Network Working Group
Request for Comments: 4324
Category: Experimental

D. Royer
IntelliCal, LLC
G. Babics
Oracle
S. Mansour
eBay

December 2005

Calendar Access Protocol (CAP)

Status of This Memo

Copyright Notice

Abstract

The Calendar Access Protocol (CAP) described in this memo permits a Calendar User (CU) to utilize a Calendar User Agent (CUA) to access an iCAL-based Calendar Store (CS). At the time of this writing, three vendors are implementing CAP, but it has already been determined that some changes are needed. In order to get implementation experience, the participants felt that a CAP specification is needed to preserve many years of work. Many properties in CAP which have had many years of debate, can be used by other iCalendar protocols.

Table of Contents

```
   1. Introduction ....................................................5
      1.1. Formatting Conventions .....................................5
      1.2. Related Documents ..........................................6
      1.3. Definitions ................................................7
   2. Additions to iCalendar .........................................11
      2.1.  New Value Types (Summary) ................................14
         2.1.1. New Parameters (summary) .............................14
         2.1.2. New or Updated Properties (summary) ..................14
         2.1.3. New Components (summary) .............................17
      2.2.  Relationship of RFC-2446 (ITIP) to CAP ...................18
   3. CAP Design .....................................................20
      3.1. System Model ..............................................20
      3.2. Calendar Store Object Model ...............................20
      3.3. Protocol Model ............................................21
         3.3.1. Use of BEEP, MIME, and iCalendar .....................22
   4. Security Model .................................................23
      4.1. Calendar User and UPNs ....................................23
         4.1.1. UPNs and Certificates ................................24
         4.1.2. Anonymous Users and Authentication ...................25
         4.1.3. User Groups ..........................................25
      4.2. Access Rights .............................................26
         4.2.1. Access Control and NOCONFLICT ........................26
         4.2.2. Predefined VCARs .....................................26
         4.2.3. Decreed VCARs ........................................28
      4.3. CAP Session Identity ......................................28
   5. CAP URL and Calendar Address ...................................29
   6. New Value Types ................................................30
      6.1. Property Value Data Types .................................30
         6.1.1. CAL-QUERY Value Type .................................30
            6.1.1.1. [NOT] CAL-OWNERS() ..............................36
            6.1.1.2. CURRENT-TARGET() ................................37
            6.1.1.3. PARAM() .........................................37
            6.1.1.4. SELF() ..........................................38
            6.1.1.5. STATE() .........................................38
            6.1.1.6. Use of Single Quote .............................38
            6.1.1.7. Comparing DATE and DATE-TIME Values .............39
            6.1.1.8. DTEND and DURATION ..............................40
            6.1.1.9. [NOT] LIKE ......................................40
            6.1.1.10. Empty vs. NULL .................................41
            6.1.1.11. [NOT] IN .......................................41
            6.1.1.12. DATE-TIME and TIME Values in a WHERE Clause ....42
            6.1.1.13. Multiple Contained Components ..................43
            6.1.1.14. Example, Query by UID ..........................43
            6.1.1.15. Query by Date-Time Range .......................43
            6.1.1.16. Query for All Unprocessed Entries ..............44
            6.1.1.17. Query with Subset of Properties by Date/Time ...44
            6.1.1.18. Query with Components and Alarms in A Range ....45
         6.1.2. UPN Value Type .......................................45
         6.1.3. UPN-FILTER Value .....................................46
   7. New Parameters .................................................48
      7.1. ACTION Parameter ..........................................48
      7.2. ENABLE Parameter ..........................................48
      7.3. ID Parameter ..............................................49
      7.4. LATENCY Parameter .........................................50
      7.5. LOCAL Parameter ...........................................50
      7.6. LOCALIZE Parameter ........................................51
      7.7. OPTIONS Parameter .........................................52
   8. New Properties .................................................52
      8.1. ALLOW-CONFLICT Property ...................................52
      8.2. ATT-COUNTER Property ......................................53
      8.3. CALID Property ............................................54
      8.4. CALMASTER Property ........................................54
      8.5. CAP-VERSION Property ......................................55
      8.6. CARID Property ............................................55
      8.7. CAR-LEVEL Property ........................................56
      8.8. COMPONENTS Property .......................................56
      8.9. CSID Property .............................................58
      8.10. DECREED Property .........................................58
      8.11. DEFAULT-CHARSET Property .................................59
      8.12. DEFAULT-LOCALE Property ..................................60
      8.13. DEFAULT-TZID Property ....................................61
      8.14. DEFAULT-VCARS Property ...................................62
      8.15. DENY Property ............................................62
      8.16. EXPAND property ..........................................63
      8.17. GRANT Property ...........................................64
      8.18. ITIP-VERSION Property ....................................64
      8.19. MAX-COMP-SIZE Property ...................................65
      8.20. MAXDATE Property .........................................65
      8.21. MINDATE Property .........................................66
      8.22. MULTIPART Property .......................................66
      8.23. NAME Property ............................................67
      8.24. OWNER Property ...........................................68
      8.25. PERMISSION Property ......................................68
      8.26. QUERY property ...........................................69
      8.27. QUERYID property .........................................70
      8.28. QUERY-LEVEL Property .....................................70
      8.29. RECUR-ACCEPTED Property ..................................71
      8.30. RECUR-LIMIT Property .....................................71
      8.31. RECUR-EXPAND Property ....................................72
      8.32. RESTRICTION Property .....................................72
      8.33. SCOPE Property ...........................................73
      8.34. STORES-EXPANDED Property .................................74
      8.35. TARGET Property ..........................................74
      8.36. TRANSP Property ..........................................75
```

9. New Components 76
   9.1. VAGENDA Component 76
   9.2. VCALSTORE Component 78
   9.3. VCAR Component 80
   9.4. VRIGHT Component 82
   9.5. VREPLY Component 83
   9.6. VQUERY Component 83
10. Commands and Responses 85
   10.1. CAP Commands (CMD) 85
   10.2. ABORT Command 88
   10.3. CONTINUE Command 89
   10.4. CREATE Command 90
   10.5. DELETE Command 96
   10.6. GENERATE-UID Command 98
   10.7. GET-CAPABILITY Command 100
   10.8. IDENTIFY Command 103
   10.9. MODIFY Command 105
   10.10. MOVE Command 110
   10.11. REPLY Response to a Command 112
   10.12. SEARCH Command 113
   10.13. SET-LOCALE Command 116
   10.14. TIMEOUT Command 118
   10.15. Response Codes 118
11. Object Registration 120
   11.1. Registration of New and Modified Entities 120
   11.2. Post the Item Definition 120
   11.3. Allow a Comment Period 120
   11.4. Release a New RFC 120
12. BEEP and CAP 120
   12.1. BEEP Profile Registration 120
   12.2. BEEP Exchange Styles 123
   12.3. BEEP Connection Details 123
13. IANA Considerations 125
14. Security Considerations 125
Appendix A. Acknowledgements 127
Appendix B. References 127
    Appendix B.1. Normative References 127
    Appendix B.2. Informative References 128

1. Introduction

This document specifies the Calendar Access Protocol (CAP). CAP permits a Calendar User (CU) to utilize a Calendar User Agent (CUA) to access an iCAL-based Calendar Store (CS) and manage calendar information. In particular, the document 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, but it is believed to be possible using CAP.

At the time of this writing, three vendors are implementing CAP. It has already been determined that some changes are needed. In order to get implementation experience, the participants felt that a CAP specification is needed to preserve many years of work. Many properties in CAP can be used by other iCalendar protocols and have had many years of debate.

CAP is specified as a BEEP (Block Extensible Exchange Protocol) "profile" [BEEP] [BEEPGUIDE]. Many aspects of the protocol (e.g., authentication and privacy) are provided within BEEP. The protocol data units of CAP leverage the standard iCalendar format iCAL [iCAL] to convey calendar-related information.

CAP can also be used to store and fetch iCalendar Transport-Independent Interoperability Protocol (iTIP) objects [iTIP]. iTIP objects used are exactly as defined in [iTIP]. When iCalendar objects are transferred between the CUA and a CS, some additional properties and parameters may be added; the CUA is responsible for correctly generating iCalendar objects to non-CAP processes.

The definition of new components, properties, parameters, and value types are broken into two parts. The first part summarizes and defines the new objects. The second part provides detail and ABNF for those objects. The ABNF rules for CAP, as for other iCalendar specifications, are order-independent. That is, properties in a component may occur in any order, and parameters in any property may occur in any order.

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".

Object states defined by this memo are referred to with capitalized, quoted-strings of text, followed by the word "state". For example, '"BOOKED" state' refers to an object in the booked state.

Within a query, the different parts are referred to as a "clause" and its value as "clause value" and the clause name will be in uppercase enclosed in quotes, for example, 'The "SELECT" claus' or 'if the "SELECT" clause value contains ...'.

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 as follows.

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

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

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

[GUIDE] (RFC3283) is 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 earlier memos. Where possible, references are made to the memo that provides the specification of these concepts and definitions.

1.3 Definitions

UNPROCESSED, BOOKED, DELETED - A conceptual state of an object in the calendar store. There are three conceptual states: "UNPROCESSED" state, "BOOKED" state, and marked for deletion, which is the "DELETED" state. How the implementation stores the state of any object is not a protocol issue and is not discussed. An object can be said to be booked, unprocessed, or marked for deletion.

1. An "UNPROCESSED" state scheduling object has been stored in the calendar store but has not been acted on by a CU or CUA. All scheduled entries are [iTIP] objects. No [iTIP] objects in the store are in the "BOOKED" state. To retrieve any [iTIP] object, simply do a query asking for any objects that are stored in the "UNPROCESSED" state.

2. A "BOOKED" state entry is stored with the "CREATE" command. It is an object 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 in the "BOOKED" state.

3. A "DELETED" state entry is created by sending a "DELETE" command with the "OPTION" parameter value set to "MARK". To retrieve any deleted object, simply do a query asking for any objects that were stored in the "DELETED" state. By default objects marked for delete are not returned. The CUA must specifically ask for marked-for-deletion objects. You cannot

ask for components in the "DELETED" state and in other states in the same "VQUERY" component, as there would be no way to distinguish between them in the reply.

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 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", defined below) is granted or denied permission to perform on a given calendar object ("SCOPE"). The calendar access rights are specified with a "VCAR" component. (Section 9.3)

Calendar Address - Also see Calendar URL, which is the same as a CAP address. 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.

Container - This is a generic name for VCALSTORE or VAGENDA.

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

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

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

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 [URL]. The CSID excludes any reference to a specific calendar. (Section 8.9)

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 represent 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 CS. If the CAP session is authenticated, 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. For example, a "VALARM" component may be contained inside a "VEVENT" component, and a "TRIGGER" property could be a contained property of a "VALARM" component.

Delegate - A CU (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 invitee who is unable to attend.

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

Experimental - The CUA and CS may implement experimental extensions to the protocol. They might also 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. There is no guarantee that the same specific experimental extension is not used by multiple vendors in incompatible ways. 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 that 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 object, 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 or entry 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.

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 [URI] and [URLGUIDE].

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 that 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 CUs. (Section 6.1.2)

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 also adds to the [iCAL] ABNF to allow IANA and experimental extensions. This memo does not address how a CUA transmits [iTIP] or [iMIP] objects to non-CAP programs. What follows is ABNF, as described in [ABNF].

```
   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          /
                cmd             /
                    ;
                    ; Target is optional, and may occur more
                    ; than once.
                    ;
                target / other-props )
                    ;
      other-props  = *(x-prop) *(iana-prop) *(other-props)
                    ;
      iana-prop   = ; Any property registered by IANA directly or
                    ; included in an RFC that may be applied to
                    ; the component and within the rules published.
                    ;
      x-prop      = ; As defined in [iCAL].
                    ;
      methodp     = ; As defined in [iCAL].
                    ;
      prodid      = ; As defined in [iCAL].
                    ;
      calscale    = ; As defined in [iCAL].
                    ;
```

Another change is that the 'component' part of the 'icalbody' ABNF as described in [iCAL] section 4.6 is optional when sending a command, as shown in the following updated ABNF:

```
      icalbody = calprops component

                ; If the "VCALENDAR" component contains the "CMD"
                ; property then the 'component' is optional:
                ;
                / calprops     ; Which MUST include a "CMD" property
                ;
      component = ; As defined in [iCAL].
```

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

```
        alarmc     = "BEGIN" ":" "VALARM" CRLF
```

```
                     alarm-seq
                     other-props
                     (audioprop / dispprop / emailprop / procprop)
                     "END" ":" "VALARM" CRLF
                     ;
       emailprop   = ; As defined in [iCAL]
                     ;
       procprop    = ; As defined in [iCAL]
                     ;
       dispprop    = ; As defined in [iCAL]
                     ;
       audioprop   = ; As defined in [iCAL]
                     ;
       alarm-seq   = "SEQUENCE" alarmseqparams ":" posint0 CRLF
                     ;
       alarmseqparams = other-params [";" local-param] other-params
                     ;
                     ; Where DIGIT is defined in [iCAL]
                     ;
       posint0     = 1*DIGIT
       posint1     = posintfirst 1*DIGIT
                     ;
                     ; A number starting with 1 through 9.
                     ;
       posintfirst = %x31-39
                     ;
       other-params = *(";" xparam) *(";" iana-params)
                                    *(";" other-params)
                     ;
       iana-params = ; Any parameter registered by IANA directly or
                     ; included in an RFC that may be applied to
                     ; the property and within the rules published.
                     ;
       xparam        ; As defined in [iCAL].
                     ;
```

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

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

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

```
                   ;
      trigrel    = ; As defined in [iCAL].
                   ;
      trigabs    = ; As defined in [iCAL].
```

See Section 7.2 and Section 7.5.

2.1. New Value Types (Summary)

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

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

CALQUERY: The "CAL-QUERY" value type is a query syntax that is used by the CUA to specify the rules that apply to a CAP command (see Section 6.1.1).

2.1.1. New Parameters (summary)

ACTION - The "ACTION" parameter informs the endpoint if it should abort or ask to continue on timeout. (Section 7.1)

ENABLE - The "ENABLE" parameter in CAP is used to tag a property in a component as disabled or enabled. (Section 7.2)

ID - The "ID" parameter specifies a unique identifier to be used for any outstanding commands.

LATENCY - The "LATENCY" parameter supplies the timeout value for command completion to the other endpoint. (Section 7.4)

LOCAL - The "LOCAL" parameter in CAP is used to tag a property in a component to signify that the component is local or to be distributed. (Section 7.5)

LOCALIZE - The "LOCALIZE" parameter specifies the locale to be used in error and warning messages.

OPTIONS - The "OPTIONS" parameter passes optional information for the command being sent.

2.1.2. New or Updated Properties (summary)

ALLOW-CONFLICT - Some entries in a calendar might not be valid if other entries were allowed to overlap the same time span. (Section 8.1)

ATT-COUNTER - When storing a "METHOD" property with the "COUNTER" method, there needs to be a way to remember the "ATTENDEE" value that sent the COUNTER. (Section 8.2)

CAP-VERSION - The version of CAP that the implementation supports. (Section 8.5)

CAR-LEVEL - The level of calendar access supported. (Section 8.7)

COMPONENTS - The list of components supported. (Section 8.8)

CSID - The Calendar Store IDentifier (CSID) uniquely identifies a CAP server. (Section 8.9)

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 8.3)

CALMASTER - The "CALMASTER" property specifies the contact information for the CS. (Section 8.4)

CARID - Access rights can be saved and fetched by unique ID - the "CARID" property. (Section 8.6)

CMD - The CAP commands, as well as replies are transmitted using the "CMD" property. (Section 10.1)

DECREED - Some access rights are not changeable by the CUA. When that is the case, the "DECREED" property value in the "VCAR" component will be "TRUE". (Section 8.10)

DEFAULT-CHARSET - The list of charsets supported by the CS. The first entry is the default for the CS. (Section 8.11)

DEFAULT-LOCALE - The list of locales supported by the CS. The first entry in the list is the default locale. (Section 8.12)

DEFAULT-TZID - This is the list of known timezones supported. The first entry is the default. (Section 8.13)

DEFAULT-VCARS - A list of the "CARID" properties that will be used to create new calendars. (Section 8.14)

DENY - The UPNs listed in the "DENY" property of a "VCAR" component will be denied access, as described in the "VRIGHT" component. (Section 8.15)

EXPAND - This property tells the CS if the query reply should expand components into multiple instances. The default is "FALSE" and is ignored for CSs that cannot expand recurrence rules. (Section 8.16)

GRANT - The UPNs listed in the "GRANT" property of a "VCAR" component will be allowed access as described in the "VRIGHT" component. (Section 8.17)

ITIP-VERSION - The version of [iTIP] supported. (Section 8.18)

MAXDATE - The maximum date supported by the CS. (Section 8.20)

MAX-COMP-SIZE - The largest component size allowed in the implementation including attachments in octets. (Section 8.19)

MINDATE - The minimum date supported by the CS. (Section 8.21)

MULTIPART - Passed in the capability messages to indicate which MIME multipart types the sender supports. (Section 8.22)

NAME - The "NAME" property is used to add locale-specific descriptions into components. (Section 8.23)

OWNER - Each calendar has at least one "OWNER" property. (xref target="OWNER"/>) Related to the "CAL-OWNERS()" query clause. (Section 6.1.1.1)

PERMISSION - This property specifies the permission being granted or denied. Examples are the "SEARCH" and "MODIFY" values. (Section 8.25)

QUERY - Used to hold the CAL-QUERY (Section 8.26) for the component.

QUERYID - A unique id for a stored query. (Section 8.27)

QUERY-LEVEL - The level of the query language supported. (Section 8.28)

RECUR-ACCEPTED - If the implementation support recurrence rules. (Section 8.29)

RECUR-EXPAND - If the implementation support expanding recurrence rules. (Section 8.31)

RECUR-LIMIT - Any maximum limit on the number of instances the implementation will expand recurring objects. (Section 8.30)

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

RESTRICTION - In the final check when granting calendar access requests, the CS test the results of a command for the value of the "RESTRICTION" property in the corresponding "VRIGHT" component, to determine if the access meets that restriction. (Section 8.32)

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 8.33)

SEQUENCE - When the "SEQUENCE" property is used in a "VALARM" component, it uniquely identifies the instances of the "VALARM" within that component.

STORES-EXPANDED - Specifies if the implementation stores recurring objects expanded or not. (Section 8.34)

TARGET - The new "VCALENDAR" component property "TARGET" (Section 8.35) is used to specify which calendar(s) will be the subject of the CAP command.

TRANSP - This is a modification of the [iCAL] "TRANSP" property and it allows more values. The new values are related to conflict control. (Section 8.36)

2.1.3. New Components (summary)

VAGENDA - CAP allows the fetching and storing of the entire contents of a calendar. The "VCALENDAR" component is not sufficient to encapsulate all of the needed data that describes a calendar. The "VAGENDA" component is the encapsulating object for an entire calendar. (Section 9.1)

VCALSTORE - Each CS contains one or more calendars (VAGENDAs), the "VCALSTORE" component is the encapsulating object that can hold all of the "VAGENDA" components along with any components and properties that are unique to the store level. (Section 9.2)

VCAR - Calendar Access Rights are specified and encapsulated in the new iCalendar "VCAR" component. The "VCAR" component holds some new properties and at least one "VRIGHT" component. (Section 9.3)

VRIGHT - This component encapsulates a set of instructions to the CS to define the rights or restrictions needed. (Section 9.4)

VREPLY - This component encapsulates a set of data that can consist of an arbitrary number of properties and components. Its contents are dependent on the command that was issued. (Section 9.5)

VQUERY - The search operation makes use of a new component, called "VQUERY" and a new value type "CAL-QUERY" (Section 6.1.1). The "VQUERY" component is used to fetch objects from the CS. (Section 9.6)

2.2. Relationship of RFC-2446 (ITIP) to CAP

[iTIP] describes scheduling methods that 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" command values, 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 are as defined in [iTIP]. This memo makes no changes to any of the methods or procedures described in [iTIP]. In this memo, 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" state object by depositing an 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 the "BOOKED" state. If the object has a "METHOD" property, then the object is stored in the "UNPROCESSED" state.

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

The CUA could also decide not to book the object. In this case, the "UNPROCESSED" state objects could be removed from the CS, or the CUA could set those objects to the marked-for-delete state. The CUA could also ignore objects for later processing.

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 by changing the "PARTSTAT" parameter to "DECLINED" in the "ATTENDEE" property that corresponds to their UPN. Then the CUA can perform [iTIP] processing such as sending back a decline, and then mark that object as marked-fo-delete. The 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" state entries can be in the CS for the same UID. However, once consolidated, only one object exists in the CS and that is the booked object. The other objects MUST be removed or have their state changed to "DELETED".

There MUST NOT be more than one "BOOKED" state object in a calendar for the same "UID". The "ADD" method value may create multiple objects in the "BOOKED" state for the same UID; however, for the purpose of this memo, they are the same object and simply have multiple "VCALENDAR" components.

For example, if you were on vacation, you could have received a "REQUEST" method to attend a meeting and several updates to that meeting. Your CUA would have to issue "SEARCH" commands to find them in the CS using CAP, process them, and determine 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 object from the consolidated results. Finally, the CUA could do a "DELETE" command to remove the related "UNPROCESSED" state objects. 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 CUA to send commands to, and receive responses from, a CS.

The CUA prepares a [MIME] encapsulated message, sends it to the CS, and receives a [MIME] encapsulated response. The calendaring-related information within these messages are represented by iCalendar objects. In addition, the "GET-CAPABILITY" command can be sent from the CS to the CUA.

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

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 9.2) 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 root of the calendar store and contains all of the calendars.

Calendar Store

```
      VCALSTORE
      |
      +-- properties
      +-- VCARs
      +-- VQUERYs
      +-- VTIMEZONEs
      +-- VAGENDA
      |     |
      |     +--properties
      |     +--VEVENTs
      |     |    |
      |     |    +--VALARMs
      |     +--VTODOs
      |     |    |
      |     |    +--VALARMs
      |     +--VJOURNALs
      |     +--VCARs
      |     +--VTIMEZONEs
      |     +--VQUERYs
      |     +--VFREEBUSYs
      |     |
      |     |   ...
      .
      .
      +-- VAGENDA
      .     .
      .     .
      .     .
```

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 a 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 a locale is supplied, the first locale in the [BEEP] ’localize’ attribute is 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" 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 on which the CS listens for connections is 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 that 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. Here initiator and listener 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
```

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 because 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 some of the commands. (Section 10.2)

CONTINUE - Sent to continue processing a command that has reached its specified timeout time. (Section 10.3)

CREATE - Create a new object on the CS. Initiated only by the CUA. (Section 10.4)

SET-LOCALE - Tell the CS to use any named locale and charset supplied. Initiated by the CUA only. (Section 10.13)

DELETE - Delete objects from the CS. Initiated only by the CUA. Can also be used to mark an object for deletion. (Section 10.5)

GENERATE-UID - Generate one or more unique ids. Initiated only by the CUA. (Section 10.6)

GET-CAPABILITY - Query the capabilities of the other end point of the session. (Section 10.7)

IDENTIFY - Set a new identity for the session. Initiated only by the CUA. (Section 10.8)

MODIFY - Modify components. Initiated by the CUA only. (Section 10.9)

MOVE - Move components to another container. Initiated only by the CUA. (Section 10.10)

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 10.11)

SEARCH - Search for components. Initiated only by the CUA. (Section 10.12)

TIMEOUT - Sent when a specified amount of time has lapsed and a command has not finished. (Section 10.14)

4. Security Model

BEEP transport performs all session authentication.

4.1. Calendar User and UPNs

A 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.

Quoted 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 2822 name [RFC2822] 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.example.com" is the same as "FANFEEDBACK@REDSOX.EXAMPLE.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 [X509CRL])

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" properties said, for example, "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 search and write components, properties, and parameters on calendars within a CS.

The "VCAR" component 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" component might be created before or after the actual "VCAR" component is defined. In addition, the "VCAR" and "VEVENT" components 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" component that conflicts with a "VCAR" components with a "DECREED" property set to the "TRUE" value 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. Predefined VCARs

The predefined calendar access CARIDs that MUST be implemented are:

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

```
BEGIN:VCAR
CARID:READBUSYTIMEINFO
BEGIN:VRIGHT
GRANT:*
PERMISSION:SEARCH
SCOPE:SELECT * FROM VFREEBUSY WHERE STATE() = ’BOOKED’
END:VRIGHT
END:VCAR
```

CARID:REQUESTONLY - Specifies the "GRANT" and "DENY" rules to UPNs other than the owner of the calendar and specifies the ability to write new objects with the "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 CAL-OWNERS()
PERMISSION:CREATE
RESTRICTION:SELECT VEVENT FROM VAGENDA
    WHERE METHOD = 'REQUEST'
RESTRICTION:SELECT VTODO FROM VAGEND
    WHERE METHOD = 'REQUEST'
RESTRICTION:SELECT VJOURNAL FROM VAGEND
    WHERE METHOD = 'REQUEST'
END:VRIGHT
END:VCAR
```

CARID:UPDATEPARTSTATUS - Grants 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 an "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" command access. They are not allowed to change the "ATTENDEE" property value. An example definition for this VCAR (only affecting the "VEVENT" components) is:

```
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:CAL-OWNERS()
            PERMISSION:*
            SCOPE:SELECT * FROM VAGENDA
            END:VRIGHT
            END:VCAR
```

4.2.3. 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 "VCAR" components that are decreed depending on the implementation. To the CUA, any "VCAR" component with the "DECREED" property set to "TRUE" cannot be changed by the currently authenticated UPN, and, depending on the implementation and other "VCAR" components, might not be able to be changed by any UPN using CAP (never when the CUA gets a "DECREED:TRUE" VCAR).

When a user attempts to modify or override a decreed "VCAR" component rules, 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 "VCAR" components MUST be readable by the calendar owner in standard "VCAR" component 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 CS 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 it 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 (e.g.,"@example.com") may be used to mean any identity from that Realm. This is useful to grant access rights to all users in a given Realm. A UPN with a non-null Username and null Realm (e.g., "bob@") could be a security risk and MUST NOT be used.

Because 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 both calendar stores and calendars accessible using the CAP protocol.

The CAP URL scheme conforms 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://" csidpart [ "/" relcalid ]
                ;
csidpart = hostport   ; As defined in Section 3.2.2 of RFC 2396
                      ;
relcalid = *uric      ; As defined in Section 2 of RFC 2396
```

A ’relcalid’ is an identifier that uniquely identifies a calendar on a particular calendar store. There is no implied structure in a Relative CALID (relcalid). 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.

Here are some examples:

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

A ’relcalid’ is permitted and is resolved according to the rules defined in Section 5 of RFC 2396.

Examples of valid relative CAP URLs:

```
opqaueXzz123String
UserName/Personal
```

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 Value Types

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" property "2.0" value 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 CSs look like tables for the purpose of a QUERY, and all of their properties look like columns in those tables.

3. You MUST NOT do any cross-component-type joins. That means you can ONLY have one component OR one "VAGENDA" component OR one "VCALSTORE" component in the "FROM" clause.

4. Everything in the "SELECT" clause and "WHERE" clauses MUST be from the same component type or "VAGENDA" component OR "VCALSTORE" component 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 that each have a single "QUERY" property.

6. The '.' is used to separate the table name (component) and column name (property or component) when selecting a property that is contained inside 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 encapsulating BEGIN/END statement will be supplied for "component-name".

In the following example, '.' is used to separate the "TRIGGER" property from its contained component (VALARM), which is contained in any "VEVENT" component in the selected "TARGET" property value (a relcalid). All "TRIGGER" properties in any "VEVENT" component in relcalid would be returned.

```
TARGET:relcalid
QUERY:SELECT VALARM.TRIGGER FROM VEVENT
SELECT VALARM FROM VEVENT WHERE UID = "123"
```

This returns one BEGIN/END "VALARM" component for each "VALARM" component in the matching "VEVENT" component. As there is no '.' (dot) in the VALARM after the SELECT above, it returns:

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

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

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

will return all of the properties in each "VALARM" component in the matching "VEVENT" component:

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

In the following SELECT clauses:

```
   (a) SELECT <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’
```

Clause (a) elects all instances of <a-property-name> from all "VEVENT" components.

Clauses (b) and (c) select all "VALARM" components from all "VEVENT" components. (b) would return them in BEGIN/END VALARM tags. (c) would return all of the properties without BEGIN/END VALARM tags.

Clause (d) selects every property and every component that is in any "VEVENT" component, with each "VEVENT" component wrapped in a BEGIN/END VEVENT tags.

Clause (e) selects all properties and all contained components in all "VEVENT" components that have a "VALARM" component with a "TRIGGER" property value between the provided dates and times, with each "VEVENT" component wrapped in BEGIN/END VEVENT tags.

Here are two invalid SELECT clauses:

```
   (f) SELECT VEVENT.VALARM.TRIGGER FROM VEVENT

   (g) SELECT DTSTART,UID FROM VEVENT
         WHERE VTODO.SUMMERY = "Fix typo in CAP"
```

Clause (f) is invalid because it contains two '.' characters.

Clause (g) Is invalid because it mixes VEVENT and VTODO properties in the same VQUERY.

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

```
  cal-query  = "SELECT"   SP   cap-val  SP
               "FROM"     SP   comp-name SP
               "WHERE"    SP   cap-expr

             / "SELECT" SP cap-cols SP
               "FROM"   SP comp-name
               ;
  cap-val    = cap-cols / param
             / ( cap-val "," cap-val )

               ; NOTE: there is NO space around the "," on
               ; the next line
  cap-cols   = cap-col / ( cap-cols "," cap-col)
               / "*"
               / "*.*" ; only valid when the target is a "VAGENDA"
               ;
               ; A 'cap-col' is:
               ;
               ; Any property name ('cap-prop') found in the
               ; component named in the 'comp-name' used in the
```

```
                  ; "FROM" clause.
                  ;
                  ;   SELECT ORGANIZER FROM VEVENT ...
                  ;
                  ; OR
                  ;
                  ; A component name ('comp-name') of an existing
                  ; component contained inside of the 'comp-name'
                  ; used in the "FROM" clause.
                  ;
                  ;   SELECT VALARM FROM VEVENT ...
                  ;
                  ; OR
                  ;
                  ; A component name ('comp-name') of an existing
                  ; component contained inside of the 'comp-name' used
                  ; in the "FROM" clause followed by a property
                  ; name ('cap-prop') to be selected from that
                  ; component.
                  ; (comp-name "." cap-prop)

                  ;   SELECT VALARM.TRIGGER FROM VEVENT ...

     cap-col    = comp-name
                / comp-name "." cap-prop
                / cap-prop

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

     cap-prop   = ; A property that may be in the 'cap-comp' named
                  ; in the "SELECT" clause.

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

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

     cap-logical= "AND" / "OR"

     cap-factor = cap-colval SP cap-oper SP col-value
                / cap-colval SP "LIKE" SP col-value
                / cap-colval SP "NOT LIKE" SP col-value
                / cap-colval SP "IS NULL"
                / cap-colval SP "IS NOT NULL"
                / col-value SP "IN" cap-colval
                / col-value SP "NOT IN" cap-colval
                / "STATE()" "=" ( "BOOKED"
                                / "UNPROCESSED"
                                / "DELETED"
                                / iana-state
                                / x-state )
                  ;
     iana-state = ; Any state registered by IANA directly or
                  ; included in an RFC that may be applied to
                  ; the component and within the rules published.
                  ;
     x-state    = ; Any experimental state that starts with
                  ; "x-" or "X-".

     cap-colval = cap-col /  param
                  ;
     param      = "PARAM(" cap-col "," cap-param ")"
                  ;
     cap-param  = ; Any parameter that may be contained in the cap-col
                  ; in the supplied PARAM() function

     col-value  = col-literal
                / "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
                   ; cannot 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.
                   ;
                   ; And, if the literal data contains any characters
                   ; that would have to be backslash-escaped if
                   ; a property or parameter value, then they must
                   ; be backslash-escaped in the literal-data.
                   ; Also, the quote character (') must be backslash
                   ; escaped.  Example:
                   ;
                   ; ... WHERE SUBJECT = 'It\'s time to ski'
                   ;
     cap-oper    = "="
                 / "!="
                 / "<"
                 / ">"
                 / "<="
                 / ">="
                   ;
     SP          = ; A single white space ASCII character
                   ; (value in HEX %x20).
                   ;
     x-comp      = ; As defined in [iCAL] section 4.6.
                   ;
     iana-comp   = ; As defined in [iCAL] section 4.6.
```

6.1.1.1. [NOT] CAL-OWNERS()

This function returns the list of "OWNER" properties for the named calendar when used in the "SELECT" clause.

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

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

If used in the "WHERE" clause, it returns true if the currently authenticated UPN is an owner of the currently selected object matched in the provided "TARGET" property. Used in a CAL-QUERY "WHERE" clause and in the UPN-FILTER.

6.1.1.2. CURRENT-TARGET()

This is equivalent to the value of the "TARGET" property in the current command. It is used in a CAL-QUERY "WHERE" clause.

6.1.1.3. PARAM()

This is used in a CAL-QUERY. It returns or tests for the value of the named parameter from the named property.

6.1.1.3.1. PARAM() in SELECT

When used in a "SELECT" clause, it returns the entire property and all of that property’s parameters; the result is not limited to the supplied parameter. If the property does not contain the named parameter, then the property is not returned. However, it could be returned as a result of another "SELECT" clause value. If multiple properties of the supplied name have the named parameter, all properties with that named parameter are returned. If multiple PARAM() clauses in a single "SELECT" CLAUSE match the same property, then the single matching property is returned only once.

Also, note that many parameters have default values defined in [iCAL] that must be treated as existing with their default value in the properties, as defined in [iCAL], even when not explicitly present. For example, if a query were performed with PARAM(ATTENDEE,ROLE) then ALL "ATTENDEE" properties would match because, even when they do not explicitly contain the "ROLE" parameter, it has a default value and therefore must match.

Therefore, when PARAM() is used in a "SELECT" clause, it is more accurate to say that it means return the property, if it contains the named parameter explicitly in the property or simply because the parameter has a default for that property.

6.1.1.3.2. PARAM() in WHERE

When PARAM() is used in the "WHERE" clause, a match is true when the parameter value matches the compare clause (according to the supplied WHERE values). If multiple named properties contain the named parameter, then each parameter value is compared in turn to the condition; if any match, the results would be true for that condition the same as if only one had existed. Each matching property or component is returned only once.

Because a parameter may be multi-valued, the comparison might need to be done with an "IN" or "NOT IN" comparator.

Given the following query:

```
ATTENDEE;PARTSTAT=ACCEPTED:cap://host.com/joe

SELECT VEVENT FROM VAGENDA
 WHERE PARAM(ATTENDEE,PARTSTAT) = 'ACCEPTED'
```

Thus, all "VEVENT" components that contain one or more "ATTENDEE" properties that have a "PARTSTAT" parameter with a "ACCEPTED" value would be returned. Also, each uniquely matching VEVENT would be returned only once, no matter how many "ATTENDEE" properties had matching roles, in each unique "VEVENT" component.

Also note that many parameters have default values defined in [iCAL]. Therefore, if the following query were performed on the "ATTENDEE" property in the above example:

```
SELECT VEVENT FROM VAGENDA
 WHERE PARAM(ATTENDEE,ROLE) = 'REQ-PARTICIPANT'
```

It would return the "ATTENDEE" property shown above because the default value for the "ROLE" parameter is "REQ-PARTICIPANT".

6.1.1.4. SELF()

Used in a CAL-QUERY "WHERE" clause. Returns the UPN of the currently authenticated UPN or their current UPN as a result of an IDENTIFY command.

6.1.1.5. STATE()

Returns one of three values, "BOOKED", "UNPROCESSED", or "DELETED" depending on the state of the object. "DELETED" is a component in the marked-for-delete state. Components that have been removed from the store are never returned.

If not specified in a query then both "BOOKED" and "UNPROCESSED" data is returned. Each unique "METHOD" property must be in a separate MIME object, per the [iCAL] section 3.2 restriction.

6.1.1.6. 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 [iCAL]. Any "LIKE" clause wildcard characters that are part of any literal data that is preceded by a "LIKE" clause or "NOT LIKE" clause and is not intended to mean wildcard search MUST be escaped as described in note (7) below.

6.1.1.7. Comparing DATE and DATE-TIME Values

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

Local time event, as described in section 4.2.19 of [iCAL], must be considered to be in the CUA default timezone that was supplied by the CUA in the "CAPABILITY" exchange.

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 "DATE" values and "DATE-TIME" values are compared with the "LIKE" clause, the comparison will be done as if the value is a [iCAL] 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" clause value of "%00%", would return any value that contained two consecutive zeros.

All comparisons will be done in UTC.

6.1.1.8. DTEND and DURATION

The "DTEND" property value is not included in the time occupied by the component. That is, a "DTEND" property value of 20030614T12000 includes all of the time up to, but not including, noon on that day.

The "DURATION" property value end time is also not inclusive. So an object with a "DTSTART" property value of 20030514T110000 and a "DURATION" property value of "1H" does not include noon on that day.

When a "QUERY" property value 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" property supplied "DTEND" value or any range of values supplied by the "QUERY" property.

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

6.1.1.9. [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" clause 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. Thus, 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 insensitive comparisoison assumes ’a’ = ’A’).

If the "LIKE" clause 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. THE CS must perform the comparisons as if each value existed by itself and was not quoted or backslash-escaped, when comparing using the LIKE element.

See related examples in Section 6.1.1.11.

6.1.1.10. Empty vs. NULL

When used in a CAL-QUERY value, "NULL" means that the property or parameter is not present in the object. Paramaters that are not provided and have a default value in the property are considered to exist with their default value and will not be "NULL".

> If the property exists but has no value, then "NULL" MUST NOT match.
>
> If the parameter exists but has no value, then "NULL" MUST NOT match.
>
> If the parameter not present and has a default value, then "NULL" MUST NOT match.
>
> If the property (or parameter) exists but has no value, then it matches the empty string '' (quote quote).

6.1.1.11. [NOT] IN

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

Some iCalendar objects can be multi-instance and multi-valued. The "IN" clause 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)         property:parameter=1,2:x     One parameter, two values.
   (d)         property:parameter="1,2",3:y  One parameter, one value.
   (e)         property:parameter=",":z     One parameter, one value.
   (f)         property:x,y,z               One property, three values
             END:A-COMPONENT
```

In this example:

```
   '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-instance 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 the "IN" clause is preceded by 'NOT', then there is a match when the value does not exist in the property or parameter value.

6.1.1.12. DATE-TIME and TIME Values in a WHERE Clause

All "DATE-TIME" and "TIME" literal values supplied in a "WHERE" 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; it is a syntax error and the CS MUST reject the QUERY:

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

6.1.1.13. Multiple Contained Components

If a query references a component and a component or property contained in the component, any clauses referring to the contained component or property must be evaluated on all of the contained components or properties. If any of the contained components or properties match the query, and the conditions on the containing component are also true, the component matches the query.

For example, in the query below, if a BOOKED VEVENT contains multiple VALARMs, and the VALARM.TRIGGER clause is true for any of the VALARMs in the VEVENT, then the UID, SUMMARY, and DESCRIPTION of this VEVENT would be included in the QUERY results.

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

6.1.1.14. 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 it would 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
```

6.1.1.15. Query by Date-Time Range

This query selects the entire content of every booked "VEVENT" component that has an instance greater than or equal to July 1, 2000 00:00:00 UTC and less than or equal to July 30, 2000 23:59:59 UTC. This includes single instance "VEVENT" components that do not explicitly contain any recurrence properties or "RECURRENCE-ID" properties. This works only for CSs that have the "RECUR-

EXPAND" property value set to "TRUE" in the "GET-CAPABILITY" exchange.

```
      BEGIN:VQUERY
      EXPAND:TRUE
      QUERY:SELECT * FROM VEVENT
      WHERE RECURRENCE-ID >= '20000701T000000Z'
      AND RECURRENCE-ID <= '20000730T235959Z'
      AND STATE() = 'BOOKED'
      END:VQUERY
```

6.1.1.16. Query for All Unprocessed Entries

The following example selects the entire contents of all non-booked "VTODO" and "VEVENT" components in the "UNPROCESSED" state. The default for the "EXPAND" property 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 in the "BOOKED" state.

```
      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" property 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() = 'DELETED'
      QUERY:SELECT UID FROM VTODO WHERE STATE() = 'DELETED'
      END:VQUERY
```

6.1.1.17. 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 have a "DTSTART" value from February 1st to February 10th 2000 (in UTC) will also be selected.

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

6.1.1.18. 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 a CU or a group of CUs.

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 [RFC2822]
                 ;
                 ;
        dot-atom-text = ; As defined in [iCAL].
```

Description: This data type is an identifier that denotes a CU or a group of CU. A UPN is an RFC 2822-compliant email address [RFC2822], 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. And 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 its simplest form it looks like "user@example.com".

In certain cases a UPN will not be RFC 2822-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".

Example:

The following is a UPN for a CU:

```
jdoe@example.com
```

The following is an example of a UPN that could be for a group of CU:

```
staff@example.com
```

The following is a UPN for an anonymous CU that belongs to a specific realm. When used as a UPN-FILTER, it applies to all UPNs in a specific realm:

```
@example.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:

```
                     ;
                     ; NOTE: "CAL-OWNERS(cal-address)"
                     ;       and "NOT CAL-OWNERS(cal-address)"
                     ;       are both NOT allowed below.
                     ;
```

```
         upn-filter    = "CAL-OWNERS()" /
                         "NOT CAL-OWNERS()" /
                         "*" /
                   [ "*" / 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 "CAL-OWNERS()" and "NOT CAL-OWNERS()", see Section 8.24.

*	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.
user@	Not allowed.

Example: The following are examples of this value type:

```
DENY:NON CAL-OWNERS()
DENY:@hackers.example.com
DENY:*@hackers.example.com
GRANT:sam@example.com
```

7. New Parameters

7.1. ACTION Parameter

Parameter Name: ACTION

Purpose: This parameter indicates the action to be taken when a timeout occurs.

Value Type: TEXT

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

When present in a "CMD" property, the "ACTION" parameter specifies the action to be taken when the command timeout expires.

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

```
     action-param     = ";" "ACTION" "=" ( "ASK" / "ABORT" )
                           ; If ’action-param’ is supplied then
                           ; ’latency-param’ MUST be supplied.
```

Example:

```
     CMD;LATENCY=10;ACTION=ASK:CREATE
```

7.2. ENABLE Parameter

Parameter Name: ENABLE

Purpose: This parameter indicates whether or not the property should be ignored. For example, it can indicate that a "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 if the "ENABLE" property is not in the "TRIGGER" property, the alarm is enabled. This parameter may not be known by pre-CAP implementations, but this should not be an issue as it conforms to an 'ianaparam' [iCAL].

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

```
     enable-param       = "ENABLE" "=" boolean
                          ;
     boolean            = ; As defined in [iCAL].
```

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

```
     TRIGGER;ENABLE=FALSE;RELATED=END:PT5M
```

7.3. ID Parameter

Parameter Name: ID

Purpose: When used in a "CMD" component, it provides a unique identifier.

Value Type: TEXT

Conformance: This parameter can be specified in the "CMD" property.

Description: If more than one command is sent, then the "ID" parameter is used to uniquely identify the command.

A CUA may add the "ID" parameter to any "CMD" property before sending the command. There must not be more than one outstanding command tagged with the same "ID" parameter value.

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

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

Example: The following is an example of this parameter component:

```
         CMD;UD=some-unique-value:CREATE
```

7.4. LATENCY Parameter

Parameter Name: LATENCY

Purpose: This parameter indicates time in seconds for when a timeout occurs.

Value Type: TEXT

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

When present in a "CMD" property, the "LATENCY" parameter specifies the time in seconds when the command timeout expires.

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

```
         latency-param    = ";" "LATENCY" "=" latency-sec
                          ; The value supplied in the time in seconds.
                          ; If 'latency-param' is supplied then
                          ; 'action-param' MUST be supplied.
                          ;
         latency-sec      = posint1

                          ; Default is zero (0) meaning no timeout.
```

Example: The following is an example of this parameter:

```
         CMD;LATENCY=10;ACTION=ASK:CREATE
```

7.5. LOCAL Parameter

Parameter Name: LOCAL

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

Value Type: BOOLEAN

Conformance: This parameter 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 "TRUE", then the alarm MUST NOT be forwarded to any other calendar. If set to "FALSE", or if the "LOCAL" parameter is not in the "SEQUENCE" property, the alarm is global.

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

```
   local-param        = "LOCAL" "=" boolean
```

Example: The following is an example of this parameter:

```
   SEQUENCE;LOCAL=TRUE:4
```

7.6. LOCALIZE Parameter

Parameter Name: LOCALIZE

Purpose: If provided, specifies the desired language for error and warning messages.

Value Type: TEXT

Conformance: This parameter can be specified in the "CMD" properties.

When the "LOCALIZE" parameter is supplied, its value MUST be one of the values listed in the initial [BEEP] greeting ’localize’ attribute.

A CUA may add the "LOCALIZE" parameter to the "CMD" property to specify the language of any error or warning messages.

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

```
   localize-param   = ";" "LOCALIZE" "=" beep-localize
                    ;
   beep-localize    = text ; As defined in [BEEP]
                    ; 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.
```

Example: The following is an example of this parameter:

```
   CMD;LOCALIZE=fr_CA:CREATE
```

7.7. OPTIONS Parameter

Parameter Name: OPTIONS

Purpose: If provided the "OPTIONS" parameter specifies some "CMD" property-specific options.

Value Type: TEXT

Conformance: This parameter can be specified in the "CMD" properties.

A CUA adds the "OPTIONS" parameter to the "CMD" property when the command needs extra values.

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

```
     option-param     = ";" "OPTIONS" "=" cmd-specific
                        ;
     cmd-specific     = ; The value supplied is dependent on the
                        ; CMD value.  See the specific CMDs for the
                        ; correct values to use for each CMD.
```

Example: The following is an example of this parameter:

```
     CMD;OPTIONS=10:GENERATE-UID
```

8. New Properties

8.1. 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, whether 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" component, then all "VAGENDA" component "ALLOW-CONFLICT" property values MUST be "FALSE" in the CS.

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

```
      allow-conflict     = "ALLOW-CONFLICT" other-params ":" boolean
   CRLF
```

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

```
      ALLOW-CONFLICT:FALSE
```

8.2. ATT-COUNTER Property

Property Name: ATT-COUNTER

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

Conformance: This property MUST be specified in an iCalendar object that specifies a counter proposal to a group-scheduled calendar entity. When storing a "METHOD" property with the "COUNTER" method, there needs to be a way to remember who sent the COUNTER. The ATT-COUNTER property MUST be added to all "COUNTER" [iTIP] components by the CUA before storing in a CS.

Description: This property is used to identify the CAL-ADDRESS of the entity that sent the "COUNTER" [iTIP] object.

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

```
      attcounter   = "ATT-COUNTER" other-params ":" cal-address CRLF
```

Examples:

```
         ATT-COUNTER:cap:example.com/Doug
         ATT-COUNTER:mailto:Doug@Example.com
```

8.3. 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" component.

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

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

```
      calid   = "CALID" other-params ":" relcalid CRLF
```

Example:

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

8.4. 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 [URL]. 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.

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

```
   calmaster = "CALMASTER" other-params ":" uri CRLF
             ;
   uri       = ; IANA registered uri as defined in [iCAL].
```

Example: The following is an example of this property:

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

8.5. CAP-VERSION Property

Property Name: CAP-VERSION

Purpose: This property specifies the version of CAP supported.

Value Type: TEXT

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

Conformance: This property is specified in the "VREPLY" component that is sent in response to a "GET-CAPABILITY" command.

Description: This specifies the version of CAP that the endpoint supports. The list is a comma-separated list of supported RFC numbers. The list MUST contain at least 4324.

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

```
     cap-version   = "CAP-VERSION" other-params ":" text CRLF
```

Example: The following are examples of this property:

```
     CAP-VERSION:4324
```

8.6. 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. A "CARID" property value is unique per container.

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

```
         carid      = "CARID" other-params ":" text CRLF
```

Example: The following are examples of this property:

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

8.7. CAR-LEVEL Property

Property Name: CAR-LEVEL

Purpose: The property specifies the level of VCAR supported.

Value Type: TEXT

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

Conformance: The property can be specified in a "VREPLY" component that is sent in response to a "GET-CAPABILITY" command.

Description: The value is one from a list of "CAR-NONE", "CAR-MIN", or "CAR-FULL-1". If "CAR-FULL-1" is supplied, then "CAR-MIN" is also available. A "CAR-MIN" implementation only supported the "DEFAULT-VCARS" property values listed in the "VCALSTORE" component, and a "CAR-MIN" implementation does not support the creation or modification of "VCAR" components from the CUA.

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

```
         car-level        = "CAR-LEVEL" ":" other-params ":"
                                              car-level-values

         car-level-values = ( "CAR-NONE" / "CAR-MIN" / "CAR-FULL-1"
                              / other-levels )

         other-levels     = ; Any name published in an RFC for a
                            ; "CAR-LEVEL" property value.
```

Example: The following is an example of this property:

```
            CAR-LEVEL:CAR-FULL-1
```

8.8. COMPONENTS Property

Property Name: COMPONENTS

Purpose: The property specifies a the list of components supported by the endpoint.

Value Type: TEXT

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

Conformance: The property can be specified in a "VREPLY" component in response to a "GET-CAPABILITY" command.

Description: A comma-separated list of components that are supported by the endpoint. A component that is not in the list sent from the endpoint is not supported by that endpoint. Sending an unsupported component results in unpredictable results. This includes any components inside of other components (VALARM for example). The recommended list is "VCALSTORE,VCALENDAR,VREPLY,VAGENDA, VEVENT,VALARM,VTIMEZONE,VJOURNAL,VTODO,VALARM, DAYLIGHT,STANDARD,VCAR,VRIGHT,VQUERY".

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

```
   components     = "COMPONENTS" other-params ":" comp-list CRLF
                  ;
                  ; All of these MUST be supplied only once.
                  ;
   comp-list-req  = "VCALSTORE" "," "VCALENDAR" "," "VTIMEZONE" ","
                    "VREPLY"    "," "VAGENDA"   "," "STANDARD"  ","
                    "DAYLIGHT"
                  ; At least one MUST be supplied. The same value
                  ; MUST NOT occur more than once.
                  ;
   comp-list-min  = ( "," "VEVENT")
                  / ( "," "VTODO")
                  / ( "," "VJOURNAL" )
                  ; The same value MUST NOT occur
                  ; more than once.  If "VCAR" is supplied then
                  ; "VRIGHT" must be supplied.
                  ;
   comp-list-opt  = ( "," "VFREEBUSY" ) / ( "," "VALARM" )
                    / ( "," "VCAR" )    / ( "," "VRIGHT" )
                    / ( "," "VQUERY")   / ( "," x-comp )
                    / ( "," iana-comp )
                  ;
   comp-list      = comp-list-req 1*3comp-list-min *(comp-list-opt)
```

Example: The following is an example of this property:

```
COMPONENTS:VCALSTORE,VCALENDAR,VREPLY,VAGENDA,
VEVENT,VALARM,VTIMEZONE,VJOURNAL,VTODO,
DAYLIGHT,STANDARD,VFREEBUSY,VCAR,VRIGHT,VQUERY
```

8.9. CSID Property

Property Name: CSID

Purpose: The property specifies a 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. Each CS needs its own unique identifier. The "CSID" property is the official unique identifier for the CS. If the BEEP ’serverName’ attribute was supplied 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’ attribute.

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

```
csid = "CSID" other-params ":" capurl CRLF
```

Example: The following is an example of this property:

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

8.10. 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. If the "DECREED" property value is "TRUE" then the CUA will be unable to change the contents of the "VCAR" component and any attempt will fail with an error.

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

```
decreed      = "DECREED" other-params ":" boolean CRLF
```

Example: The following is an example of this property:

```
DECREED:TRUE
```

8.11. 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" component this property is used to indicate the charset of calendar. If not specified, the default is the first value in the "VCALSTORE" components "DEFAULT-CHARSET" property value list. The value MUST be an IANA registered character set as defined in [CHARREG].

In a "VCALSTORE" component it is a comma-separated list of charsets supported by the CS. The first entry is the default entry for all newly created "VAGENDA" components. The "UTF-8" value MUST be in the "VCALSTORE" component "DEFAULT-CHARSET" property list. All compliant

CAP implementations (CS and CUA) MUST support at least the "UTF-8" charset.

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

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

```
     default-charset     = "DEFAULT-CHARSET" other-params ":" text
     *( "," text) CRLF
```

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

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

8.12. 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" component, the "DEFAULT-LOCALE" property is used to indicate the locale of the calendar. The full locale SHOULD be used. The default and minimum locale is POSIX (aka the ’C’ locale).

In a "VCALSTORE" component, 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.

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

```
     default-locale     = "DEFAULT-LOCALE" other-params ":" language
                             *( "," language) CRLF
                ;
     language = ; Text identifying a locale, as defined in [CHARPOL]
```

Example: The following is an example of this property:

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

8.13. DEFAULT-TZID Property

Property Name: DEFAULT-TZID

Purpose: This property specifies the text value that specifies the time zones.

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: A multi-valued property that lists the known time zones. The first is the default. Here "TZID" property values are the same as the "TZID" property defined in [iCAL].

If used in a "VCALSTORE" component, it is a comma-separated list of TZIDs known to the CS. The entry is used as the default TZID list for all newly created calendars. The list MUST contain at least "UTC". A "VCALSTORE" components MUST contain one "VTIMEZONE" component for each value in the "DEFAULT-TZID" property value.

If used in a "VAGENDA" component, it is a comma-separated list of "TZID" property values naming the time zones known to the calendar. The first time zone in the list is the default and is used as the localtime for objects that contain a date or date-time value without a time zone. All "VAGENDA" components MUST have one "VTIMEZONE" component contained for each value in the "DEFAULT-TZID" property value.

If a "TZID" property value contains a comma (,), the comma must be backslash-escaped.

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

```
     default-tzid       = "DEFAULT-TZID" other-params
                          ":" [tzidprefix] text
                          *("," [tzidprefix] text) CRLF
                          ;
     txidprefix         = ; As defined in [iCAL].
```

Example: The following is an example of this property:

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

8.14. DEFAULT-VCARS Property

Property Name: DEFAULT-VCARS

Purpose: This property is used to specify the "CARID" property ids 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" component to specify the "CARID" value of the "VCAR" components that MUST be copied into now "VAGENDA" components at creation time by the CS. All "DEFAULT-VCAR" values must have "VCARS" components stored in the "VCALSTORE".

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

```
        defautl-vcars      = "DEFAULT-VCARS" other-params ":" text
        *( "," text ) CRLF
```

Example: The following is an example of this property:

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

8.15. 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" components.

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

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

```
deny       = "DENY" other-params ":" upn-filter CRLF
```

Example: The following are examples of this property:

```
DENY:*

DENY:bob@example.com
```

8.16. EXPAND property

Property Name: EXPAND

Purpose: This property is used to notify the CS whether to 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" 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". Note that if the CS has its "RECUR-EXPAND" CS property value set to "FALSE", then the "EXPAND" property will be ignored and the result will be as if the "EXPAND" value was set to "FALSE". The results will be bounded by any date range or other limits in the query.

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

```
expand     = "EXPAND" other-params ":" ("TRUE" / "FALSE") CRLF
```

Example: The following are examples of this property:

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

8.17. 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.

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

```
     grant     = "GRANT" other-params ":" upn-filter CRLF
```

Example: The following are examples of this property:

```
     GRANT:*

     GRANT:bob@example.com
```

8.18. ITIP-VERSION Property

Property Name: ITIP-VERSION

Purpose: This property specifies the version of ITIP supported.

Value Type: TEXT

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

Conformance: This property is specified in the "VREPLY" component that is sent in response to a "GET-CAPABILITY" command.

Description: This specifies the version of ITIP that the endpoint supports. The list is a comma-separated list of supported RFC numbers. The list MUST contain at least 2446, which is [iTIP]

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

```
      itip-version   = "ITIP-VERSION" other-params ":" text CRLF
```

Example: The following are examples of this property:

```
      ITIP-VERSION:2446
```

8.19. MAX-COMP-SIZE Property

Property Name: MAX-COMP-SIZE

Purpose: This property specifies the largest size of any object accepted.

Value Type: TEXT

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

Conformance: This property is specified in the "VREPLY" component that is sent in response to a "GET-CAPABILITY" command.

Description: A positive integer value that specifies the size of the largest iCalendar object that can be accepted in octets. Objects larger than this will be rejected. A value of zero (0) means no limit. This is also the maximum value of any [BEEP] payload that will be accepted or sent.

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

```
      max-comp-size   = "MAX-COMP-SIZE" other-params ":" posint0 CRLF
```

Example: The following are examples of this property:

```
      MAX-COMP-SIZE:1024
```

8.20. MAXDATE Property

Property Name: MAXDATE

Purpose: This property specifies the date/time in the future, beyond which the CS or CUA 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" component.

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

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

```
      maxdate    = "MAXDATE" other-params ":" date-time CRLF
```

Example: The following is an example of this property:

```
      MAXDATE:20990101T000000Z
```

8.21. 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" component.

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

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

```
      mindate    = "MINDATE" other-params ":" date-time CRLF

      date-time = ; As defined in [iCAL].
```

Example: The following is an example of this property:

```
      MINDATE:19710101T000000Z
```

8.22. MULTIPART Property

Property Name: MULTIPART

Purpose: This property provides a comma-separated list of supported MIME multipart types supported by the sender.

Value Type: TEXT

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

Conformance: This property is specified in the "VREPLY" component that is sent in response to a "GET-CAPABILITY" command.

Description: This property is used in the in the "GET-CAPABILITY" command reply to indicate the MIME multipart types supported. A CS and CUA SHOULD support all registered MIME multipart types.

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

```
    multipart = "MULTIPART" other-params ":" text *( "," text) CRLF
```

Example: The following is an example of this property:

```
    MULTIPART:related,alternate,mixed
```

8.23. NAME Property

Property Name: NAME

Purpose: This property provides a localizable 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 component to specify a localizable display name. If more than one "NAME" properties are in a component, then they MUST have unique "LANG" parameters. If the "LANG" parameter is not supplied, then it defaults to the "VAGENDA" component’s "DEFAULT-LOCALE" first value. If the component is a "VAGENDA", then the default value is the "VAGENDA"s component’s "DEFAULT-LOCALE" first value. A "VCALSTORE" component’s "DEFAULT-LOCALE" first value is the default if the component is stored at the "VCALSTORE" level.

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

```
     name          = "NAME" nameparam ":" text CRLF
                     ;
     nameparam     = other-params [ ";" languageparam ] other-params
                     ;
     languageparam = ; As defined in [iCAL].
```

Example: The following is an example of this property:

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

8.24. 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 in a "VAGENDA" component.

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

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

```
     owner = "OWNER" other-params ":" upn CRLF
```

Example: The following is an example of this property:

```
     OWNER:jsmith@example.com
     OWNER:jdough@example.com
```

8.25. 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" components.

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

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

```
     permission  = "PERMISSION" other-params ":" permvalue CRLF
                   ;
     permvalue = ( "SEARCH" / "CREATE" / "DELETE"
                 / "MODIFY" / "MOVE" / all
                 / iana-cmd / x-cmd )
                   ;
     all         = "*"
                   ;
     iana-cmd    = ; Any command registered by IANA directly or
                   ; included in an RFC that may be applied as
                   ; a command.
                   ;
     x-cmd       = ; Any experimental command that starts with
                   ; "x-" or "X-".
```

Example: The following is an example of this property:

```
   PERMISSION:SEARCH
```

8.26. 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.

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

```
     query      = "QUERY" other-params ":" cal-query CRLF
```

Example: The following is an example of this property:

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

8.27. 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" property is used to specify the unique id for a query. A "QUERYID" property value is unique per container.

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

```
     queryid      = "QUERYID" other-params ":" text CRLF
```

Example: The following are examples of this property:

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

8.28. QUERY-LEVEL Property

Property Name: QUERY-LEVEL

Purpose: This property specifies the level of query supported.

Value Type: TEXT

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

Conformance: The property can be specified in the "VREPLY" component in response to a "GET-CAPABILITY" command.

Description: Indicates level of query support. CAL-QL-NONE is for CS’s that allow ITIP methods only to be deposited and nothing else.

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

```
     query-level = "QUERY-LEVEL" other-params
                     ":" ( "CAL-QL-1" / "CAL-QL-NONE") CRLF
```

Example: The following is an example of this property:

```
QUERY-LEVEL:CAL-QL-1
```

8.29. RECUR-ACCEPTED Property

Property Name: RECUR-ACCEPTED

Purpose: This property specifies if the endpoint supports recurring instances.

Value Type: BOOLEAN

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

Conformance: The property can be specified in the "VREPLY" component in response to a "GET-CAPABILITY" command.

Description: Indicates if recurrence rules are supported. If "FALSE" then the endpoint cannot process any kind of recurring rules.

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

```
recur-accepted = "RECUR-ACCEPTED" other-params ":" boolean CRLF
```

Example: The following is an example of this property:

```
RECUR-ACCEPTED:TRUE
RECUR-ACCEPTED:FALSE
```

8.30. RECUR-LIMIT Property

Property Name: RECUR-LIMIT

Purpose: This property specifies the maximum number of instances the endpoint will expand instances at query or storage time.

Value Type: INTEGER

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

Conformance: The property can be specified in the "VREPLY" component in response to a "GET-CAPABILITY" command.

Description: For implementations that have the "STORES-EXPANDED" value set to "TRUE", this value specifies the maximum number of instances that will be stored and fetched. For all implementations, this is the maximum number of instances that will be returned when the "EXPAND" parameter is specified as "TRUE" and the results contain an infinite or large number of recurring instances.

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

```
recur-limit = "RECUR-LIMIT" other-params ":" posint1 CRLF
```

Example: The following is an example of this property:

```
RECUR-LIMIT:1000
```

8.31. RECUR-EXPAND Property

Property Name: RECUR-EXPAND

Purpose: This property specifies if the endpoint can expand recurrences into multiple objects.

Value Type: BOOLEAN

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

Conformance: The property can be specified in the "VREPLY" component in response to a "GET-CAPABILITY" command.

Description: If "TRUE", then the endpoint can expand an object into multiple instances as defined by its recurrence rules when the "EXPAND" property is supplied. If "FALSE", then the endpoint ignores the "EXPAND" property.

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

```
recur-expand = "RECUR-EXPAND" other-params ":" boolean CRLF
```

Example: The following is an example of this property:

```
RECUR-EXPAND:TRUE
RECUR-EXPAND:FALSE
```

8.32. 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" components, but only when the "PERMISSION" property is set to "CREATE", "MODIFY", or "*" property value.

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 any specified "RESTRICTION" property values.

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

```
restriction  = "RESTRICTION" other-params ":" 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 ORGANIZER

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

8.33. SCOPE Property

Property Name: SCOPE

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

Value Type: CAL-QUERY

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

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

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

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

```
     scope    = "SCOPE" other-params ":" cal-query CRLF
```

Example: The following is an example of this property:

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

8.34. STORES-EXPANDED Property

Property Name: STORES-EXPANDED

Purpose: This property specifies if the sending endpoint expands recurrence rules prior to storing them into the CS.

Value Type: BOOLEAN

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

Conformance: This property can be specified in a "VREPLY" component in response to a "GET-CAPABILITY" command.

Description: If the value is "TRUE", then the endpoint expands recurrence rules and stores the results into the CS. If this is "TRUE", then the "RECUR-LIMIT" property is significant because an infinitely-recurring appointment will store no more than "RECUR-LIMIT" property values into the CS and all other instances will be lost.

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

```
     stores-expanded   = "STORES-EXPANDED" other-params ":" boolean
  CRLF
```

The following is an example of this property:

```
     STORES-EXPANDED:TRUE
     STORES-EXPANDED:FALSE
```

8.35. TARGET Property

Property Name: TARGET

Purpose: This property defines the container that the issued command will act upon. Its value is a capurl, as defined 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 property 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 that has a capurl matching the value.

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

```
target   = "TARGET" other-params ":" ( capurl / relcalid ) CRLF
```

Example: The following is an example of this property:

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

8.36. TRANSP Property

Property Name: TRANSP

Purpose: This property defines whether a component is transparent or not to busy-time searches. This is a modification to [iCAL] "TRANSP" property, 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.

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

```
   transp    = "TRANSP" other-params ":" 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, and no other object can overlap it:

```
   TRANSP:OPAQUE-NOCONFLICT
```

9. New Components

9.1. VAGENDA Component

Component Name: VAGENDA

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

Formal Definition: There are two formats of the "VAGENDA" component. (1) When it is being created, and (2) how it exists in the "VCALSTORE" component.

A "VAGENDA" component in a "VCALSTORE" component is defined by the following notes and ABNF notation:

CALSCALE - The value MUST be from the "VCALSTORE" "CALSCALE" property list. The default is the first entry in the VCALSTORE CALSCALE list.

CREATED - The timestamp of the calendar’s create date. This

```
            is a READ ONLY property in a "VAGENDA".

         LAST-MODIFIED - The timestamp of any change to the "VAGENDA"
            properties or when any component was last created, modified,
            or deleted.

      agenda      = "BEGIN" ":" "VAGENDA" CRLF
                    agendaprop
                    *(icalobject)     ; as defined in [iCAL]
                    "END" ":" "VAGENDA" CRLF

      agendaprop  = *(
                    ; The following MUST occur exactly once.
                    ;
                      allow-conflict / relcalid / calscale / created
                    / default-charset / default-locale
                    / default-tzid / last-mod
                    ;
                    ; 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 / other-props / x-comp
                  )

      icalobject = ; As defined in [iCAL].
                   ;
      created    = ; As defined in [iCAL].
                   ;
      related-to = ; As defined in [iCAL].
```

When creating a VAGENDA, use the following notation:

```
      agendac     = "BEGIN" ":" "VAGENDA" CRLF
                    agendacprop
                    *(icalobject)     ; as defined in [iCAL].
                    "END" ":" "VAGENDA" CRLF

      agendacprop  = *(
                     ; The following MUST occur exactly once.
                     ;
                       allow-conflict / relcalid / 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 / other-props / x-comp
                    )
```

To fetch all of the properties from the targeted "VAGENDA" component but do not fetch any components, use:

```
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
```

9.2. 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" component is an administrative task and not part of the CAP protocol.

The following are notes to some of the properties in the "VCALSTORE" component.

CALSCALE - A comma-separated list of CALSCALEs supported by this CS. All "VAGENDA" component calendar CALSCALE properties MUST be from this list. This list MUST contain at least "GREGORIAN". The default for newly created "VAGENDA" components is the first entry.

RELATED-TO - This is a multiple-instance property. There MUST be a "RELATED-TO" property for each of the "VAGENDA" components contained in the "VCALSTORE" component, each with the "RELTYPE" parameter value set to "CHILD". Other "RELATED-TO" properties may be included.

```
      CREATED -  The timestamp of the CS creation time.  This is a
         READ ONLY property.

      CSID -  The CSID of this calendar store.  This 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.  This is a READ ONLY property.  A suggested
         value is the fully-qualified host name or a fully-qualified
         virtual host name supported by the system.

      LAST-MODIFIED -  The timestamp when the Properties of the
         "VCALSTORE" component were last updated or calendars were
         created or deleted.  This is a READ ONLY PROPERTY.

   calstorec     = "BEGIN" ":" "VCALSTORE" CRLF
                  calstoreprop
                  *(vagendac)
                  "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-mod / maxdate / mindate
                       ;
                       ; the following are optional,
                       ; and MAY occur more than once
                       ;
                       / name / related-to / other-props / x-comp
                      )
                    ;

   vagendac     = ; As defined in [iCAL].
                  ;
   last-mod     = ; As defined in [iCAL].

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

      SELECT * FROM VCALSTORE

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

9.3. VCAR Component

Component Name: VCAR

Purpose: Provide a grouping of calendar access rights.

Formal 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 / decreed / other-props
        )
```

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 localizable display name.

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

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

In this example, all UPNs are given search 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:SEARCH
   SCOPE:SELECT DTSTART,DTEND FROM VEVENT
   END:VRIGHT
   END:VCAR
```

In these examples, full calendar access rights are given to the CAL-OWNERS(), 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:CAL-OWNERS()
   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, search, modify or delete calendar access 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 changing any VCAR:

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

9.4. VRIGHT Component

Component Name: "VRIGHT"

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

Formal 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 / other-props
       )
```

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 that 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 restrictions on commands and results. If the command does not match the restrictions, or if the results of the command do not match the restrictions, then it is an access violation.

9.5. VREPLY Component

Component Name: "VREPLY"

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

Formal 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 component is used by query and other commands to return data that does not fit into any other component. It may contain any valid property or component, even if they are not registered.

9.6. VQUERY Component

Component Name: VQUERY

Purpose: A component describes a set of objects to be acted upon.

Formal 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. If the "TARGET" property
          ; is supplied then the "QUERYID" property
          ; MUST be supplied.
          ;
           queryid / target
          ;
          ; 'expand' is OPTIONAL but MUST NOT occur
          ; more than once.
          ;
           expand
          ;
          ; the following are OPTIONAL, and MAY occur
          ; more than once
          ;
          / name / other-props
          ;
          ; the following MUST occur at least once if
          ; queryid is not supplied.
          ;
          / query
        )
```

Description: A "VQUERY" contains properties that describe which properties and components the CS is requested to act upon.

The "QUERYID" property specifies the local identifier for a "VQUERY" component.

For a search, if the "TARGET" property is supplied in a "VQUERY" component, then the CS is to search for the query in the CALID supplied by the "TARGET" property value.

For a create, the "TARGET" property MUST NOT be supplied because the destination container is already supplied in the "TARGET" property of the "VCALENDAR" component.

Examples: see Section 6.1.1.

10. Commands and Responses

CAP commands and responses are described in this section.

10.1. CAP Commands (CMD)

All commands are sent using the CMD property.

Property Name: CMD

Purpose: This 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; it can exist in any object sent to the CS.

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

Formal 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
                   / iana-cmd
                   / x-cmd
                    ) CRLF
                  ;
option-value      = "OPTION" "=" paramtext
                  ;
paramtext         ; As defined in [iCAL].
```

Calendaring commands allow a CUA to directly manipulate a calendar.

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

10.1.1. 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 with an "ACTION" parameter set to the "ASK" value), then a "TIMEOUT" command MUST be sent on the same channel". The reply MUST be a an "ABORT" or a "CONTINUE" command. 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 cannot be aborted. The "ABORT", "TIMEOUT", and "REPLY" commands MUST NOT have latency set.

Upon receiving a "CONTINUE" command the work continues as if it had not been delayed or stopped. 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" parameters.

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.

If a CS can both start sending the reply to a command and guarantee that all of the results can be sent from a command (short of something like network or power failure) prior to the "LATENCY" timeout value, then the "LATENCY" time has not expired.

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;LATENCY=3;ACTION=ask:GET-CAPABILITY
   I: END:VCALENDAR
```

```
# After 3 seconds
```

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

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

```
   I: Content-Type: text/calendar
   I:
   I: BEGIN:VCALENDAR
   I: VERSION:2.0
   I: PRODID:-//someone’s prodid
   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: PRODID:-//someone’s prodid
   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
```

Here the "2.0.3" status is returned because it is not an error, it is a progress status sent in reply to the "CONTINUE" command.

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

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

If the "ACTION" value had been set to "ABORT", then the listner would send a "7.0" error on timeout in the reply to the command that initiated the command that timed out.

10.2. ABORT Command

CMD: ABORT

Purpose: The "ABORT" command is sent to request that the named or the 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
                      ;
                      / other-params
                      )
```

The REPLY of any "ABORT" command is:

```
         abort-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                         calprops
                         abort-vreply
                        "END" ":" "VCALENDAR" CRLF
                      ;
         abort-vreply = "BEGIN" ":" "VREPLY" CRLF
                         rstatus
                         other-props
                         "END" ":" "VREPLY" CRLF
```

10.3. 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 are optional,
                          ; and MAY occur more than once
                          ;
                             / other-params
                             )
```

The REPLY of any "CONTINUE" command is:

```
         continue-reply   = "BEGIN" ":" "VCALENDAR" CRLF
                            calprops
                            continue-vreply
                            "END" ":" "VCALENDAR" CRLF
                          ;
         continue-vreply  = "BEGIN" ":" "VREPLY" CRLF
                            rstatus
                            other-props
                            "END" ":" "VREPLY" CRLF
```

10.4. 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 and the hierarchy restrictions, as defined in Section 3.2:

```
         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.
                      ;
                        / action-param
                      ;
                      ; the following is optional,
                      ; and MAY occur more than once
                      ;
                        / other-params
                       )
```

Response:

One iCalendar object per TARGET property MUST be returned.

The REPLY of any "CREATE" command is limited to the restriction tables defined in [iTIP] for iTIP objects, in addition to this ABNF:

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

                     ;
      create-vreply  = "BEGIN" ":" "VREPLY" CRLF
                       created-id
                       rstatus
                       other-props
                       "END" ":" "VREPLY" CRLF
                     ;
                     ; Where the id is appropriate for the
                     ; type of object created:
                     ;
                     ; VAGENDA = relcalid
                     ; VALARM = sequence
                     ; VCAR = carid
                     ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
                     ; VQUERY = queryid
                     ; VTIMEZONE = tzid
                     ; x-comp = x-id
                     ;
      created-id    = ( relcalid / carid / uid / queryid /
                        tzid / sequence / x-id)
                      ;
      tzid          = ; As defined in [iCAL].
                      ;
      sequence      = ; As defined in [iCAL].
                      ;
      uid           = ; As defined in [iCAL].
                      ;
      x-id          = ; An ID for an x-component.
                      ;
      creply-props  = 4*(
                      ; These are REQUIRED and MUST NOT occur
                      ; more than once.
                      ;
                       prodid /version / target / reply-cmd
                      ;
                      ; These are optional, and may occur more
                      ; than once.
                      ;
                       other-props )
```

For a "CREATE" command, the "TARGET" property specifies the containers where the components will be created.

If the iCalendar object being created does not have a "METHOD" property, then its state is "BOOKED" and it is not an [iTIP] scheduling object. Use the "DELETE" command to set the state of an object to the "DELETED" state (tagged for deletion). A CUA cannot use the "CREATE" command to create an object in the "DELETED" state.

If the intention is to book an [iTIP] object, then the "METHOD" property MUST NOT be supplied. Otherwise, any [iTIP] object MUST have a valid [iTIP] "METHOD" property value and it is a scheduling request being deposited into the CS with its state set to "UNPROCESSED".

Format Definition: ABNF for a "CREATE" object is:

```
   create-object = "BEGIN" ":" "VCALENDAR" CRLF
             ; If 'calprops' contain the "METHOD" property
             ; then this 'create-object' component MUST
             ; conform to [iTIP] restrictions.
             ;
             ; calprops MUST include 'create-cmd'
             ;
                 calprops
                 other-props
                 1*(create-comp)
                 "END" ":" "VCALENDAR" CRLF

             ; NOTE: The 'VCALSTORE' component is not included in
             ; 'create-comp' as it is out of scope for CAP to create
             ; a new CS.
             ;
    create-comp =  agendac / carc / queryc
                  / timezonec / freebusyc
                  / eventc / todoc / journalc
                  / iana-comp / x-comp
                  ;
   freebusyc    = ; As defined in [iCAL].
                  ;
   eventc       = ; As defined in [iCAL].
                  ;
   journalc     = ; As defined in [iCAL].
                  ;
   timezonec    = ; As defined in [iCAL].
                  ;
   todoc        = ; As defined in [iCAL].
```

In the following example, two new top level "VAGENDA" components are created. Note that the "CSID" value of the server is cal.example.com, which is where the new "VAGENDA" components are going to be created.

```
   C: Content-Type: text/calendar
   C:
   C: BEGIN:VCALENDAR
   C: PRODID:-//someone’s prodid
   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

   S: Content-Type: text/calendar
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.0
   S: PRODID:-//someone’s prodid
   S: CMD;ID=creation01:REPLY
   S: TARGET:cal.example.com
   S: BEGIN:VREPLY                <- 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. A new "VEVENT" component is created in two TARGET components. In this example, the "VEVENT" component is one new [iTIP] "REQUEST" to be stored in two calendars. The results would be iCalendar objects that conform to the [iTIP] replies as defined in [iTIP].

This example shows two [iTIP] "VEVENT" components being created in each of the two supplied "TARGET" properties. As it contains the "METHOD" property, they will be stored in the "UNPROCESSED" state:

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: PRODID:-//someone’s prodid
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 sends the "VREPLY" commands in separate MIME objects, one per supplied "TARGET" property value.

```
S: Content-Type: text/calendar
S:
S: BEGIN:VCALENDAR
S: VERSION:2.0
S: PRODID:-//someone’s prodid
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 the second reply for the 2nd TARGET:

```
      S: Content-Type: text/calendar
      S:
      S: BEGIN:VCALENDAR
      S: VERSION:2.0
      S: PRODID:-//someone's prodid
      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
```

10.5. 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.
                  ; It MUST NOT be supplied if the latency-param
                  ; is not supplied.
                  ;
                      / action-param
                  ;
                  ; the following is optional,
                  ; and MAY occur more than once
                  ;
                      / other-params
                     )
```

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

To mark a component for delete without physically removing it, include the "OPTIONS" parameter with its value set to the "MARK" value in order to alter its state to "DELETED".

When components are deleted, only the top-most component "REQUEST-STATUS" properties are returned. No "REQUEST-STATUS" properties are 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. Note that if no "VREPLY" components are returned, then nothing matched and nothing was deleted.

Restriction Table for the "REPLY" command for any "DELETE" command.

```
   delete-reply   = "BEGIN" ":" "VCALENDAR" CRLF
                    calprops   ; MUST include ’reply-cmd’
                    *(delete-vreply)
                    "END" ":" "VCALENDAR" CRLF
                  ;
   delete-vreply  = "BEGIN" ":" "VREPLY" CRLF
                    deleted-id
                    rstatus
                    "END" ":" "VREPLY" CRLF
                  ;
                  ; Where the id is appropriate for the
                  ; type of object deleted:
                  ;
                  ; VAGENDA = relcalid
                  ; VCAR = carid
                  ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
```

```
                        ; VQUERY = queryid
                        ; ALARM = sequence
                        ; VTIMEZONE = tzid
                        ; x-comp = x-id
                        ; An instance = uid recurid
                        ;
         deleted-id    = ( relcalid / carid / uid / uid recurid
                        / queryid / tzid / sequence / x-id )
```

   Example: to delete a "VEVENT" component with "UID" value of
      "abcd12345" from the calendar "relcalid-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 VEVENT FROM VAGENDA WHERE UID = 'abcd12345'
         C: END:VQUERY
         C: END:VCALENDAR

         S: BEGIN:VCALENDAR
         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 "REQUEST-STATUS" properties for the deleted components. More than one component could have been deleted. Any booked component and any number of unprocessed [iTIP] scheduling components that matched the QUERY value in the above example will be returned. 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.

10.6. GENERATE-UID Command

   CMD: GENERATE-UID

   Purpose: The "GENERATE-UID" command returns one or more unique
      identifiers that 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.
                        ; It MUST NOT be supplied if the latency-param
                        ; is not supplied.
                        ;
                            / action-param
                        ;
                        ; The following is optional,
                        ; and MAY occur more than once.
                        ;
                            / other-params
                        ;
                        ; The following MUST be supplied exactly once.
                        ; The value specifies the number of UIDs to
                        ; be returned.
                        ;
                            / option-param posint1
                          )
```

Response:

```
     gen-reply   = "BEGIN" ":" "VCALENDAR" CRLF
     calprops              ; Which MUST include ’reply-cmd’
     1*(gen-vreply)
     "END" ":" "VCALENDAR" CRLF

     gen-vreply  = "BEGIN" ":" "VREPLY" CRLF
                   1*(uid)
                   rstatus
```

```
                       "END" ":" "VREPLY" CRLF
         {%%%IS THIS RIGHT%%%?]
```

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: PRODID:-//someone’s prodid
         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: REQUEST-STATUS:2.0
         S: END:VREPLY
         S: END:VCALENDAR
```

10.7. GET-CAPABILITY Command

CMD: GET-CAPABILITY

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

A CUA MUST send a "GET-CAPABILITY" command to a CS after the initial connection. A CS MUST send a "GET-CAPABILITY" command to a CUA after the initial connection. The "GET-CAPABILITY" command and reply MUST be implemented by all CSs and CUAs.

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

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

      capabilityparam     = *(

                    ; 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
                        ;
                             / other-params
                             )
```

Response:

The "GET-CAPABILITY" command returns information about the implementation at the other end of the session. 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 or future RFC publications. They 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.

When sending a reply to a "GET-CAPABILITY" command, all of these MUST be supplied. The following properties are returned in response to a "GET-CAPABILITY" command:

```
   cap-vreply     = "BEGIN" ":" "VCALENDAR" CRLF
                    ; The following properties may be in any order.
                    ;
                    rodid
                    version
                    reply-cmd
                    other-props

                    "BEGIN" ":" "VREPLY" CRLF
                    ; The following properties may be in any order.
                    ;
                    cap-version
                    car-level
                    components
```

```
                         stores-expanded
                         maxdate
                         mindate
                         itip-version
                         max-comp-size
                         multipart
                         query-level
                         recur-accepted
                         recur-expand
                         recur-limit
                         other-props
                        "END" ":" "VREPLY" CRLF
                        "END" ":" "VCALENDAR" CRLF
```

   Example:

```
         I: Content-Type: text/calendar
         I:
         I: BEGIN:VCALENDAR
         I: VERSION:2.0
         I: PRODID:-//someone’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: PRODID:-//someone’s prodid
         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: MAXDATE:99991231T235959Z
         L: MINDATE:00000101T000000Z
         L: MAX-COMPONENT-SIZE:0
         L: COMPONENTS:VCALENDAR,VTODO,VJOURNAL,VEVENT,VCAR,
         L: VALARM,VFREEBUSY,VTIMEZONE,STANDARD,DAYLIGHT,VREPLY
         L: ITIP-VERSION:2446
         L: RECUR-ACCEPTED:TRUE
         L: RECUR-EXPAND:TRUE
         L: RECUR-LIMIT:0
         L: STORES-EXPANDED:FALSE
         L: X-INET-PRIVATE-COMMANDS:1.0
         L: END:VREPLY
         L: END:VCALENDAR
```

10.8. 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 an "IDENTIFY" command to any CUA.

Formal Definition: An "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
                        ;
                          / other-params
                        ;
                        ; The value is the UPN of the requested
                        ; identity.  If option is not supplied it is
```

```
                       ; a request to return to the original
                       ; authenticated identity.
                       ;
                         / option-param upn
                         )
```

Response:

A "REQUEST-STATUS" property wrapped in a "VREPLY" component 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: PRODID:-//someone’s prodid
      S: BEGIN:VREPLY
      S: REQUEST-STATUS:2.0;Request Approved
      S: END:VREPLY
      S: END:VCALENDAR
```

Or if denied:

```
         S: Content-Type: text/calendar
         S:
         S: BEGIN:VCALENDAR
         S: PRODID:-//someone's prodid
         S: VERSION:2.0
         S: BEGIN:VREPLY
         S: REQUEST-STATUS:6.4;Request Denied
         S: END:VREPLY
         S: END:VCALENDAR
```

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.

10.9. 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
                       ;
                       / other-params
                       )
```

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

The format of the request is three components inside of "VCALENDAR" component:

```
   BEGIN:VCALENDAR
   BEGIN:VQUERY
   END:VQUERY
   BEGIN:XXX
   END:XXX
   BEGIN:XXX
   END:XXX
   END:VCALENDAR
```

The "VQUERY" component selects the components that are to be modified.

The "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 cannot 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" component with a "UID" property value of ’unique-58’ has the "LOCATION" property and "LAST-MODIFIED" properties changed, the "VALARM" component with the "SEQUENCE" property with a value of "3" has its "TRIGGER" property disabled, the "X-LOCAL" property is removed from the "VEVENT" component, and a "COMMENT" property is added.

Because "SEQUENCE" property is used to locate the "VALARM" component in this example, both the old-values and the new-values contain the "SEQUENCE" property with a value of "3". If the "SEQUENCE" property were to be 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: PRODID:-//someone’s prodid
      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                     <- End of old data.
      C: BEGIN:VEVENT                   <- Start 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                     <- End of new data.
         C: END:VCALENDAR
```

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

old-values:

```
      BEGIN:VCALENDAR
      VERSION:2.0
      PRODID:-//someone’s prodid
      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
```

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

Furthermore, the following additional restrictions apply:

1.   One cannot 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. All contained components that match old-values will be modified. In the first modify example above, if "SEQUENCE" properties were to be deleted from both the old-values and new-values, then all "TRIGGER" properties that matched the old-values in all "VALARM" components in the selected "VEVENT" components would be disabled.

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

Response:

A "VCALENDAR" component is returned with one ore more "REQUEST-STATUS" property values.

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" property 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 it does 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
```

10.10. 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
               ;
               / other-params
               ;
               )
```

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" component, 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" property value MUST be returned, and the "MOVE" command MUST NOT be performed.

The "TARGET" property value specifies the new location, and the "VQUERY" component specifies the old location.

Restriction Table for the "REPLY" command of any "MOVE" command.

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

   move-vreply  = "BEGIN" ":" "VREPLY" CRLF
                   move-id
                     rstatus
                  "END" ":" "VREPLY" CRLF

                 ; Where the id is appropriate for the
                 ; type of object moved:
                 ;
                 ; VAGENDA = relcalid
                 ; VCAR = carid
                 ; VEVENT, VFREEBUSY, VJOURNAL, VTODO = uid
                 ; VQUERY = queryid
                 ; ALARM = sequence
                 ; An instance = uid recurid
                 ; x-comp = x-id
                 ;
   move-id    =  ( relcalid / carid / uid / uid recurid
                  / 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: PRODID:-//someone’s prodid
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: PRODID:-//someone’s prodid
         S: TARGET:Area-51
         S: BEGIN:VREPLY
         S: CALID:Nellis
         S: REQUEST-STATUS: 2.0
         S: END:VREPLY
         S: END:VCALENDAR
```

10.11. 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
                        ;
                     / other-params
                        )
```

10.12. 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
                       ;
                       / other-params
                       )
```

The format of the request is the search command (search-cmd) followed by one or more (query) "VQUERY" components

Response:

The data in each result set contains one or more iCalendar components composed of all the selected results enclosed in a single "VREPLY" component per "QUERY".

Only "REQUEST-STATUS" property and the properties mentioned in the "SELECT" clause of the QUERY are included in the components. Each "VCALENDAR" component is tagged with the "TARGET" property.

Searching for objects

In the example below, objects on March 10,1999 between 080000Z and 190000Z are read. In this case only four properties for each object are returned. Two calendars are specified. Only booked (vs. scheduled) entries are to be returned (this example only selected VEVENT objects are to be returned):

```
C: Content-Type: text/calendar
C:
C: BEGIN:VCALENDAR
C: VERSION:2.0
C: PRODID:-//someone’s prodid
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, however one of the "VEVENT" components 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: PRODID:-//someone's prodid
         S: BEGIN:VREPLY
         S: BEGIN:VEVENT
         S: REQUEST-STATUS:4.1
         S: END:VEVENT
         S: BEGIN:VEVENT
         S: REQUEST-STATUS:2.0
         S: UID:123
         S: DTEND:19990310T080000Z
         S: DSTART:19990310T190000Z
         S: SUMMARY: Big meeting
         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 will look the same if the particular components do not exist.

```
         S: Content-Type: text/calendar
         S:
         S: BEGIN:VCALENDAR
         S: VERSION:2.0
         S: PRODID:-//someone's prodid
         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.

10.12.1. Searching for VFREEBUSY

If a CS sets the "RECUR-EXPAND" property to "TRUE" and contains the "VFREEBUSY" component in the "COMPONENTS" value in a reply to the "GET-CAPABILITY" command, then it is the CS's responsibility (and not the CUA's responsibility) to provide the correct "VFREEBUSY" information for a calendar.

If a CUA issues a "CREATE" "VFREEBUSY", such a CS MUST return success and not store the "VFREEBUSY" component as the results would never be used.

Such a CS MUST dynamically create the results of a search for "VFREEBUSY" components at search time when searching for STATE() = 'BOOKED' items.

If a CUA searches for "VFREEBUSY" components with STATE() = 'UNPROCESSED', such a CS MUST return a "VREPLY" with no components.

If a CUA searches for "VFREEBUSY" components without specifying the STATE, such a CS MUST return the same result as if STATE()='BOOKED' had been specified.

For CSs that set the "CAPABILITY" "RECUR-EXPAND" property to "FALSE" and have the "VFREEBUSY" component in the "COMPONENTS" value in the "CAPABILITY" reply, a CUA MAY store the "VFREEBUSY" information on the CS. These CSs then MUST return a "VFREEBUSY" component calculated from the stored components. If no "VFREEBUSY" information is available for the "TARGET" calendar, then a "VFREEBUSY" with no blocked out time will be returned with a success code. A CUA sets the "VFREEBUSY" time on a/those calendars by creating a "VFREEBUSY" component without a "METHOD" creating a "BOOKED" entry.

If a CS does not set the "VFREEBUSY" value in the "COMPONENTS" "CAPABILITY" value, the CS does not support the "VFREEBUSY" component and all creation and searching for a "VFREEBUSY" component MUST fail. Examples of calendars that may be in this category are public event calendars that will never require scheduling with other UPNs.

10.13. SET-LOCALE Command

CMD: SET-LOCALE

Purpose: The "SET-LOCALE" command is used to select the locale that will be used in error codes that are used in the "REQUEST-STATUS" property.

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   = setlocaleparam ":" "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
                     ; only when the latency-param has been supplied.
                     ; It MUST NOT be supplied if the latency-param
                     ; is not supplied.
                     ;
                       / action-param
                     ;
                     ; the following is optional,
                     ; and MAY occur more than once
                     ;
                       / other-params )

        setlocale-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
```

Examples:

```
        CMD:OPTIONS=en_US.UTF-8:SET-LOCALE
        CMD:OPTIONS=th_TH.ISO8859-11:SET-LOCALE
        CMD:OPTIONS=es_MX.ISO8859-1:SET-LOCALE
```

Restriction Table for the "REPLY" command of any "SET-LOCALE" command.

```
        setlocale-reply  = "BEGIN" ":" "VCALENDAR" CRLF
                            calpros
                            1*(setlocale-vreply)
                           "END" ":" "VCALENDAR" CRLF

        setlocale-vreply  = "BEGIN" ":" "VREPLY" CRLF
                             rstatus
                             "END" ":" "VREPLY" CRLF
```

10.14. 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 "TIMEOUT" command is defined by the following notation:

```
      timeout-cmd   = timeoutparam ":" "TIMEOUT"

      timeoutparam   = *(
                       ; the following are optional,
                       ; but MUST NOT occur more than once
                         id-param
                       / localize-param
                       / other-params
                       )
```

10.15. Response Codes

Numeric response codes are returned using the "REQUEST-STATUS" property.

The format of these codes is described in [iCAL] and extended 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" property 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 that 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 CS 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 to be 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.

11. Object Registration

This section provides the process for registration of new or modified properties, parameters, commands, or other modifications, additions, or deletions to objects.

11.1. Registration of New and Modified Entities

New objects are registered by the publication of an IETF Request for Comment (RFC). Changes to objects are registered by the publication of a revision to the RFC in a new RFC.

11.2. Post the Item Definition

The object description MUST be posted to the new object discussion list: ietf-calendar@imc.org.

11.3. 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.

11.4. Release a New RFC

The new object will be submitted for publication like any other Internet Draft requesting RFC status.

12. BEEP and CAP

12.1. BEEP Profile Registration

BEEP replies will be one-to-one (1:1 MSG/RPY) if possible, and one-to-many (1:many MSG/ANS) when the "TARGET" property value changes. No more than one "TARGET" property value is allowed per reply.

Profile Identification: specify a [URI] that authoritatively identifies this profile.

http://iana.org/beep/cap/1.0

Message Exchanged during Channel Creation:

CUAs SHOULD supply the BEEP "localize" attributes in the BEEP "greeting" messages.

CSs SHOULD supply the BEEP "localize" attributes in the BEEP "greeting" messages.

CUAs SHOULD supply the BEEP "serverName" attribute at channel creation time to the CS, so that, if the CS is performing virtual hosting, the CS can determine the intended virtual host. CSs that do not support virtual hosting may ignore the BEEP "serverName" attribute.

Messages starting one-to-one exchanges:

The initial message, after authentication in each direction, MUST be a single "text/calendar" object containing a CAP "CAPABILITY" CMD. It must not be part of a MIME multipart message.

After the initial message, a BEEP "MSG" may contain one or more MIME objects (at least one of which MUST be "text/calendar"), and each "text/calendar" MIME object MUST contain a CAP "CMD" property.

Multiple iCalendar objects may be sent in a single BEEP message either by representing them as separate MIME text/calendar parts contained within a MIME multipart/mixed part or by simple concatenation within a single text/calendar MIME object.

In either case, all iCalendar objects that are transmitted together must have the same TARGET property.

The sending of multipart MIME entities over BEEP is not permitted for CAP unless the other endpoint has indicated its ability to accept them via the appropriate CAPABILITY.

Messages in positive replies:

After the initial message, a BEEP "RPY" may contain one or more MIME objects (at least one of which MUST be "text/calendar"), and each "text/calendar" MIME object MUST contain a CAP "CMD" property. All "text/calendar" MIME objects in a single BEEP "RPY" messages MUST have the same "TARGET" property value.

Multiple iCalendar objects may be sent in a single BEEP message by either representing them as separate MIME text/calendar parts contained within a MIME multipart/mixed part or by simple concatenation within a single text/calendar MIME object.

In either case, all iCalendar objects transmitted together must have the same TARGET property.

Sending multipart MIME entities over BEEP is not permitted for CAP unless the other endpoint has indicated its ability to accept them via the appropriate CAPABILITY.

Messages in negative replies:

Will contain any valid "text/calendar" MIME object that contains CAP "REQUEST-STATUS" property and a CAP "CMD" property with a property value of "REPLY". And where the CS has determined the requested operation to be a fatal error. And when the CS has performed NO operation that effected the contents of any part of the CS or any calendar controlled by the CS.

Messages in one-to-many exchanges:

After the initial message then a BEEP "MSG" may contain one or more MIME objects at least one of which MUST be "text/calendar" and each "text/calendar" MIME object MUST contain a CAP "CMD" property.

The BEEP "MSG" messages can only contain MIME "multipart" MIME objects if the other endpoint has received a CAP "CAPABILITY" indicating the other endpoint supports multipart MIME objects. This does not prevent the endpoint from sending multiple [iCAL] 'icalobject' objects in a single BEEP "MSG" so long as all of them have the same "TARGET" property value.

Multiple iCalendar objects may be sent in a single BEEP message by either representing them as separate MIME text/calendar parts contained within a MIME multipart/mixed part or by simple concatenation within a single text/calendar MIME object.

In either case, all iCalendar objects transmitted together must have the same TARGET property.

The sending of multipart MIME entities over BEEP is not permitted for CAP unless the other endpoint has indicated its ability to accept them via the appropriate CAPABILITY.

Message Syntax:

They are CAP "text/calendar" MIME objects as specified in this memo.

Message Semantics:

As defined in this memo.

12.2. 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 container MAY use a one- to-many exchange with a distinct answer associated with each target. A 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.

12.3. BEEP Connection Details

All CAP communications must be done securely, so the initial greeting includes the TLS profile.

```
L: <wait for incoming connection>

I: <open connection>

L: RPY 0 0 . 0 110
L: Content-Type: application/beep+xml
L:
L: <greeting>
L:    <profile uri='http://iana.org/beep/TLS' />
L: </greeting>
L: END

I: RPY 0 0 . 0 52
I: Content-Type: application/beep+xml
I:
I: <greeting/>
I: END
```

At this point, the connection is secure. The TLS profile 'resets' the connection, so it resends the greetings, advertises the CAP profiles that are supported, and replies with the profile selected (only one profile exists at this time):

```
      L: <wait for incoming connection>

      I: <open connection>

      L: RPY 0 0 . 0 110
      L: Content-Type: application/beep+xml
      L:
      L: <greeting>
      L:    <profile uri='http://iana.org/beep/cap/1.0'/>
      L: </greeting>
      L: END

      I: RPY 0 0 . 0 110
      I: Content-Type: application/beep+xml
      I:
      I: <greeting>
      I:    <profile uri='http://iana.org/beep/cap/1.0'/>
      I: </greeting>
      I: END
```

Each channel must be authenticated before work can start, but starting a channel involves authentication. Any SASL profile may be included, for example:

```
      <profile uri='http://iana.org/beep/SASL/OTP'/>
      <profile uri='http://iana.org/beep/SASL/DIGEST-MD5'/>
      <profile uri='http://iana.org/beep/SASL/ANONYMOUS'/>
```

Example of anonymous channel:

```
      C: <start number='1'>
      C:    <profile uri='http://iana.org/beep/SASL/ANONYMOUS'/>
      C: </start>

      S: RPY 0 1 . 221 87
      S: Content-Type: application/beep+xml
      S:
      S: <profile uri='http://iana.org/beep/SASL/ANONYMOUS'/>
      S: END
```

Example of DIGEST-MD5 channel:

```
      C: <start number='1'>
      C:    <profile uri='http://iana.org/beep/SASL/DIGEST-MD5'/>
      C: </start>

      S: RPY 0 1 . 221 87
      S: Content-Type: application/beep+xml
      S:
      S: <profile uri='http://iana.org/beep/SASL/DIGEST-MD5'/>
      S: END
```

Piggybacking the "CAPABILITY" command.

The "CAPABILITY" reply may be included during channel start (see RFC3080, section 2.3.1.2), as BEEP allows the start command to include the initial data transfer. This reduces the number of round trips to initiate a CAP session.

13. 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].

The the IANA has registered the profile specified in Section 12.1, and has selected an IANA-specific URI: http://iana.org/beep/cap/1.0.

14. Security Considerations

Access rights should be granted cautiously. Without careful planning, it is possible to open up access to a greater degree than desired.

The "IDENTIFY" command should be carefully implemented. If it is done incorrectly, UPNs may gain access as other, unintended, UPNs. The "IDENTIFY" command may not chain; that is, the identity is always validated against the original UPN and not the new UPN.

Since CAP is a profile of [BEEP], consult [BEEP]'s Section 9 for a discussion of BEEP-specific security issues.

There are risks of allowing anonymous UPNs to deposit REQUEST and REFRESH objects (calendar spam and denial-of-service, for example). Implementations should consider methods to restrict anonymous requests to within acceptable usages.

CS implementations might consider automatically creating VCARs that allow CAP ATTENDEEs in booked objects to deposit REFRESH and REPLY objects for those UIDs if they otherwise do not have access rather then opening up world access. And they may also consider allowing COUNTER objects for those ATTENDEEs.

When an object is booked by a CUA ,the CS reply may wish to include warning messages to the CUA for ATTENDEEs that have CAP urls that do not have local UPNs as those ATTENDEES may be unable to REPLY or REFRESH. Some CSs may wish this to be an error.

Although service provisioning is a policy matter, at a minimum, all implementations must provide the following tuning profiles:

- o for authentication: http://iana.org/beep/SASL/DIGEST-MD5
- o for confidentiality: http://iana.org/beep/TLS (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher)
- o for both: http://iana.org/beep/TLS (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher supporting client-side certificates)

Appendix A. Acknowledgements

The following individuals were major contributors to the drafting and discussion of this memo, and they are greatly appreciated:

Alan Davies, Andrea Campi, Andre Courtemanche, Andrew Davison, Anil Srivastava, ArentJan Banck, Arnaud Quillaud, Benjamin Sonntag, Bernard Desruisseaux, Bertrand Guiheneuf, Bob Mahoney, Bob Morgan, Bruce Kahn, Chris Dudding, Chris Olds, Christopher Apple, Cortlandt Winters, Craig Johnson, Cyrus Daboo, Damon Chaplin, Dan Hickman, Dan Kohn, Dan Winship, Darryl Champagne, David C. Thewlis, David Nicol, David Nusbaum, David West, Derik Stenerson, Eric R. Plamondon, Frank Dawson, Frank Nitsch, Gary Frederick, Gary McGath, Gilles Fortin, Graham Gilmore, Greg Barnes, Greg FitzPatrick, Harald Alvestrand, Harrie Hazewinkel, Helge Hess, Jagan Garimella, Jay Parker, Jim Ray, Jim Smith, Joerg Reichelt, John Berthels, John Smith, John Stracke, Jonathan Lennox, JP Rosevear, Karen Chu, Katie Capps Parlante, Kees Cook, Ken Crawford, Ki Wong, Lars Eggert, Lata Kannan, Lawrence Greenfield, Libby Miller, Lisa Dusseault, Lyndon Nerenberg, Mark Davidson, Mark Paterson, Mark Smith, Mark Swanson, Mark Tearle, Marshall Rose, Martijn van Beers, Martin Jackson, Matthias Laabs, Max Froumentin, Micah Gorrell, Michael Fair, Mike Higginbottom, Mike Hixson, Murata Makoto, Natalia Syracuse, Nathaniel Borenstein, Ned Freed, Olivier Gutknecht, Patrice Lapierre, Patrice Scattolin, Paul Hoffman, Paul Sharpe, Payod Deshpande, Pekka Pessi, Peter Thompson, Preston Stephenson, Prometeo Sandino Roman Corral, Ralph Patterson, Robert Lusardi, Robert Ransdell, Rob Siemborski, Satyanarayana Vempati, Satya Vempati, Scott Hollenbeck, Seamus Garvey, Shannon Clark, Shriram Vishwanathan, Steve Coya, Steve Mansour, Steve Miller, Steve Vinter, Stuart Guthrie, Suchet Singh Khalsa, Ted Hardie, Tim Hare, Timo Sirainen, Vicky Oliver, Paul Hill, and Yael Shaham-Gafni.

Appendix B. References

Appendix B.1. Normative References

[ABNF] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[BEEP] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[BEEPTCP] Rose, M., "Mapping the BEEP Core onto TCP", RFC 3081, March 2001.

[BEEPGUIDE] Rose, M., "BEEP, The Definitive Guide", ISBN 0-596-00244-0, O’Reilly & Associates, Inc.

[GUIDE] Mahoney, B., Babics, G., and A. Taler, "Guide to Internet Calendaring", RFC 3283, June 2002.

[iCAL] Dawson, F. and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998.

[iTIP] Silverberg, S., Mansour, S., Dawson, F., and R. Hopson, "iCalendar Transport-Independent Interoperability Protocol (iTIP) Scheduling Events, BusyTime, To-dos and Journal Entries", RFC 2446, November 1998.

[iMIP] Dawson, F., Mansour, S., and S. Silverberg, "iCalendar Message-Based Interoperability Protocol (iMIP)", RFC 2447, November 1998.

[MIME] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

Appendix B.2. Informative References

[CHARREG] Freed, N. and J. Postel, "IANA Charset Registration Procedures", RFC 2278, January 1998.

[CHARPOL] Alvestrand, H., "IETF Policy on Character Sets and Languages", RFC 2277, January 1998.

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, April 2001.

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

[SQL92] "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

[URLGUIDE] Masinter, L., Alvestrand, H., Zigmond, D., and R. Petke, "Guidelines for new URL Schemes", RFC 2718, November 1999.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 3986, January 2005.

[URL] Berners-Lee, T, Masinter, L., and M. McCahil, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[X509CRL] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

Authors' Addresses

Doug Royer
IntelliCal, LLC
267 Kentlands Blvd. #3041
Gaithersburg, MD 20878
US

Phone: +1-866-594-8574
Fax: +1-866-594-8574
EMail: Doug@IntelliCal.com
URI: http://Royer.com

George Babics
Oracle
600 Blvd. de Maisonneuve West
Suite 1900
Montreal, Quebec H3A 3J2
CA

Phone: +1-514-905-8694
EMail: george.babics@oracle.com

Steve Mansour
eBay
2145 Hamilton Avenue
San Jose, CA 95125
USA

Phone: +1-408-376-8817
EMail: smansour@ebay.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.