU belonging to the specified realm
user@*	Not allowed.

Example: The following are examples of this value type:

```
DENY:NON OWNER()
```

6.2 New Parameter

6.2.1 ENABLE Parameter

Parameter Name: ENABLE

Purpose: This parameter indicates whether or not the "TRIGGER" property in a "VALARM" component should be ignored.

Value Type: BOOLEAN

Conformance: This property can be specified in the "TRIGGER" properties.

Description: When a non owner sends an iTIP "REQUEST" to a calendar that object might contain a "VALARM" component. The owner may wish to have local control over their own CUA and when or how alarms are

triggered.

A CUA may add the "ENABLE" parameter to any "TRIGGER" property before booking the component. If the "ENABLE" parameter is set to "FALSE", then the alarm will be ignored by the CUA. If set to "TRUE", or of the "ENABLE" property is not in the "TRIGGER" property, the alarm is enabled. The CUA should remove the "ENABLE" parameter before forwarding the component to a non-cap CUA.

If FALSE in the "VCALSTORE", then all "VAGENDA" ALLOW-CONFLICT values MUST BE false in the CS.

Format Definition: The property is defined by the following notation:

```
allow-conflict     = "ALLOW-CONFLICT"
                     *(";" xparam) ":" boolean CRLF
```

Example: The following is an example of this property for a "VAGENDA" component:

```
ALLOW-CONFLICT:FALSE
```

6.2.2 LOCAL Parameter

Parameter Name: LOCAL

Purpose: Indicates if the VALARM should be exported to any non-organizer calendar.

Value Type: BOOLEAN

Conformance: This property can be specified in the "SEQUENCE" properties in a "VALARM" component.

Description: When a non owner sends an iTIP "REQUEST" to a calendar that object might contain a "VALARM" component. The owner may wish to have local control over their own CUA and when or how alarms are triggered.

A CUA may add the "LOCAL" parameter to the "SEQUENCE" property before booking the component. If the "LOCAL" parameter is set to "FALSE", then the alarm MUST NOT be forwarded to any non organizer calendar. If set to "TRUE", or of the "LOCAL" property is not in the "SEQUENCE"

property, the alarm is global. The CUA should remove the "LOCAL" parameter before forwarding the component to a non-cap CUA and to non organizer calendars.

If FALSE in the "VCALSTORE", then all "VAGENDA" ALLOW-CONFLICT values MUST BE false in the CS.

Format Definition: The property is defined by the following notation:

```
allow-conflict     = "ALLOW-CONFLICT"
                     *(";" xparam) ":" boolean CRLF
```

Example: The following is an example of this property for a "VAGENDA" component:

```
ALLOW-CONFLICT:FALSE
```

7. New Properties

7.1 ALLOW-CONFLICT Property

Property Name: ALLOW-CONFLICT

Purpose: This property indicates whether or not the calendar and CS supports component conflicts. That is, whether or not any of the components in the calendar can overlap.

Value Type: BOOLEAN

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VAGENDA" and "VCALSTORE" component.

Description: This property is used to indicate whether components may conflict. That is, if their expanded instances may share the same time or overlap the same time periods. If it has a value of TRUE, then conflicts are allowed. If FALSE, the no two components may conflict.

If FALSE in the "VCALSTORE", then all "VAGENDA" ALLOW-CONFLICT values MUST BE false in the CS.

Format Definition: The property is defined by the following notation:

```
allow-conflict     = "ALLOW-CONFLICT"
                     *(";" xparam) ":" boolean CRLF
```

Example: The following is an example of this property for a "VAGENDA" component:

```
ALLOW-CONFLICT:FALSE
```

7.2 CALID Property

Property Name: CALID

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in the "VAGENDA".

Description: This property is used to specify a fully qualified calid.

Format Definition: The property is defined by the following notation:

```
CALID   = "CALID" *(";" xparam) ":" calid CRLF
```

Example:

```
CALID:cap://cal.example.com/sdfifgty4321
```

7.3 CALMASTER Property

Property Name: CALMASTER

Purpose: The property specifies an e-mail address of a person responsible for the calendar store.

Value Type: URI

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: The property can be specified in a "VCALSTORE" component.

Description: The parameter value SHOULD BE a MAILTO URI as defined in [RFC1738]. It MUST BE a contact URI such as a MAILTO URI and not a home page or file URI that describes how to contact the calmasters.

Format Definition: The property is defined by the following notation:

```
calmaster = "CALMASTER" *(";" xparam) ":" uri CRLF

uri = IANA registered uri and defined by RFC 2445
```

Example: The following is an example of this property:

```
CALMASTER:mailto:administrator@example.com
```

7.4 CARID Property

Property Name: CARID

Purpose: This property specifies the identifier for an access right component.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property MUST BE specified once in a "VCAR" component.

Description: This property is used in the "VCAR" component to specify an identifier.

Format Definition: The property is defined by the following notation:

```
carid      = "CARID" *(";" xparam) ":" text CRLF
```

Example: The following are examples of this property:

```
CARID:xyzzy-007
CARID:User Rights
```

7.5 CSID Property

Property Name: CSID

Purpose: The property specifies a the globally unique identifier for the calendar store.

Value Type: URI

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: The property can be specified in a "VCALSTORE" component.

Description: The identifier MUST BE globally unique.

Format Definition: The property is defined by the following notation:

```
csid = "CSID" *(";" xparam) ":" capurl CRLF
```

Example: The following is an example of this property:

```
CSID:cap://calendar.example.com
```

7.6 DECREED Property

Property Name: DECREED

Purpose: This property specifies if an access right calendar component is decreed or not.

Value Type: BOOLEAN

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property MAY be specified once in a "VCAR" component.

Description: This property is used in the "VCAR" component to specify whether the component is decreed or not.

Format Definition: The property is defined by the following notation:

```
decreed      = "DECREED" *(";" xparam) ":" boolean CRLF
```

Example: The following is an example of this property:

```
DECREED:TRUE
```

7.7 DEFAULT-CHARSET Property

Property Name: DEFAULT-CHARSET

Purpose: This property indicates the default charset.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VAGENDA" and "VCALSTORE" calendar component.

Description: In a "VAGENDA", this property is used to indicate the charset of calendar. If not specified, the default the first value in the "VCALSTORE" DEFAULT-CHARSET list. The value MUST BE an IANA registered character set as defined in [RFC 2278].

In a "VCALSTORE" it is a comma separated list of charsets supported by the CS. The first entry is the default entry for all newly created "VAGENDA"s. "UTF-8" MUST BE in the "VCALSTORE" DEFAULT-CHARSET list.

If a charset name contains a comma (,), then that comma must be backslashed escaped in the value.

Format Definition: The property is defined by the following notation:

```
default-charset     = "DEFAULT-CHARSET" *(";" xparam)
                      ":" text CRLF
```

Example: The following is an example of this property for a "VAGENDA" component:

```
DEFAULT-CHARSET:Shift_JIS,UTF-8
```

7.8 DEFAULT-LOCALE Property

Property Name: DEFAULT-LOCALE

Purpose: This property specifies the default language for text values.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VAGENDA" and "VCALSTORE" components.

Description: In a "VAGENDA", this property is used to indicate the locale of the calendar. The full locale SHOULD be used. The default and minimum locale is POSIX.

In a "VCALSTORE" it is a comma separated list of locales supported by the CS. The first value in the list is the default for all newly created VAGENDAs. POSIX MUST BE in the list.

Format Definition: The property is defined by the following notation:

```
default-locale     = "DEFAULT-LOCALE" *(";" xparam)
                     ":" language CRLF

language = Text identifying a locale, as defined in [RFC 2277]
```

Example: The following is an example of this property:

```
DEFAULT-LOCALE:en-US.iso-8859-1,POSIX
```

7.9 DEFAULT-TZID Property

Property Name: DEFAULT-TZID

Purpose: This property specifies the text value that specifies the default time zone for a calendar.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property may be specified once in a "VAGENDA" and "VCALSTORE" components.

Description: In a "VAGENDA" it is the value of the time zone for the

calendar. This time zone is used when as the localtime for object that contain a date or date-time value without a time zone.

In a "VCALSTORE" it is a comma separated list of TZIDs known to the CS. Where TZID values are the same as the TZID property as defined in [iCAL]. The first entry in the list is used as the default TZID for all newly created calendars. The list MUST contain at least UTC.

If the TZID contains a comma (,), the comma must be backslash escaped.

Format Definition: This property is defined by the following notation:

```
default-tzid       = "DEFAULT-TZID" *(";" xparam)
                     ":" [tzidprefix] text CRLF
```

Example: The following is an example of this property:

```
DEFAULT-TZID:US/Eastern,UTC
```

7.10 DEFAULT-VCARS Property

Property Name: DEFAULT-VCARS

Purpose: This property is used to specify the CARIDs of the default VCAR components for newly created VAGENDA components.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property MUST BE specified in "VCALSTORE" calendar component and MUST at least specify the following values: READBUSYTIMEINFO, REQUESTONLY, UPDATEPARTSTATUS, and DEFAULTOWNER.

Description: This property is used in the "VCALSTORE" calendar component to specify the CARID of the VCAR components that MUST BE copied in VAGENDA at creation time by the CS. These VCARS components MUST BE stored in the "VCALSTORE".

Format Definition: The property is defined by the following notation:

```
def-vcars      = "DEFAULT-VCARS" *(";" xparam) ":" text
                 *( "," text ) CRLF
```

Example: The following is an example of this property:

```
DEFAULT-VCARS:READBUSYTIMEINFO,REQUESTONLY,
 UPDATEPARTSTATUS,DEFAULTOWNER
```

7.11 DENY Property

Property Name: DENY

Purpose: This property identifies the UPN(s) being denied access in the VRIGHT component.

Value Type: UPN-FILTER

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VRIGHT" calendar components.

Description: This property is used in the "VRIGHT" component to define the CU or UG being denied access.

Format Definition: The property is defined by the following notation:

```
deny       = "DENY" *(";" xparam) ":" upn-filter CRLF
```

Example: The following are examples of this property:

```
DENY:*

DENY:bob@example.com
```

7.12 EXPAND property

Property Name: EXPAND

Purpose: This property is to notify the CS if it should or should not expand any component with recurrence rules into multiple instances in a query reply.

Value Type: BOOLEAN

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VQUERY" calendar components.

Description: If a CUA wishes to see all of the instances of a recurring component the CUA sets EXPAND=TRUE in the VQUERY component. If not specified, the default is FALSE.

Format Definition: The property is defined by the following notation:

```
expand     = "EXPAND" *(";" xparam) ":" ("TRUE" / "FALSE") CRLF
```

Example: The following are examples of this property:

```
EXPAND:FALSE
EXPAND:TRUE
```

7.13 GRANT Property

Property Name: GRANT

Purpose: This property identifies the UPN(s) being granted access in the VRIGHT component.

Value Type: UPN-FILTER

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VRIGHT" calendar components.

Description: This property is used in the "VRIGHT" component to specify the CU or UG being granted access.

Format Definition: The property is defined by the following notation:

```
grant     = "GRANT" *(";" xparam) ":" upn-filter CRLF
```

Example: The following are examples of this property:

```
GRANT:*

GRANT:bob@example.com
```

7.14 MAXDATE Property

Property Name: MAXDATE

Purpose: This property specifies the date/time in the future beyond which the CS cannot represent.

Value Type: DATE-TIME

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: The property can be specified in the "VCALSTORE".

Description: The date and time MUST BE a UTC value and end with 'Z'.

Format Definition: The property is defined by the following notation:

```
maxdate    = "MAXDATE" *(";" xparam) ":" date-time CRLF
```

Example: The following is an example of this property:

```
MAXDATE:20990101T000000Z
```

7.15 MINDATE Property

Property Name: MINDATE

Purpose: This property specifies the date/time in the past prior to which the server cannot represent.

Value Type: DATE-TIME

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: The property can be specified in the "VCALSTORE".

Description: The date and time MUST BE a UTC value and end with 'Z'.

Format Definition: The property is defined by the following notation:

```
mindate    = "MINDATE" *(";" xparam) ":" date-time CRLF
```

Example: The following is an example of this property:

```
MINDATE:19710101T000000Z
```

7.16 NAME Property

Property Name: NAME

Purpose: This property provides a localizeable display name for a component.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in a component.

Description: This property is used in the in component to specify a localizeable display name. If more than one NAME property is in a component, then they MUST have unique LANG parameters. If the LANG parameter is not supplied, then it defaults to the VAGENDAs default if the component is in a VAGENDA, or the VCALSTORE default if the

component is stored at the VCALSTORE level.

Format Definition: The property is defined by the following notation:

```
name = "NAME" nameparam ":" text CRLF

nameparam = *(
                ; the following is optional,
                ; but MUST NOT occur more than once

                ( ";" languageparam ) /

                ; the following is optional,
                ; and MAY occur more than once

                ( ";" xparam )
              )

languageparam = ; As defined in [iCAL].
```

Example: The following is an example of this property:

```
NAME:Restrict Guests From Creating VALARMs On VEVENTs
```

7.17 OWNER Property

Property Name: OWNER

Purpose: The property specifies an owner of the component.

Value Type: UPN

Property Parameters: Non-standard, alternate text representation and language property parameters can be specified on this property.

Conformance: The property MUST BE specified at in a "VAGENDA" component.

Description: A multi-instanced property indicating the calendar owner.

Format Definition: The property is defined by the following notation:

```
owner = "OWNER" *(";" xparam) ":" upn CRLF
```

Example: The following is an example of this property:

```
OWNER:jsmith@acme.com
OWNER:jdoe@acme.com
```

7.18 PERMISSION Property

Property Name: PERMISSION

Purpose: This property defines a permission that is granted or denied in a VRIGHT component.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VRIGHT" calendar components.

Description: This property is used in the "VRIGHT" component to define a permission that is granted or denied.

Format Definition: The property is defined by the following notation:

```
perm      = "PERMISSION" *(";" xparam) ":" permvalue CRLF

permvalue = ( "READ" / "CREATE" / "DELETE" / "MODIFY" / all )

all         = "*"
```

Example: The following is an example of this property:

```
PERMISSION:READ
```

7.19 QUERY property

Property Name: QUERY

Purpose: Specifies the query for the component.

Value Type: CAL-QUERY

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VQUERY" components.

Description: A "QUERY" is used to specify the CAL-QUERY (Section 6.1.1 for the query.

Format Definition: The property is defined by the following notation:

```
query      = "QUERY" *(";" xparam) ":" cal-query CRLF
```

Example: The following is an example of this property:

```
QUERY:SELECT * FROM VEVENT
```

7.20 QUERYID property

Property Name: QUERYID

Purpose: Specifies a unique ID for a query in the targeted container.

Value Type: TEXT

Property Parameters: Non-standard property parameters are specified on this property.

Conformance: This property can be specified in "VQUERY" components.

Description: A "QUERYID" is used to specify the unique id for a stored query.

Format Definition: The property is defined by the following notation:

```
queryid      = "QUERYID" *(";" xparam) ":" text CRLF
```

Example: The following are examples of this property:

```
QUERYID:Any Text String
QUERYID:fetchUnProcessed
```

7.21 REQUEST-STATUS property

This description is a revision of the REQUEST-STATUS property for VCALENDAR version 2.0. The 'statdesc' is optional and the 'extdata' may be included when 'statdesc' is not provided.

```
rstatus  = "REQUEST-STATUS" rstatparam ":"
           statcode ";" *(statdesc ) ";" *(extdata)

rstatparam  = *(
           ; the following is optional,
           ; but MUST NOT occur more than once

              (";" languageparm) /

           ; the following is optional,
           ; and MAY occur more than once

              (";" xparam)

             )

  statcode  = 1*DIGIT *("." 1*DIGIT)
              ;Hierarchical, numeric return status code

  statdesc  = text
              ;An optional textual status description, content is
              ;decided by the implementor. May be empty.

   extdata  = text
              ; Textual exception data. For example, the offending
              ; property name and value or complete property line.
```

Example: The following are some possible examples of this property. The COMMA and SEMICOLON separator characters in the property value

are BACKSLASH character escaped because they appear in a text value.

```
REQUEST-STATUS:2.0;Success

REQUEST-STATUS:3.1;Invalid property value;DTSTART:96-Apr-01

REQUEST-STATUS:2.8; Success\, repeating VEVENT ignored. Scheduled
 as a single VEVENT.;RRULE:FREQ=WEEKLY;INTERVAL=2

REQUEST-STATUS:4.1;Time conflict. Date/time is busy.

REQUEST-STATUS:3.7;Invalid calendar user;ATTENDEE:
 MAILTO:jsmith@example.com

REQUEST-STATUS:3.7;;ATTENDEE:MAILTO:jsmith@example.com

REQUEST-STATUS:10.4;Help! That really shouldn't have happened.
```

7.22 RESTRICTION Property

Property Name: RESTRICTION

Purpose: This property defines restrictions on the result value of new or existing components.

Value Type: CAL-QUERY

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VRIGHT" calendar components, but only when the PERMISSION property is set to "CREATE", "MODIFY", or "*".

Description: This property is used in the "VRIGHT" component to define restrictions on the components that can be written (i.e., by using the "CREATE" or "MOVE" commands) as well as on the values that may take existent calendar store properties, calendar properties, components, and properties (i.e., by using the "MODIFY" command). Accepted values MUST match the specified RESTRICTION.

Format Definition: The property is defined by the following notation:

```
restrict      = "RESTRICTION" *(";" xparam) ":" cal-query CRLF
```

Example: The following are examples of this property:

```
RESTRICTION:SELECT * FROM VCALENDAR WHERE METHOD = 'REQUEST'

RESTRICTION:SELECT * FROM VEVENT WHERE
 SELF() IN CAL-OWNERS(ORGANIZER)

RESTRICTION:SELECT * FROM VEVENT WHERE 'BUSINESS' IN
 CATEGORIES
```

7.23 SCOPE Property

Property Name: SCOPE

Purpose: This property identifies the objects in the CS to which the access rights applies.

Value Type: CAL-QUERY

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in "VRIGHT" calendar components.

Description: This property is used in the "VRIGHT" component to define the set of objects subject to the access right being defined.

Format Definition: The property is defined by the following notation:

```
scope      = "SCOPE" *(";" xparam) ":" cal-query CRLF
```

Example: The following is an example of this property:

```
SCOPE:SELECT DTSTART,DTEND FROM VEVENT WHERE CLASS = 'PUBLIC'
```

7.24 TARGET Property

Property Name: TARGET

Purpose: This property defines the container that the command that is issued will act upon. It its value is a capurl as defended in Section 5.

Value Type: URI

Property Parameters: Non-standard property parameters can be specified on this property.

Conformance: This property can be specified in a command component.

Description: This properties value is used to specify the container that the command will effect. When used in a command, the command will be performed on the container which has a capurl matching the value.

Format Definition: The property is specified by the following notation:

```
target   = "TARGET" *(";" xparam) ":" capurl CRLF
```

The following is an example of this property:

```
TARGET:cap://mycal.example.com
TARGET:SomeRelCalid
```

7.25 TRANSP Property

Property Name: TRANSP

Purpose: This property defines whether an component is transparent or not to busy time searches. This is a modification to [iCAL] TRANSP in that it adds some values.

Value Type: TEXT

Property Parameters: Non-standard property parameters can be

specified on this property.

Conformance: This property can be specified in a component.

Description: Time Transparency is the characteristic of an object that determines whether it appears to consume time on a calendar. Objects that consume actual time for the individual or resource associated with the calendar SHOULD be recorded as OPAQUE, allowing them to be detected by free-busy time searches. Other objects, which do not take up the individual's (or resource's) time SHOULD be recorded as TRANSPARENT, making them invisible to free-busy time searches.

Format Definition: The property is specified by the following notation:

```
transp   = "TRANSP" *(";" xparam) ":" transvalue CRLF

transvalue   = "OPAQUE" ;Blocks or opaque on busy time searches.
             / "TRANSPARENT"    ;Transparent on busy time searches.

             / "TRANSPARENT-NOCONFLICT" ; Transparent on busy time
             ; searches and no other OPAQUE or OPAQUE-NOCONFLICT objects
             ; can overlap it.

             / "OPAQUE-NOCONFLICT"  ; Opaque on busy time
             ; searches and no other OPAQUE or OPAQUE-NOCONFLICT objects
             ; can overlap it.
             ;
             ;Default value is OPAQUE
```

The following is an example of this property for an object that is opaque or blocks on free/busy time searches plus no other object can overlap it:

```
    TRANSP:OPAQUE-NOCONFLICT
```

8. New Components

8.1 VAGENDA Component

Component Name: VAGENDA

Purpose: Provide a grouping of properties that defines an agenda.

Formal Definition: There are two formats of a VAGENDA. (1) When it is being created, and (2) how it exists in the VCALSTORE. A "VAGENDA" component in the VCALSTORE is defined by the following notation table and ABNF notation.

The following is a summary of the properties of a calendar.

Name	Read Only	Description
ALLOW-CONFLICT	N	This boolean value indicates whether or not the calendar supports object conflicts. That is, whether or not any of the components in the calendar can have a time overlap. MUST BE FALSE if VCALSTORE ALLOW-CONFLICT is FALSE.
CALID	N	A unique identifier within the VCALSTORE for the calendar. MUST NOT BE empty. MUST BE a relative calid in a VAGENDA.
CALSCALE	N	The CALSCALE for this calendar. MUST BE from the VCALSTORE CALSCALE list. The default is the first entry in the VCALSTORE CALSCALE list.
CREATED	Y	timestamp of the calendar's create date.
DEFAULT-CHARSET	N	The charset for this calendar. MUST BE from the VCALSTORE DEFAULT-CHARSET list. If empty then it is the first entry in the VCALSTORE DEFAULT-CHARSET list.
DEFAULT-LOCALE	N	The locale for this calendar. MUST BE from the VCALSTORE DEFAULT-LOCALE list. If empty then it is the first entry in the VCALSTORE DEFAULT-CHARSET list.

DEFAULT-TZID	N	The id of the timezone associated with this calendar. If empty it is the first entry in VCALSTORE DEFAULT-TZID.
LAST-MODIFIED	Y	The timestamp when the properties of the calendar were last updated.
NAME	N	Optional display name for this calendar. It is a localizeable string. May be multiple instance and no two instances may have the same LANG parameter. All instances MUST have the LANG parameter.
OWNER	N	A multi-instanced property indicating the calendar owner. Each entry returned will be a UPN. There MUST BE at least one owner.
RELATED-TO	N	An optional multi-instance property indicating any relationship to other CALIDs and their CALIDs.

```
  agenda      = "BEGIN" ":" "VAGENDA" CRLF
                 agendaprop
                "END" ":" "VAGENDA" CRLF

  agendaprop  = *(
              ; The following MUST occur exactly once.

                allow-conflict / calid / calscale / created
              / default-charset / default-locale
              / default-tzid / last-modified /

              ; The following MUST occur at least once.
              ; and the value MUST NOT be empty.

              / owner

              ; The following are optional,
              ; and MAY occur more than once.

              / name / related-to / iana-token / x-prop / x-comp
              )
```

When creating a VAGENDA, use the following notation:

```
agendac     = "BEGIN" ":" "VAGENDA" CRLF
               agendacprop
              "END" ":" "VAGENDA" CRLF

agendacprop  = *(
             ; The following MUST occur exactly once.

               allow-conflict / calid / calscale
             / default-charset / default-locale
             / default-tzid /

             ; The following MUST occur at least once.
             ; and the value MUST NOT be empty.

             / owner

             ; The following are optional,
             ; and MAY occur more than once.

             / name / related-to / iana-token / x-prop / x-comp
             )
```

To fetch all of the properties from the targeted VAGENDA. This does not fetch any components:

```
SELECT * FROM VAGENDA
```

To fetch all of the properties from the targeted VAGENDA and all of the contained components, use the special '*.*' value:

```
SELECT *.* FROM VAGENDA
```

8.2 VCALSTORE Component

Component Name: VCALSTORE

Purpose: Provide a grouping of properties that defines a calendar store.

Formal Definition: A "VCALSTORE" component is defined by the

following table and ABNF notation. The creation of a VCALSTORE is an administrative task and not part of the CAP protocol.

The following is a summary of the properties of the calendar store.

Name	Read Only	Description
ALLOW-CONFLICT	Y	This boolean value indicates Whether or not the VCALSTORE supports object conflicts. That is, whether or not any of the objects in any calendar can overlap. If FALSE, then the CS does not allow conflicts for any entry in any calendar. How this property is set in the VCALSTORE is an administrative or implementation specific issue and is not covered in CAP.
CALSCALE	Y	A comma separated list of CALSCALEs supported by this CS. All Calendar CALSCALE properties MUST BE from this list. MUST contain at least "GREGORIAN". The default for newly created calendars is the first entry. How this property is set in the VCALSTORE is an administrative or implementation specific issue and is not covered in CAP.
CALMASTER	N	URL of contact address for person responsible. SHOULD BE mailto URL. MUST BE an IANA registered URL scheme. This is to allow external entities a contact point for the CS.
CHILDREN	N	A multi instance property that lists all of the calendars in this VCALSTORE. The values are the relative CSID for each calendar.
CREATED	Y	The timestamp of the CS creation time.
CSID	Y	The CSID of this calendar store. MUST NOT be empty. How this property is set in the VCALSTORE is an administrative or implementation specific issue and is not covered in CAP.
DEFAULT-CHARSET	Y	A comma separated lists of charsets supported by this CS. MUST contain at least "UTF-8". The first is the default for all newly created calendars. How this property is set in the VCALSTORE is an administrative or implementation

		specific issue and is not covered in CAP.
DEFAULT-LOCALE	Y	A comma separated list of locales supported by this CS. MUST contain at least one entry. ("en_US" for example). The first is the default for all newly created calendars. There is no default for this property in the VCALSTORE.
DEFAULT-VCARS	N	A multivalued property containing the CARID's for the default VCARs for newly created top level calendars. It MUST include at a minimum READBUSYTIMEINFO, REQUESTONLY, UPDATEPARTSTATUS, and DEFAULTOWNER.
DEFAULT-TZID	N	A comma separated list of TZID's supported by the CS. These will be known across all calendars. Calendar entries take precedence if they exist in both the CS and calendar. MUST include at least UTC. First entry is default for all newly created calendars.
LAST-MODIFIED	Y	The timestamp when the Properties of the CS were last updated or calendars were created or deleted.
NAME	N	Optional, multi instance display names for this CS. It is a localizeable string. May be multiple instance and no two instances may have the same LANG parameter. All instances MUST have the LANG parameter in the VCALSTORE.
RELATED-TO	N	An optional multi-instance property indicating any relationship to other CSs and those CALIDs.

```
calstorec     = "BEGIN" ":" "VCALSTORE" CRLF
                calstoreprop
                "END" ":" "VCALSTORE" CRLF

calstoreprop  = *(

                  ; the following MUST occur exactly once

                    allow-conflict / calscale / calmaster
                  / created / csid / default-charset
                  / default-locale / default-vcars
                  / default-tzid / last-modified

                  ; the following are optional,
                  ; and MAY occur more than once

                  / children / name / related-to

                 )
```

To fetch all of the properties from the targeted VCALSTORE and not fetch the calendars that it contains:

```
SELECT * FROM VCALSTORE
```

To fetch all of the properties from the targeted VCALSTORE and all of the contained calendars and all of those calendars contained properties and components, use the special '*.*' value:

```
SELECT *.* FROM VCALSTORE
```

8.3 VCAR Component

Component Name: VCAR

Purpose: Provide a grouping of calendar access rights.

Format Definition: A "VCAR" component is defined by the following notation:

```
carc    =  "BEGIN" ":" "VCAR" CRLF
           carprop 1*rightc
           "END" ":" "VCAR" CRLF

carprop = 1*(

        ; 'carid' is REQUIRED,
        ; but MUST NOT occur more than once

        carid /

        ; the following are OPTIONAL,
        ; and MAY occur more than once

        name / x-prop / iana-prop
        )
```

Description: A "VCAR" component is a grouping of properties, and "VRIGHT" components, that represents access rights granted or denied to UPNs.

The "CARID" property specifies the local identifier for the "VCAR" component. The "NAME" property specifies a localizeable display name.

Example: In the following example, the UPN "foo@example.com" is given read access to the "DTSTART" and "DTEND" VEVENT properties. No other access is specified:

```
BEGIN:VCAR
CARID:Veiw Start and End Times
NAME:View Start and End Times
BEGIN:VRIGHT
GRANT:foo@example.com
PERMISSION:READ
SCOPE:SELECT DTSTART,DTEND FROM VEVENT
END:VRIGHT
END:VCAR
```

In this example, all UPNs are given read access to "DTSTART" and "DTEND" properties of VEVENT components. "All CUs and UGs" are specified by the UPN value "*". Note that this enumerated UPN value is not in quotes:

```
BEGIN:VCAR
CARID:ViewStartEnd2
NAME:View Start and End Times 2
BEGIN:VRIGHT
GRANT:*
PERMISSION:READ
SCOPE:SELECT DTSTART,DTEND FROM VEVENT
END:VRIGHT
END:VCAR
```

In these examples, full calendar access rights are given to the OWNER(), and a hypothetical administrator is given access rights to specify calendar access rights. If no other rights are specified, only these two UPNs can specify calendar access rights:

```
BEGIN:VCAR
CARID:some-id-3
NAME:Only OWNER or ADMIN Settable VCARs
BEGIN:VRIGHT
GRANT:OWNER()
PERMISSION:*
SCOPE:SELECT * FROM VAGENDA
END:VRIGHT
BEGIN:VRIGHT
GRANT:cal-admin@example.com
PERMISSION:*
SCOPE:SELECT * FROM VCAR
RESTRICTION:SELECT * FROM VCAR
END:VRIGHT
END:VCAR
```

In this example, rights to write, read, modify or delete calendar access rights are denied to all UPNs. This example would disable providing different access rights to the calendar store or calendar. This calendar access right should be specified with great care, as it removes the ability to change calendar access; even for the owner or administrator. It could be used by small devices that do not support the changing of any VCAR:

```
BEGIN:VCAR
CARID:VeryRestrictiveVCAR-2
NAME:No CAR At All
BEGIN:VRIGHT
DENY:*
PERMISSION:*
SCOPE:SELECT * FROM VCAR
END:VRIGHT
END:VCAR
```

8.4 VRIGHT Component

Component Name: "VRIGHT"

Purpose: Provide a grouping of properties that describe an access right (granted or denied).

Format Definition: A "VRIGHT" component is defined by the following notation:

```
rightc    =  "BEGIN" ":" "VRIGHT" CRLF
             rightprop
             "END" ":" "VRIGHT" CRLF

rightprop = 2*(

          ; either 'grant' or 'deny' MUST
          ; occur at least once
          ; and MAY occur more than once

          grant / deny /

          ; 'permission' MUST occur at least once
          ; and MAY occur more than once

          permission /

          ; the following are optional,
          ; and MAY occur more than once

          scope / restriction / x-prop / iana-prop

          )
```

Description: A "VRIGHT" component is a grouping of calendar access right properties.

The "GRANT" property specifies the UPN that is being granted access. The "DENY" property specifies the UPN is being denied access. The "PERMISSION" property specifies the actual permission being set. The "SCOPE" property identifies the calendar store properties, calendar properties, components, or properties to which the access right applies. The "RESTRICTION" property specifies restriction on the value that may take calendar store properties, calendar properties, calendar components, and properties after a CREATE or MODIFY operation. Values MUST match all the instances of the RESTRICTION property to be valid.

8.5 VREPLY Component

Component Name: "VREPLY"

Purpose: Provide a grouping of arbitrary properties and components that are the data set result from an issued command.

Format Definition: A "VREPLY" component is defined by the following notation:

```
replyc    =  "BEGIN" ":" "VREPLY" CRLF
              any-prop-or-comp
             "END" ":" "VREPLY" CRLF

any-prop-or-comp = ; Zero or more iana or experimental
                   ; properties and components, in any order.
```

Description: Provide a grouping of arbitrary properties and components that are the data set result from an issued command.

A query can return a predictable set of arbitrary properties and components. This container is used by query and other commands to return data that does not fit into any other container. It may contain any valid property or component, even if they are not registered.

8.6 VQUERY Component

Component Name: VQUERY

Purpose: A container to specify what is to be fetched from a CS.

Format Definition: A "VQUERY" component is defined by the following notation:

```
queryc    =  "BEGIN" ":" "VQUERY" CRLF
             queryprop
             "END" ":" "VCAR" CRLF

queryprop = 1*(

             ; 'queryid' is OPTIONAL but MUST NOT occur
             ; more than once
             ;
             queryid

             ; the following are OPTIONAL, and MAY occur
             ; more than once
             ;
             / name / x-prop / iana-prop

             ; the following MUST occur at least once.
             ;
             / query

            )
```

Description: A "VQUERY" contains properties that specify which properties and components the CS is requested to return during a SEARCH command.

The "QUERYID" property specifies the local identifier for a stored "VQUERY" component. The "NAME" property specifies a localizeable display name of a stored "VQUERY" component. Normally "NAME" and "QUERYID" are used when looking for a correct stored "VQUERY" component, or when storing a "VQUERY" component.

For examples, see Section 6.1.1.

9. Commands and Responses

CAP commands and responses are described in this section.

9.1 CAP Commands (CMD)

All commands are send using the CMD property.

Property Name: CMD

Purpose: The property defines the command to be sent.

Value Type: TEXT

Property Parameters: Non-standard, id, localize, latency, action or options.

Conformance: This property is the method used to specify the commands to a CS and can exist in any object sent to the CS.

Description: All of the command to the CS are supplied in this property. The OPTIONS parameter is overloaded and its meaning is dependent on the CMD value supplied.

Format Definition: The property is defined by the following notation:

```
cmd             = "CMD" (
                / abort-cmd
                / continue-cmd
                / create-cmd
                / delete-cmd
                / generate-uid-cmd
                / get-capability-cmd
                / identify-cmd
                / modify-cmd
                / move-cmd
                / reply-cmd
                / search-cmd
                / set-locale-cmd
                ) CRLF


id-param        = ";" "ID" "=" unique-id
                ; The text value supplied is a unique value
                ; shared between the CUA and CS to uniquely
                ; identify the instance of command in the
                ; the current CUA session. The value has
```

```
                    ; no meaning to other CUAs or other sessions.

     unique-id      = ; text


     localize-param = ";" "LOCALIZE" "=" beep-localize
                    ; The value supplied MUST BE one value from the initial
                    ; BEEP greeting 'localize' attribute specifying
                    ; the locale to use for error messages during
                    ; this instance of the command sent.

     beep-localize  = ; text

     latency-param  = ";" "LATENCY" "=" latency-sec
                    ; The value supplied in the time in seconds.
                    ; If latency is supplied then action MUST BE
                    ; supplied.

     latency-sec    = integer

     action-parm    = ";" "ACTION" "=" ( "ASK" / "ABORT" )
                    ; If latency is supplied then action MUST BE
                    ; supplied.

     option-param   = ";" "OPTIONS" "=" cmd-specific

     cmd-specific   = ; The value supplied is dependent on the
                      ; CMD value. See the specific CMDs below
                      ; for the correct values to use for each
                      ; CMD.

     option-value   = paramtext
```

Calendaring commands allow a CUA to directly manipulate a calendar.

Calendar access rights can be granted or denied for any commands.

9.1.1 Bounded Latency

A CAP command can have an associated maximum latency time by specifying the "LATENCY" parameter. If the command is unable to be completed in the specified amount of time (as specified by the "LATENCY" parameter value), then a "TIMEOUT" command MUST BE sent on the same channel to which there MUST BE a an "ABORT" or a "CONTINUE" command reply. If the CUA initiated the original command, then the CS would issue the "TIMEOUT" command and the CUA would then have to issue an "ABORT" or "CONTINUE" command. If the CS initiated the

original command then the CUA would have to issue the "TIMEOUT" and the CS would send the "ABORT" or "CONTINUE".

Upon receiving an "ABORT" command, the command must then be terminated. Only the "ABORT", "TIMEOUT", "REPLY, and "CONTINUE" commands can not be aborted. The "ABORT", "TIMEOUT", and "REPLY" commands MUST NOT have latency set.

Upon receiving a "CONTINUE" command the work continues. Note that a new latency time MAY BE included in a "CONTINUE" command indicating to continue the original command until the "LATENCY" parameter value expires or the results of the original command can be returned.

Both the "LATENCY" parameter and the "ACTION" parameter MUST BE supplied to any "CMD" property, or nether can be added to the "CMD" property. The "LATENCY" parameter MUST BE set to the maximum latency time in seconds. The "ACTION" parameter accepts the following values: "ASK" and "ABORT".

If the maximum latency time is exceeded and the "ACTION" parameter is set to the "ASK" value, then "TIMEOUT" command MUST BE sent. Otherwise if the "ACTION" parameter is set to the "ABORT" value then the command MUST BE terminated and return a REQUEST-STATUS code of 2.0.3 for the original command.

Example:

In this example the initiator asks for the listeners capabilities.

```
I: Content-Type: text/calendar
I:
I: BEGIN:VCALENDAR
I: VERSION:2.0
I: PRODID:The CUA's PRODID
I: CMD;ID=xyz12346:GET-CAPABILITY
I: END:VCALENDAR

# After 3 seconds

L: Content-Type: text/calendar
L:
L: BEGIN:VCALENDAR
L: VERSION:2.0
L: CMD;ID=xyz12346:TIMEOUT
L: END:VCALENDAR
```

In order to continue and give the CS more time then the CUA would issue a "CONTINUE" command:

```
I: Content-Type: text/calendar
I:
I: BEGIN:VCALENDAR
I: VERSION:2.0
I: CMD;ID=xyz12346;LATENCY=3;ACTION=ask:CONTINUE
I: END:VCALENDAR

L: Content-Type: text/calendar
L:
L: BEGIN:VCALENDAR
L: VERSION:2.0
L: CMD;ID=xyz12346:REPLY
L: BEGIN:VREPLY
L: REQUEST-STATUS:2.0.3;Continued for 3 more seconds
L: END:VREPLY
L: END:VCALENDAR
```

To abort the command and not wait any further then issue an "ABORT" command:

```
I: Content-Type: text/calendar
I:
I: BEGIN:VCALENDAR
I: VERSION:2.0
I: CMD;ID=xyz12346:ABORT
I: END:VCALENDAR

# Which would result in a 2.0.3 reply.

L: Content-Type: text/calendar
L:
L: BEGIN:VCALENDAR
L: VERSION:2.0
L: CMD;ID=xyz12346:REPLY
L: BEGIN:VREPLY
L: REQUEST-STATUS:2.0.3;Aborted As Requested.
L: END:VREPLY
L: END:VCALENDAR
```

9.1.2 ABORT Command

CMD: ABORT

Purpose: The "ABORT" command is sent to request that the named or only in process command be aborted. Latency MUST not be supplied with the "ABORT" command.

Formal Definition: An "ABORT" command is defined by the following notation:

```
 abort-cmd   = abortparam ":" "ABORT"

abortparam   = *(

             ; the following are optional,
             ; but MUST NOT occur more than once

               id-param
             / localize-param

             ; the following is optional,
             ; and MAY occur more than once

             / (";" xparam)

             )
```

The REPLY of any "ABORT" command is:

```
abort-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                calprops
                abort-vreply
               "END" ":" "VCALENDAR" CRLF

abort-vreply  = "BEGIN" ":" "VREPLY" CRLF
                 request-status
                *(x-prop)
                "END" ":" "VREPLY" CRLF
```

9.1.3 CONTINUE Command

CMD: CONTINUE

Purpose: The "CONTINUE" command is only sent after a "TIMEOUT" command has been received to inform the other end of the session to resume working on a command.

Formal Definition: A "CONTINUE" command is defined by the following notation:

```
      continue-cmd   = continueparam ":" "CONTINUE"

     continueparam   = *(

                     ; the following are optional,
                     ; but MUST NOT occur more than once

                       id-param
                     / localize-param
                     / latency-param

                     ; the following MUST occur exactly once and only
                     ; when the latency-param has been supplied and
                     ; MUST NOT be supplied if the latency-param is
                     ; not supplied.

                     / action-param

                     ; the following is optional,
                     ; and MAY occur more than once

                     / (";" xparam)

                     )
```

The REPLY of any "CONTINUE" command is:

```
       continue-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                         calprops
                         continue-vreply
                        "END" ":" "VCALENDAR" CRLF

      continue-vreply  = "BEGIN" ":" "VREPLY" CRLF
                          request-status
                         *(x-prop)
                         "END" ":" "VREPLY" CRLF
```

9.1.4 CREATE Command

CMD: CREATE

Purpose: The "CREATE" command is used to create one or more iCalendar objects in the store in the "BOOKED" or "UNPROCESSED" state.

A CUA MAY send a CREATE command to a CS. The CREATE command MUST BE implemented by all CSs.

The CS MUST NOT send a CREATE command to any CUA.

Formal Definition: A "CREATE" command is defined by the following notation:

```
     create-cmd     = createparam ":" "CREATE"

     createparam    = *(

                      ; the following are optional,
                      ; but MUST NOT occur more than once

                        id-param
                      / localize-param
                      / latency-param

                      ; the following MUST occur exactly once and only
                      ; when the latency-param has been supplied and
                      ; MUST NOT be supplied if the latency-param is
                      ; not supplied.
n
                      / action-param

                      ; the following is optional,
                      ; and MAY occur more than once

                      / (";" xparam)

                      )
```

Response:

One iCalendar object per TARGET property MUST BE returned.

The REPLY of any "CREATE" command is:

```
create-reply   = "BEGIN" ":" "VCALENDAR" CRLF
                 calprops
                 1*(create-vreply)
                "END" ":" "VCALENDAR" CRLF

create-vreply  = "BEGIN" ":" "VREPLY" CRLF
                  created-id
                  request-status
                 *(x-prop)
                 "END" ":" "VREPLY" CRLF

                 ; Where the id is appropriate for the
                 ; type of object created:
                 ;
                 ; VAGENDA = calid
                 ; VCAR = carid
                 ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
                 ; VQUERY = queryid
                 ; ALARM = sequence
                 ; x-component = x-id
                 ;
 created-id    = ( calid / carid / uid / uid dtstamp
                   / queryid / tzid / sequence / x-id)
```

The TARGET property specify the containers where the component(s) will be created. This can be a CSID, or a CALID.

The iCalendar portion of the command can be any valid [iTIP] object or any valid CAP object using the following restriction table. There MUST BE at least one component inside of the VCALENDAR object.

If the iCalendar object being created does not have a "METHOD" property, then is not an iTIP object, then its state will be "BOOKED". Use the "DELETE" command to set the state of an object to the "DELETED" state. A CUA can not use the "CREATE" command to create an object in the "DELETED" state.

If an iTIP object is being booked, then the "METHOD" property MUST NOT BE supplied". Otherwise any iTIP object MUST BE have valid iTIP METHOD value and it is a scheduling request being deposited into the CS and will have its state set to "UNPROCESSED".

Restriction table for the CREATE command:

```
create-minumum = "BEGIN" ":" "VCALENDAR" CRLF
                 calprops
                 *(iana-prop)
                 *(x-prop)
                 1*(create-comp)
                 "END" ":" "VCALENDAR" CRLF

                 ; If The following  contain the "METHOD"
                 ; property they MUST conform to [iTIP].
                 ;
  create-comp = agendac / carc / queryc
                 / timezonec / freebusyc
                 / eventc / todoc / journalc
                 / iana-component / x-component
```

Restriction Table for the iCalendar section of a reply that contains an iCalendar object is any valid [iTIP] response plus any from this restriction table:

```
    create-reply = "BEGIN" ":" "VCALENDAR" CRLF
                   calprops
                   *(iana-prop)
                   *(x-prop)
                   1*(create-comp-vreply)
                   "END" ":" "VCALENDAR" CRLF

   create-vreply = "BEGIN" ":" "VREPLY" CRLF
                   created-id
                   request-status
                   "END" ":" "VREPLY" CRLF

                   ; Where the id is appropriate for the
                   ; type of object created:
                   ;
                   ; VAGENDA = calid
                   ; VCAR = carid
                   ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
                   ; VQUERY = queryid
                   ; ALARM = sequence
                   ; x-component = x-id
                   ;
    created-id    =  ( calid / carid / uid / uid dtstamp
                   / queryid / tzid / sequence / x-id)
```

In the following example, two new top level VAGENDAs are created.

Note that the CSID of the server is cal.example.com which is where the new VAGENDAs are going to be created.

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: CMD;ID=creation01:CREATE
C: TARGET:cal.example.com
C: BEGIN:VAGENDA                <- data for 1st new calendar
C: CALID:relcalz1
C: NAME;LANGUAGE=en_US:Bill's Soccer Team
C: OWNER:bill
C: CALMASTER:mailto:bill@example.com
C: TZID:US/Pacific
C: END:VAGENDA
C: BEGIN:VAGENDA                <- data for 2nd new calendar
C: CALID:relcalz2
C: NAME;LANGUAGE=EN-us:Mary's personal calendar
C: OWNER:mary
C: CALMASTER:mailto:mary@example.com
C: TZID:US/Pacific
C: END:VAGENDA
C: END:VCALENDAR
```

When there are multiple TARGET values in the original command object then the replies MUST BE in the exact same order as they were provided to the CS. The same is true for the objects created, their responses MUST BE in the exact same order as they were supplied to the CS.

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: CMD;ID=creation01:REPLY
S: TARGET:cal.example.com
S: BEGIN:REPLY                    <- Reply for 1st calendar create
S: CALID:relcalz1
S: REQUEST-STATUS:2.0
S: END:REPLY
S: BEGIN:VREPLY                   <- Reply for 2nd calendar create
S: CALID:relcalz2
S: REQUEST-STATUS:2.0
S: END:VREPLY
S: END:VCALENDAR
```

To create a new component in multiple containers simply name all of the containers in the TARGET in the create command. Here a new VEVENT is created in two TARGETs. In this example, the VEVENT is one new iTIP REQUEST object in two calendars. The results would be iCalendar object that conform to the iTIP replies as defined in iTIP.

The "VREPLY" components MUST always be returned in the same order that the objects were listed in the original "CREATE" command. If there are multiple "TARGET" and components in the same create command then the reply is first listed by the "TARGET" order of the original create command, then component replies within that "TARGET" are ordered the same as in the original create command.

This example shows two VEVENTs being created in each of two TARGETs:

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: CMD;ID=creation02:CREATE
C: METHOD:REQUEST
C: TARGET:relcalz1
C: TARGET:relcalz2
C: BEGIN:VEVENT
C: DTSTART:20030307T180000Z
C: UID:FirstInThisExample-1
C: DTEND:20030307T190000Z
C: SUMMARY:Important Meeting
C: END:VEVENT
C: BEGIN:VEVENT
C: DTSTART:20040307T180000Z
C: UID:SecondInThisExample-2
C: DTEND:20040307T190000Z
C: SUMMARY:Important Meeting
C: END:VEVENT
C: END:VCALENDAR
```

The CS would sends the REPLY in separate MIME objects, one per TARGET.

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: CMD;ID=creation02:REPLY
S: TARGET:relcalz1    <- 1st TARGET listed
S: BEGIN:REPLY        <- Reply for 1st VEVENT create in 1st TARGET.
S: UID:FirstInThisExample-1
S: REQUEST-STATUS:2.0
S: END:VREPLY
S: BEGIN:REPLY        <- Reply for 2nd VEVENT crate in 1st TARGET.
S: UID:SecondInThisExample-2
S: REQUEST-STATUS:2.0
S: END:VREPLY
S: END:VCALENDAR
```

And would send the second part of the reply later:

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: CMD;ID=creation02:REPLY
S: TARGET:relcalz2    <- 2nd TARGET listed
S: BEGIN:REPLY        <- Reply for 1st VEVENT create in 2nd TARGET.
S: UID:FirstInThisExample-1
S: REQUEST-STATUS:2.0
S: END:VREPLY
S: BEGIN:REPLY        <- Reply for 2nd VEVENT crate in 2nd TARGET.
S: UID:SecondInThisExample-2
S: REQUEST-STATUS:2.0
S: END:VREPLY
S: END:VCALENDAR
```

9.1.5 DELETE Command

CMD: DELETE

Purpose: The DELETE command physically removes the QUERY result from the store or marks it for deletion.

A CUA MAY send a DELETE command to a CS. The DELETE command MUST BE implemented by all CSs.

The CS MUST NOT send a DELETE command to any CUA.

Formal Definition: A "DELETE" command is defined by the following notation:

```
delete-cmd   = deleteparam ":" "DELETE"

deleteparam  = *(

               ; the following are optional,
               ; but MUST NOT occur more than once

                 id-param
               / localize-param
               / latency-param
               / option-param "MARK"

               ; The following MUST occur exactly once and only
               ; when the latency-param has been supplied and
               ; MUST NOT be supplied if the latency-param is
               ; not supplied.

               / action-param

               ; the following is optional,
               ; and MAY occur more than once

               / (";" xparam)

               )
```

The DELETE command is used to delete calendars or components. The included "VQUERY" component(s) specifies the container(s) to delete.

If a component is to be marked for delete and not physically removed, then include the "OPTIONS" parameter with its value set to "MARK" to alter its state to "DELETED".

When components are deleted, only the top most component REQUEST-STATUS is returned. No REQUEST-STATUS is returned for components inside of the selected components. There MUST BE one "VREPLY" component returned for each object that is deleted or marked for delete.

Restriction Table for the REPLY of any DELETE command.

```
delete-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                calprops
                1*(delete-vreply)
               "END" ":" "VCALENDAR" CRLF


delete-vreply  = "BEGIN" ":" "VREPLY" CRLF
                 deleted-id
                 request-status
                "END" ":" "VREPLY" CRLF

                ; Where the id is appropriate for the
                ; type of object deleted:
                ;
                ; VAGENDA = calid
                ; VCAR = carid
                ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
                ; VQUERY = queryid
                ; ALARM = sequence
                ; x-component = x-id
                ;
deleted-id    =  ( calid / carid / uid / uid dtstamp
                   / queryid / tzid / sequence / x-id)
```

Example to delete a VEVENT with VEVENT UID 'abcd12345' from the calendar "relcald-22" from the current CS:

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: TARGET:relcalid-22
C: CMD;ID:"random but unique per CUA":DELETE
C: BEGIN:VQUERY
C: QUERY:SELECT * FROM VEVENT WHERE UID = 'abcd12345'
C: END:VQUERY
C: END:VCALENDAR

S: BEGIN:VCALENAR
S: TARGET:relcalid-22
S: CMD;ID:"random but unique per CUA":REPLY
S: BEGIN:VREPLY
S: UID:abcd12345
S: REQUEST-STATUS:3.0
S: END:VREPLY
S: END:VCALENDAR
```

One or more iCalendar objects will be returned that contain a REQUEST-STATUS for the deleted components. There could have been more than one component deleted, Any booked and any number of unprocessed iTIP scheduling components that matched the QUERY value in the above example. Each unique "METHOD" property value that was deleted from the store MUST BE in a separate iCalendar object. This is because only one "METHOD" property is allowed in a single "VCALENDAR" BEGIN/END block.

9.2 GENERATE-UID Command

CMD: GENERATE-UID

Purpose: The GENERATE-UID command returns one or more unique identifiers which MUST BE globally unique.

The GENERATE-UID command MAY BE sent to any CS. The GENERATE-UID command MUST BE implemented by all CSs.

The GENERATE-UID command MUST NOT be sent to a CUA.

Formal Definition: A "GENERATE-UID" command is defined by the following notation:

```
  generate-uid-cmd   = genuidparam ":" "GENERATE-UID"

     genuidparam     = *(
```

```
                        ; The following are optional,
                        ; but MUST NOT occur more than once.

                          id-param
                        / localize-param
                        / latency-param

                        ; The following MUST occur exactly once and only
                        ; when the latency-param has been supplied and
                        ; MUST NOT be supplied if the latency-param is
                        ; not supplied.

                        / action-param

                        ; The following is optional,
                        ; and MAY occur more than once.

                        / (";" xparam)

                        ; The following MUST BE supplied exactly once.
                        ; The value specifies the number of UIDs to
                        ; be returned.

                        / option-param integer

                        )
```

Response:

```
     gen-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                   calprops
                   1*(delete-vreply)
                  "END" ":" "VCALENDAR" CRLF


    gen-vreply  = "BEGIN" ":" "VREPLY" CRLF
                   *(uid)
                   request-status
                  "END" ":" "VREPLY" CRLF
```

Example:

```
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: PRODID:-//someone's prodid
C: CMD;ID=unique-per-cua-124;OPTIONS=5:GENERATE-UID
C: END:VCALENDAR
```

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: CMD;ID=unique-per-cua-124:REPLY
S: BEGIN:VREPLY
S: UID:20011121T120000Z-12340@cal.example.com
S: UID:20011121T120000Z-12341@cal.example.com
S: UID:20011121T120000Z-12342@cal.example.com
S: UID:20011121T120000Z-12343@cal.example.com
S: UID:20011121T120000Z-12344@cal.example.com
S: END:VREPLY
S: END:VCALENDAR
```

9.3 GET-CAPABILITY Command

CMD: GET-CAPABILITY

Purpose: The GET-CAPABILITY command returns the capabilities of the other end of the session.

A CUA MAY send a GET-CAPABILITY command to a CS. The GET-CAPABILITY command MUST BE implemented by all CSs.

The CS MAY send a GET-CAPABILITY command to any CUA. The GET-CAPABILITY command MAY be implemented by a CUA.

Formal Definition: A "GET-CAPABILITY" command is defined by the following notation:

```
get-capability-cmd   = capibiltyparam ":" "GET-CAPABILITY"

   genuidparam     = *(

                   ; the following are optional,
                   ; but MUST NOT occur more than once

                     id-param
                   / localize-param
                   / latency-param

                   ; the following MUST occur exactly once and only
                   ; when the latency-param has been supplied and
                   ; MUST NOT be supplied if the latency-param is
                   ; not supplied.

                   / action-param

                   ; the following is optional,
                   ; and MAY occur more than once

                   / (";" xparam)

                   )
```

Response:

The GET-CAPABILITY command returns information about the Calendar other end of the session given the current state of the connection. The values returned may differ depending on current user identify and the security level of the connection.

Client implementations SHOULD NOT require any capability element beyond those defined in this specification, and MAY ignore any nonstandard, experimental capability elements. The GET-CAPABILITY reply may return different results depending on the UPN and if the UPN is authenticated.

The following CS properties are returned in response to a GET-CAPABILITY command:

Name	Occurs	Description
CAP-VERSION	1	Version of CAP. MUST include at least "1.0" for

this version of CAP. Like the "VERSION" property, it may have a range. Uses the exact same syntax as the "VERSION" property value.

PRODID	1	The product id of the CS. If supplied in the "VCALENDAR" component, the values MUST BE identical when originated from the CS.
QUERY-LEVEL	1	Indicates level of SQL support. CAL-QL-1 or NONE. (NONE is for CS's that allow ITIP methods only to be deposited and nothing else). If set to NONE, then the 'car' capability MUST BE set to NONE.
CAR-LEVEL	1	Indicates level of CAR support. CAR-NONE, CAR-MIN or CAR-FULL-1. If CAR-FULL-1 is supplied then CAR-MIN MUST BE assumed. CAR = NONE MUST BE used when query-level of NONE is supplied. A CAR-MIN implementation only supports the DEFAULT-VCARS listed in the VCALSTORE and does not support the creation or modification of VCARS.
DATE-MAX	1	The datetime value in UTC beyond which the server cannot accept. The maximum value allowed is 99991231T235959Z.
DATE-MIN	1	The datetime value in UTC prior to which the server cannot accept. The minimum value allowed is 00000101T000000Z.
MAX-COMPONENT-SIZE	1	A positive integer value that specifies the size of the largest iCalendar object that the server will accept in octets. Objects larger than this will be rejected. The absence of this attribute indicates no limit. This is also the maximum value of any BEEP payload the CS will accept or send.
COMPONENTS	1	A comma separated list of the names of components that this CS supports. This includes any components inside of other components (VALARM for example). MUST include at least VCALSTORE, VCALENDAR, VREPLY, and VAGENDA and at least one of VEVENT, VTODO, VTIMEZONE, or VJOURNAL.

VERSION	1	Versions of iCalendar support. MUST BE at least "2.0". This is the same property as defined in [iCAL].
RECUR-ACCEPTED	1	Whether the CS accepts recurrence rules. TRUE or FALSE.
RECUR-EXPAND	1	Whether or not the CS supports the expansion of recurrence rules. TRUE or FALSE
RECUR-LIMIT	1	The maximum number of occurrences or a recurrence rule that are expanded by the CS. Zero means unlimited.
CS-STORES-EXPANDED	1	If TRUE then the CS expands and stores multiple instances separately when they are booked. If FALSE then the CS expands instances dynamically during query.
ITIP-VERSION	1	Version(s) of ITIP, MUST include at least "2447" to specify RFC-2447 support. Comma separated list.
X-...	0+	May include zero or more experimental properties.

Example:

```
I: Content-Type: text/calendar
I:
I: BEGIN:VCALENDAR
I: VERSION:2.0
I: PRODID:The CUA's PRODID
I: CMD;ID=unique-per-cua-125:GET-CAPABILITY
I: END:VCALENDAR

L: Content-Type: text/calendar
L:
L: BEGIN:VCALENDAR
L: VERSION:2.0
L: CMD;ID=unique-per-cua-125:REPLY
L: BEGIN:VREPLY
L: CAP-VERSION:1.0
L: PRODID:The CS prodid
L: QUERY-LEVEL:CAL-QL-1
L: CAR-LEVEL:CAR-FULL-1
L: DATE-MAX:99991231T235959Z
```

```
L: DATE-MIN:00000101T000000Z
L: MAX-COMPONENT-SIZE:0
L: COMPONENTS:VCALENDAR,VTODO,VJOURNAL,VEVENT,VCAR,
L:  VALARM,VFREEBUSY,VTIMEZONE,STANDARD,DAYLIGHT,VREPLY
L: ITIP-VERSION:2447
L: RECUR-ACCEPTED:TRUE
L: RECUR-EXPAND:TRUE
L: RECUR-LIMIT:0
L: CS-STORES-EXPANDED:FALSE
L: X-INET-PRIVATE-COMMANDS:1.0
L: END:VREPLY
L: END:VCALENDAR
```

9.4 IDENTIFY Command

CMD: IDENTIFY

Purpose: The IDENTIFY command allows the CUA to set a new identity to be used for calendar access.

A CUA MAY send an IDENTIFY command to a CS. The IDENTIFY command MUST BE implemented by all CSs. A CS implementation MAY reject all IDENTIFY commands.

The CS MUST NOT send a IDENTIFY command to any CUA.

Formal Definition: A "IDENTIFY" command is defined by the following notation:

```
identify-cmd    = identifyparam ":" "IDENTIFY"

identifyparam   = *(

                ; the following are optional,
                ; but MUST NOT occur more than once

                  id-param
                / localize-param
                / latency-param

                ; the following MUST occur exactly once and only
                ; when the latency-param has been supplied and
                ; MUST NOT be supplied if the latency-param is
                ; not supplied.
```

```
                        / action-param

                        ; the following is optional,
                        ; and MAY occur more than once

                        / (";" xparam)

                        ; The value is the UPN of the requested
                        ; identity. If not supplied it is a request
                        ; to return to the original authenticated identity.

                        / option-param upn

                        )
```

Response:

"REQUEST-STATUS" with only one of the following request-status codes:

2.0 Successful.

6.4 Identity not permitted. VCAR restriction.

The CS determines through an internal mechanism if the credentials supplied at authentication permit the operation as the selected identity. If they do, the session assumes the new identity, otherwise a security error is returned.

Example:

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: PRODID:-//someone's prodid
C: CMD;ID=unique-per-cua-999;OPTIONS=newUserId:IDENTIFY
C: END:VCALENDAR

S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: REQUEST-STATUS:2.0;Request Approved
S: END:VCALENDAR
```

Or if denied:

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: REQUEST-STATUS:6.4;Request Denied
S: END:VCALENDAR
```

And for the CUA to return to its original authenticated identity the OPTIONS parameter is omitted:

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: PRODID:-//someone's prodid
C: CMD;ID=unique-per-cua-995:IDENTIFY
C: END:VCALENDAR
```

The CS may accept (2.0) or deny (6.4) the request to return to the original identity.

If a CS considers the IDENTIFY command an attempt to violate security, the CS MAY terminate the BEEP session without any further notice to the CUA after sending the REQUEST-STATUS 6.4 reply.

9.5 MODIFY Command

CMD: MODIFY

Purpose: The "MODIFY" command is used to modify existing components.

A CUA MAY send a MODIFY command to a CS. The MODIFY command MUST BE implemented by all CSs.

The CS MUST NOT send a MODIFY command to any CUA.

Formal Definition: A "MODIFY" command is defined by the following notation:

```
        modify-cmd   = modifyparam ":" "MODIFY"

       modifyparam   = *(

                     ; the following are optional,
                     ; but MUST NOT occur more than once

                       id-param
                     / localize-param
                     / latency-param

                     ; the following MUST occur exactly once and only
                     ; when the latency-param has been supplied and
                     ; MUST NOT be supplied if the latency-param is
                     ; not supplied.

                     / action-param

                     ; the following is optional,
                     ; and MAY occur more than once

                     / (";" xparam)

                     )
```

Response:

The "MODIFY" command is used to modify existing components. The TARGET property specifies the calendars were the components exist that are going to be modified.

The format of the request is two containers inside of VCALENDAR container object:

```
BEGIN:VCALENDAR
...
BEGIN:VQUERY
...
END:VQUERY
BEGIN:XXX
...old-values...
END:XXX
BEGIN:XXX
...new-values...
END:XXX
END:CALENDAR
```

The VQUERY selects the components that are to be modified.

Where "XXX" above is a named component type (VEVENT, VTODO, ...). Both the old and new components MUST BE of the same type.

The old-values is a component and the contents of that component are going to change and may contain information that helps uniquely identify the original component (SEQUENCE in the example below). If the CS can not find a component that matches the QUERY and does not have at least all of the OLD-VALUES, then a 6.1 error is returned.

The new-values is a component of the same type as old-values and new-values contains the new data for each selected component. Any data that is in old-values and not in new-values is deleted from the selected component. Any values in new-values that was not in old-values is added to the component.

In this example the VEVENT with UID:unique-58 has; the LOCATION and LAST-MODIFIED changed, the VALARM with SEQUENCE:3 has its TRIGGER disabled, the X-LOCAL property is removed from the VEVENT, and a COMMENT is added.

Because SEQUENCE is used to locate the VALARM in this example, both the old-values and the new-values contains SEQUENCE:3 and if SEQUENCE was left out of new-values - it would have been deleted.

Example:

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: TARGET:my-cal
C: CMD:ID=unique-mod:MODIFY
C: BEGIN:VQUERY                  <- Query to select data set.
C: QUERY:SELECT * FROM VEVENT WHERE UID = 'unique-58'
C: END:VQUERY
C: BEGIN:VEVENT                  <- Start of old data.
C: LOCATION:building 3
C: LAST-MODIFIED:20020101T123456Z
C: X-LOCAL:some private stuff
C: BEGIN:VALARM
C: SEQUENCE:3
C: TRIGGER;RELATED=END:PT5M
C: END:VALARM
C: END:VEVENT
C: BEGIN:VEVENT                  <- End of new data.
C: LOCATION:building 4
C: LAST-MODIFIED:20020202T010203Z
C: COMMENT:Ignore global trigger.
C: BEGIN:VALARM
C: SEQUENCE:3
C: TRIGGER;ENABLE=FALSE:RELATED=END:PT5M
C: END:VALARM
C: END:VEVENT
```

X-LOCAL was not supplied in the new-values, so it was deleted. LOCATION was altered, as was LAST-MODIFIED. The VALARM with SEQUENCE:3 had its TRIGGER disabled, and SEQUENCE did not change so it was not effected. COMMENT was added.

When it comes to inline ATTACHMENTs, the CUA only needs to uniquely identify the contents of the ATTACHMENT value in the old-values in order to delete them. When the CS compares the attachment data it is compared in its binary form. The ATTACHMENT value supplied by the CUA MUST BE valid encoded information.

For example, to delete the same huge inline attachment from every VEVENT in 'my-cal' that has an ATTACH with the old-values:

```
BEGIN:VCALENDAR
VERSION:2.0
TARGET:my-cal
CMD:MODIFY
BEGIN:VQUERY
QUERY:SELECT ATTACH FROM VEVENT
END:VQUERY
BEGIN:VEVENT
ATTACH;FMTTYPE=image/basic;ENCODING=BASE64;VALUE=BINARY:
 MIICajCCAdOgAwIBAgICBEUwDQYJKoZIhvcNAQEEBQAwdzELMAkGA1U
 EBhMCVVMxLDAqBgNVBAoTI05ldHNjYXBlIENvbW11bmljYXRpb25zIE
 ...< remainder of attachment data NOT supplied >....
END:VEVENT
BEGIN:VEVENT
END:VEVENT
END:VCALENDAR
```

Above the new-values is empty, so everything in the old-values is deleted.

Furthermore, the following additional restrictions apply:

1. One can not change the "UID" property of a component.

2. If a contained component is changed inside of a selected component, and that contained component has multiple instances, then old-values MUST contain information that uniquely identifies the instance or instances that are changing. It is valid to change more than one. As all contained components that match old-values will be modified. In the first modify example above, if SEQUENCE were to be deleted from both the old-values and new-values, then all TRIGGERs that matched the old-values in all VALARM in the selected VEVENTs would be disabled.

3. The result of the modify MUST BE a valid iCalendar object.

If the REQUEST-STATUS is 2.0, then the entire modification was successful.

If any error occurred:

No component will be changed at all. That is, it will appear just as it was prior to the modify and the CAP server SHOULD return a REQUEST-STATUS for each error that occurred.

There MUST BE at least one error reported.

If multiple components are selected, then what uniquely identified the component MUST BE returned (UID, QUERYID, ...) if the component contains a unique identifier. If not sufficient information to uniquely identify the modified components MUST BE returned in the reply.

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: TARGET:relcalid
S: CMD;ID=delete#1:REPLY
S: BEGIN:VREPLY
S: BEGIN:VEVENT
S: UID:123
S: REQUEST-STATUS:2.0
S: END:VEVENT
S: END:VREPLY
S: END:VCALENDAR
```

9.6 MOVE Command

CMD: MOVE

Purpose: The MOVE command is used to move components within the CS.

A CUA MAY send a MOVE command to a CS. The MOVE command MUST BE implemented by all CSs.

The CS MUST NOT send a MOVE command to any CUA.

Formal Definition: A "MOVE" command is defined by the following notation:

```
move-cmd   = moveparam ":" "MOVE"

moveparam  = *(

                ; the following are optional,
                ; but MUST NOT occur more than once

                  id-param
                / localize-param
                / latency-param

                ; the following MUST occur exactly once and only
                ; when the latency-param has been supplied and
                ; MUST NOT be supplied if the latency-param is
                ; not supplied.

                / action-param

                ; the following is optional,
                ; and MAY occur more than once

                / (";" xparam)

                )
```

Response:
The REQUEST-STATUS in a VCALENDAR object.

The content of each "result" is subject to the result restriction table defined below.

The access control on the VAGENDA after it has been moved to its new location in the calstore MUST BE at least as secure as it was prior to the move. If the CS is not able to ensure the same level of security, a permission denied REQUEST-STATUS MUST BE returned and the MOVE operation not performed.

The TARGET property specifies the new location, and the VQUERY property specifies the old location.

Restriction Table for the "REPLY":

```
move-reply  = "BEGIN" ":" "VCALENDAR" CRLF
              calprops
              1*(move-vreply)
              "END" ":" "VCALENDAR" CRLF


move-vreply  = "BEGIN" ":" "VREPLY" CRLF
               move-id
                request-status
               "END" ":" "VREPLY" CRLF

               ; Where the id is appropriate for the
               ; type of object moved:
               ;
               ; VAGENDA = calid
               ; VCAR = carid
               ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
               ; VQUERY = queryid
               ; ALARM = sequence
               ; x-component = x-id
               ;
move-id    =  ( calid / carid / uid / uid dtstamp
              / queryid / tzid / sequence / x-id)
```

Example: moving the VAGENDA Nellis to Area-51

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: CMD:MOVE
C: TARGET:Area-51
C: BEGIN:VQUERY
C: QUERY: SELECT * FROM VAGENDA WHERE CALID='Nellis'
C: END:VQUERY
C: END:VCALENDAR

S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: TARGET:Area-51
S: BEGIN:VREPLY
S: CALID:Nellis
S: REQUEST-STATUS: 2.0
S: END:VREPLY
S: END:VCALENDAR
```

9.7 REPLY Response to a Command

CMD: REPLY

Purpose: The "REPLY" value to the "CMD" property is used to return the results of all other commands to the CUA.

A CUA MUST send a REPLY command to a CS for any command a CS MAY send to the CUA. The REPLY command MUST BE implemented by all CUAs that support getting the GET-CAPABILITY command.

A CS MUST send a REPLY command to a CUA for any command a CUA MAY send to the CS. The REPLY command MUST BE implemented by all CSs.

Formal Definition: A "REPLY" command is defined by the following notation:

```
reply-cmd    = replyparam ":" "REPLY"

replyparam   = *(

           ; The 'id' parameter value MUST BE exactly the
              ; same as the value sent in the original
              ; CMD property. If the original CMD did
              ; not have an 'id' parameter, then the 'id'
              ; MUST NOT be supplied in the REPLY.

              id-param

              ; the following is optional,
              ; and MAY occur more than once

              / (";" xparam)

              )
```

9.8 SEARCH Command

CMD: SEARCH

Purpose: The "SEARCH" command is used to return selected components to the CUA.

A CUA MAY send a SEARCH command to a CS. The SEARCH command MUST BE implemented by all CSs.

The CS MUST NOT send a SEARCH command to any CUA.

Formal Definition: A "SEARCH" command is defined by the following notation:

```
search-cmd   = searchparam ":" "SEARCH"

searchparam  = *(

               ; the following are optional,
               ; but MUST NOT occur more than once

                 id-param
               / localize-param
               / latency-param

               ; the following MUST occur exactly once and only
               ; when the latency-param has been supplied and
               ; MUST NOT be supplied if the latency-param is
               ; not supplied.

               / action-param

               ; the following is optional,
               ; and MAY occur more than once

               / (";" xparam)

               )
```

Response:

The data in each result contains an iCalendar object composed of all the selected components enclosed in a "VREPLY" component. Only "REQUEST-STATUS" and the properties mentioned in the "SELECT" clause of the QUERY are included in the components. Each iCalendar object is tagged with the TARGET property and optional CMD property.

Searching for objects

In the example below objects on March 10,1999 between 080000Z and 190000Z are read. In this case only 4 properties for each objects are returned. Two calendars are specified. Only booked (vs scheduled) entries are to be returned (this example only selecte VEVENT objects):

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: CMD:SEARCH
C: TARGET:relcal2
C: TARGET:relcal3
C: BEGIN:VQUERY
C: QUERY:SELECT DTSTART,DTEND,SUMMARY,UID
C:  FROM VEVENT
C:  WHERE DTEND >= '19990310T080000Z'
C:  AND DTSTART <= '19990310T190000Z'
C:  AND STATE() = 'BOOKED'
C: END:VQUERY
C: END:VCALENDAR
```

The return values are subject to VCAR filtering. That is, if the request contains properties to which the UPN does not have access, those properties will not appear in the return values. If the UPN has access to at least one property of the component, but has been denied access to all properties called out in the request, the response will contain a single REQUEST-STATUS property indicating the error.

Here the request was successful, but the VEVENT contents were not accessible (4.1).

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: TARGET:relcalid
S: CMD:REPLY
S: VERSION:2.0
S: BEGIN:VREPLY
S: BEGIN:VEVENT
S: REQUEST-STATUS:4.1
S: END:VEVENT
S: END:VREPLY
S: END:VCALENDAR
```

If the UPN has no access to any components at all, the response will simply be an empty data set. The response looks the same if there the particular components did not exist.

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: CMD:REPLY
S: TARGET:ralcalid
S: BEGIN:VREPLY
S: REQUEST-STATUS:2.0
S: END:VREPLY
S: END:VCALENDAR
```

If there are multiple targets, each iCalendar reply is contained within its own iCalendar object.

Stored VQUERY can be used by specifying the property QUERYID instead of QUERY.

9.9 SET-LOCALE Command

CMD: SET-LOCALE

Purpose: The "SET-LOCALE" command is used to select the locale that will be used in error codes used int the "REQUEST-STATUS" property. It also effect the locale sorting order for queries.

A CUA MAY send a "SET-LOCALE" command to a CS. The SET-LOCALE command MUST BE implemented by all CSs.

The CS MUST NOT send a "SET-LOCALE" command to any CUA.

Formal Definition: A "SET-LOCALE" command is defined by the following notation:

```
      setlocale-cmd   = searchparam ":" "SET-LOCALE"

     setlocaleparam   = *(

                      ; the following are optional,
                      ; but MUST NOT occur more than once

                        id-param
                      / localize-param
                      / latency-param
                      / setlocale-option

                      ; the following MUST occur exactly once and only
                      ; when the latency-param has been supplied and
                      ; MUST NOT be supplied if the latency-param is
                      ; not supplied.

                      / action-param

                      ; the following is optional,
                      ; and MAY occur more than once

                      / (";" xparam)

    setlocal-option   = option-param newlocale

          newlocale   = ; Any locale supplied in the initial BEEP
                        ; "greeting" "localize" parameter and
                        ; and any charset supported by the CS
                        ; and listed in the DEFAULT-CHARSET property
                        ; of the VCALSTORE.

                      )
```

Restriction Table for the REPLY of any SET-LOCALE command.

```
      setlocale-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                         calprops
                         1*(setlocale-vreply)
                        "END" ":" "VCALENDAR" CRLF

     setlocale-vreply  = "BEGIN" ":" "VREPLY" CRLF
                         request-status
                         "END" ":" "VREPLY" CRLF
```

9.10 TIMEOUT Command

CMD: TIMEOUT

Purpose: The "TIMEOUT" command is only sent after a command has been sent with a latency value set. When received it means the command could not be completed in the time allowed.

Formal Definition: A "CONTINUE" command is defined by the following notation:

```
      continue-cmd   = continueparam ":" "CONTINUE"

     continueparam   = *(

                     ; the following are optional,
                     ; but MUST NOT occur more than once

                       id-param
                     / localize-param

                     / (";" xparam)

                     )
```

The REPLY of any "TIMEOUT" command is:

```
       timeout-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                        calprops
                        timeout-vreply
                       "END" ":" "VCALENDAR" CRLF

      timeout-vreply  = "BEGIN" ":" "VREPLY" CRLF
                         request-status
                        *(x-prop)
                        "END" ":" "VREPLY" CRLF
```

9.11 Response Codes

Numeric response codes are returned using the REQUEST-STATUS property.

The format of these codes is described in [iCAL], and extend in [iTIP] and [iMIP]. The following describes new codes added to this set and how existing codes apply to CAP.

At the application layer response codes are returned as the value of a "REQUEST-STATUS" property. The value type of this property is modified from that defined in [iCAL], in order to make the accompanying REQUEST-STATUS text optional.

Code	Description
2.0	Success. The parameters vary with the operation and are specified.
2.0.3	In response to the client issuing an "abort" reply, this reply code indicates that any command currently underway was successfully aborted.
3.1.4	Capability not supported.
4.1	Calendar store access denied.
6.1	Container not found.
6.2	Attempt to create or modify an object such that it would overlap another object in either of the following two circumstances: (a) One of the objects has a TRANSP property set to OPAQUE-NOCONFLICT or TRANSPARENT-NOCONFLICT. (b) The calendar's ALLOW-CONFLICT property is set to FALSE.
6.3	Bad args.
6.4	Permission denied - VCAR restriction. A VCAR exists and the CS will not perform the operation.
7.0	A timeout has occurred. The server was unable to complete the operation in the requested time.
8.0	A failure has occurred in the Calendar Server that prevents the operation from succeeding.
8.1	A query was performed and the query is

	too complex for the CS. The operation was not performed.
8.2	Used to signal that an iCalendar object has exceeded the server's size limit
8.3	A DATETIME value was too far in the future represented on this Calendar.
8.4	A DATETIME value was too far in the past to be represented on this Calendar.
8.5	An attempt was made to create a new object but the unique UID specified is already in use.
9.0	An unrecognized command was received. Or an unsupported command was received.
10.4	The operation has not been performed because it would cause the resources (memory, disk, CPU, etc) to exceed the allocated quota.

---

10. Object Registration

This section provides the process for registration of new or modified properties, parameters, commands, or other modifications, additions, or deletions to objects.

10.1 Registration of New and Modified Entities

New objects are registered by the publication of an IETF Request for Comment (RFC). Changes to a objects are registered by the publication of a revision to the RFC in a new RFC.

10.2 Post the item definition

The object description MUST BE posted to the new object discussion list: ietf-calendar@imc.org.

10.3 Allow a comment period

Discussion on a new object MUST BE allowed to take place on the list for a minimum of two weeks. Consensus MUST BE reached on the object before proceeding to the next step.

10.4 Release a new RFC

The new object will be submitted for publication as any other internet draft requesting RFC status.

11. BEEP and CAP

11.1 BEEP Profile Registration

TBD

11.2 BEEP Exchange Styles

[BEEP] defines three styles of message exchange:

MSG/ANS,ANS,...,NUL - For one to many exchanges.

MSG/RPY - For one to one exchanges.

MSG/ERR - For requests the cannot be processed due to an error.

A CAP request, targeted at more than one containers, MAY use a one-to-many exchange, with a distinct answer associated with each target. CAP request targeted at a single container MAY use a one-to-one exchange or a one-to-many exchange. "MSG/ERR" MAY only be used when an error condition prevents the execution of the request on all the targeted calendars.

12. IANA Considerations

This memo defines IANA registered extensions to the attributes defined by iCalendar, as defined in [iCAL], and [iTIP].

IANA registration proposals for iCalendar and iTIP are to be mailed to the registration agent for the "text/calendar" [MIME] content-type, <MAILTO: ietf-calendar@imc.org> using the format defined in section 7 of [iCAL].

If the IESG approves this memo for publication, then the IANA registers the profile specified in Section 11.1, and selects an IANA-specific URI, e.g., http://iana.org/beep/cap/1.0.

13. Security Considerations

Access rights should be granted cautiously, consult Section 2.4.2 for a discussion of the subject. Without careful planning it is possible to open up access to a greater degree than desired.

The "IDENTIFY" command should be carefully implemented as discussed in Section 6.1.3.

In addition, since CAP is a profile of the BEEP, consult [BEEP]'s Section 9 for a discussion of BEEP-specific security issues.

Although service provisioning is a policy matter, at a minimum, all implementations must provide the following tuning profiles:

for authentication: http://iana.org/beep/SASL/DIGEST-MD5

for confidentiality: http://iana.org/beep/TLS (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher)

for both: http://iana.org/beep/TLS (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher supporting client-side certificates)

URIs

[1] <http://www.imc.org/html.charters/calsch-charter.html>

Authors' Addresses

Doug Royer
INET-Consulting.com
1795 W. Broadway #266
Idaho Falls, ID 83402
US

Phone: +1-866-594-8574
Fax: +1-866-594-8574
EMail: Doug@Royer.com
URI: http://INET-Consulting.com

George Babics
Steltor
2000 Peel Street
Montreal, Quebec H3A 2W5
CA

Phone: +1-514-733-8500 x4201
Fax: +1-514-733-8878
EMail: georgeb@steltor.com

Paul Hill
Massachusetts Institute of Technology
W92-172
77 Massachusetts Avenue
Cambridge, MA 02139
US

Phone: +1-617-253-0124
Fax: +1-617-258-8736
EMail: phb@mit.edu

Steve Mansour
AOL/Netscape
466 Ellis Road
Mountain View, CA 94043
US

Phone: +1-650-937-3351
EMail: sman@netscape.com

Appendix A. Acknowledgments

The following have individuals were major contributors in the drafting and discussion of this memo:

Harald Alvestrand, Mario Bonin, Andre Courtemanche, Dave Crocker, Bernard Desruisseaux, Pat Egen, Gilles Fortin, Jeff Hodges, Alex Hoppman, Bruce Kahn, Patrice Lapierre, Lisa Lippert, David Madeo, Bob Mahoney, Bob Morgan, Pete O'Leary, Richard Shusterman, Tony Small, John Stracke, Alexander Taler, Mark Wahl.

Appendix B. Bibliography

[RFC1521] Borenstein, N., Freed, N., "Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993 ftp://ftp.isi.edu/in-notes/rfc1521.txt

[RFC1738] Berners-Lee, T, Masinter, L. and McCahil, M., "Uniform Resource Locators (URL)", RFC 1738, December 1994 ftp://ftp.isi.edu/in-notes/rfc1738.txt

[RFC2045] Borenstein, N. and Freed, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996 ftp://ftp.isi.edu/in-notes/rfc2045.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997 ftp://ftp.isi.edu/in-notes/rfc2119.txt

[RFC2222] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997 ftp://ftp.isi.edu/in-notes/rfc2222.txt

[RFC2246] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999 ftp://ftp.isi.edu/in-notes/rfc2246.txt

[RFC2392] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998 ftp://ftp.isi.edu/in-notes/rfc2392.txt

[RFC2396] Berners-Lee, T, Fielding, R. and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998 ftp://ftp.isi.edu/in-notes/rfc2396.txt

[iCAL] Dawson, F. and Stenerson, D., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998 ftp://ftp.isi.edu/in-notes/rfc2245.txt

[iTIP] Silverberg, S., Mansour, S., Dawson, F. and Hopson, R., "iCalendar Transport-Independent Interoperability Protocol (iTIP) Events, BusyTime, To-dos and Journal Entries", RFC 2446, November 1998 ftp://ftp.isi.edu/in-notes/rfc2446.txt

[iMIP] Dawson, F., Mansour, S. and Silverberg, "iCalendar Message-Based Interoperability Protocol (iMIP)", RFC 2447, November 1998 ftp://ftp.isi.edu/in-notes/rfc2447.txt

[RFC3080] Rose, M., "The Block Extensible Exchange Protocol Core", RFC 3080, March 2001
ftp://ftp.isi.edu/in-notes/rfc3080.txt

[RFC3081] Rose, M., "Mapping the BEEP Core onto TCP", RFC 3081, March 2001
ftp://ftp.isi.edu/in-notes/rfc3081.txt

[SQL] "Database Language SQL", ANSI/ISO/IEC 9075: 1992,
aka ANSI X3.135-1992, aka FiPS PUB 127-2

[SQLCOM] ANSI/ISO/IEC 9075:1992/TC-1-1995, Technical corrigendum 1
to ISO/IEC 9075: 1992, also adopted as Amendment 1 to
ANSI X3.135.1992

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 3.1"
http://www.unicode.org/unicode/standard/standard.html

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

[CharEncoding] "Worldwide Character Encoding -- Version 1.0",
Addison-Wesley, Volume 1, 1991, Volume 2, 1992.
UTF-8 is described in Unicode Technical Report #4.

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.