Network Working Group
Internet Draft
<draft-ietf-calsch-cap-00.txt>

Steve Mansour/Netscape
Frank Dawson/Lotus
Doug Royer/Sun Microsystems
Alexander Taler/CS&T
Paul Hill/MIT

Expires six months from: August 5, 1999

Calendar Access Protocol (CAP)

This memo is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Distribution of this document is unlimited.

Abstract

The Calendar Access Protocol (CAP) is an Internet protocol that permits a Calendar User (CU) to utilize a Calendar User Agent (CUA) to access an [RFC2445] based Calendar Store (CS). This memo defines the CAP specification.The CAP definition is based on requirements identified by the Internet Engineering Task Force (IETF) Calendaring and Scheduling (CALSCH) Working Group. More information about the IETF CALSCH Working Group activities can be found on the IMC web site at http://www.imc.org/ietf-calendar, and at the IETF web site at http://www.ietf.org/html.charters/calsch-charter.html. Refer to the references within this memo for further information on how to access these various documents.

Table of Contents

1. Introduction

This document specifies how a Calendar User Agent (CUA) interacts with a Calendar Store (CS) to manage calendar information. In particular, it specifies how to query, create, modify, and delete iCalendar components (e.g., events, to-dos, or daily journal entries). It further specifies how to search for available busy time information.

This protocol is based on request/response form of protocol data units, sent from a client CUA to a calendar server. The protocol data units leverage the standard iCalendar format [RFC2445] for conveying CS related information.

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 this memo. Calendar components defined by [RFC2445] are referred to with capitalized, quoted-strings of text. All calendar components start with the letter "V". For example, "VEVENT" refers to the event calendar component, "VTODO" refers to the to-do calendar component and "VJOURNAL" refers to the daily journal calendar component. Calendar access methods defined by this memo, as well as scheduling methods defined by [RFC2446], are referred to with capitalized, quoted-strings of text. For example, "CREATE" refers to the method for creating a calendar component on a calendar, "READ" refers to the method for reading calendar components.

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 of a "Calendar User". Property parameters defined by this memo are referred to with lower case, quoted-strings of text, followed by the word "parameter". For example, "value" parameter refers to the iCalendar property parameter used to override the default data type for a property value. Enumerated values defined by this memo are referred to with capitalized text, either alone or followed by the word "value".

In tables, the quoted-string text is specified without quotes in order

to minimize the table length.

1.2 Related Documents

Implementers will need to be familiar with several other memos that, along with this one, describe the Internet calendaring and scheduling standards. This document,

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

[RFC2446] specifies an interoperability protocol for scheduling between different implementations. The related documents are:

[RFC2447] specifies an Internet email binding for [RFC2446].

[iRIP] specifies a real-time binding for [to be published].

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

1.3 Definitions

Authentication ID (AuthID)

A tuple of username, realm, and authentication method, used by the Calendar Service internally to identify a successfully authenticated CAP session.

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 other calendar properties or calendar components. In addition, a calendar might be hierarchically related to other sub-calendars. A calendar is identified by its unique calendar identifier. The [RFC2445] defines calendar properties, calendar components and component properties that make up the content of a calendar.

Calendar Access Protocol (CAP)

The standard Internet protocol that permits a Calendar User Agent to access and manipulate a calendar residing on a Calendar Store.

Calendar Access Rights (CAR)
: The mechanism for specifying the CAP operations ("ACTIONS") that a particular calendar user ("UPN") are granted or denied permission to perform on a given calendar entity ("OBJECT"). The calendar access rights are specified with the "VCAR" calendar components within a CS and calendar.

Calendar Component
: An entity within a calendar. Some types of calendar components include events, to-dos, journals, alarms, time zones and freebusy data. A calendar component consists of component properties and possibly other sub-components. For example, an event may contain an alarm component.

Calendar Component Properties
: An attribute of a particular calendar component. Some calendar component properties are applicable to different types of calendar components. For example, DTSTART is applicable to VEVENT, VTODO,

VJOURNAL calendar components. Other calendar components are applicable only to an individual type of calendar component. For example, TZURL is only applicable to VTIMEZONE calendar 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.

Calendar Policy
: A CAP operational restriction on the access or manipulation of a calendar. For example, "events MUST be scheduled in unit intervals of one hour".

Calendar Properties
: An attribute of a calendar. The attribute applies to the calendar, as a whole. For example, CALSCALE specifies the calendar scale (e.g., GREGORIAN) for the whole calendar.

Calendar Service
: An implementation of a Calendar Store that manages one or more calendars.

Calendar Store (CS)
The data and service model definition for a Calendar Service.

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 [RFC2396].

Calendar Store Components
Components maintained in a CS specify a grouping of calendar store-wide information. Calendar store components can be accessed using CAP.

Calendar Store Properties
Properties maintained in a Calendar Store calendar store-wide information. Calendar store properties can be accessed using CAP.

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

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

Calendaring and Scheduling System
The computer sub-system that provides the services for accessing, manipulating calendars and scheduling calendar components.

CAP Session
An open communication channel between a CAP CUA and a CAP CS.

Connected Mode

A mobile computing mode where the CUA is directly connected to the CS.

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

Designate

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

Disconnected Mode

A mobile computing mode where a CUA can be disconnected from a CS. When the CUA is disconnected, it is in the disconnected mode.

Fan Out

The calendaring and scheduling process by which a calendar operation on one calendar is also performed on every other calendar specified in the operation. This may include the calendar associated with TARGET calendar property.

Hierarchical Calendars

A CS feature where a calendar have a hierarchical relationship with another calendar in the CS. The top-most calendar in the hierarchical relationship has the CS as its parent. There may be multiple top-most calendars in a given CS. Within a given hierarchical relationship, all sub-calendars have a calendar with a "parent" topographical relationship. In addition, sub-calendars may have a relationship with another calendar that has a "child" topographical relationship. In addition, a calendar may have a relationship such that one or more calendars have a "sibling" topographical relationship with the calendar. The hierarchical calendar feature is not a storage relationship of the calendars within the CS. Instead it is a feature that relates access control rights to calendar content between different calendars in the CS. The hierarchical relationship of a calendar is specified in the "PARENT" and "CHILDREN" calendar properties.

High Bandwidth Connection

A communications connection supporting high transfer rates; transfer rates commonly found within a LAN.

Local Store

A CS which is on the same platform as the CUA.

Low Bandwidth Connection

A communications connection supporting slow transfer rates; transfer rates commonly found in remote access technology.

Owner

A CU or CUs that have "OWNER" calendar access rights for a calendar. The owner is specified in the "OWNER" calendar property.

Qualified Calendar Identifier (Qualified CalID)
A CalID where both the <scheme> and <csid> 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. (eg: 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 is unique within a calendar store. It is recommended to be globally unique. A Relative CalID consists of the portion of the "scheme part" of a Qualified CalID following the Calendar Store Identifier. This is the same as the "URL path" of the "Common Internet Scheme Syntax" portion of a URL, as defined by [RFC2396].

Remote Store
A CS which is not on the same platform as the CUA.

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

Sub-calendars
Calendars that have a "child" hierarchical relationship with another calendar, its "parent".

User Name
A name which denotes a Calendar User within a realm. This is part of a UPN.

User Principal Name (UPN)
An identifier that denotes a unique CU. A UPN strongly resembles an RFC 822 style email address and in some cases it may be identical to the email address for the CU. It consists of a realm in the form of a DNS domain name and a username. It may also have an optional instance. In it's simplest form it looks like "user@example.com".

2. CAP Design

2.1 System Model

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

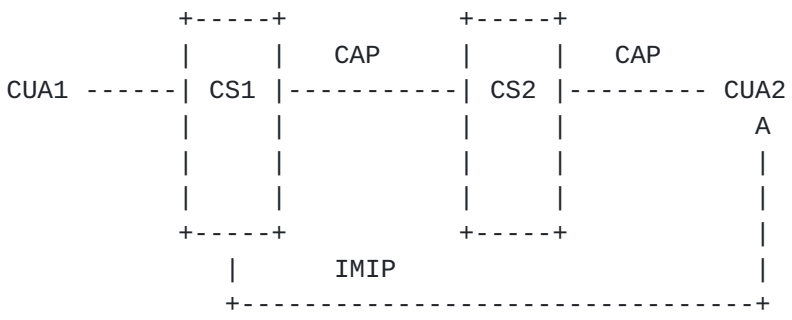
CAP is used by a "Calendar User Agent" (CUA) to send commands to and receive responses from a "Calendar Service" (CS). The CUA prepares an

MIME encapsulated iCalendar object containing a command, sends it to the CS, and receives an iCalendar object as a response. There are two distinct protocols in operation to accomplish this exchange. The Transport Protocol is used to move iCalendar objects between a CUA and a CS. The Application Protocol defines the content and semantics of the iCalendar objects sent between the CUA and the CS. This document defines both the Transport Protocol and the Application Protocol.

In the diagram below, a user uses CUA1 to communicate with CS1 using CAP. The CUA must authenticate the Calendar User (CU) so that access to calendars on CS1 can be controlled. The CUA can then view, create, edit, and delete calendars, calendar properties, and calendar components subject to the access rights.

CAP servers support fanout. Fanout allows a CUA to communicate with a single CS to perform scheduling operations with calendars on multiple CSs. That is, a Calendar User (CU) can book events on or read events from calendars on other calendar stores. To accomplish this, a CAP server has several options:

CS1 MAY play the role of a CUA and use CAP to access CS2;
CS1 MAY be able to play the role of a CUA and use [iRIP] to interoperate with the possible iRIP support in CS2;
CS1 MUST be able play the role of a CUA and use [RFC2447] to interoperate with other CUAs.
Storage Agent

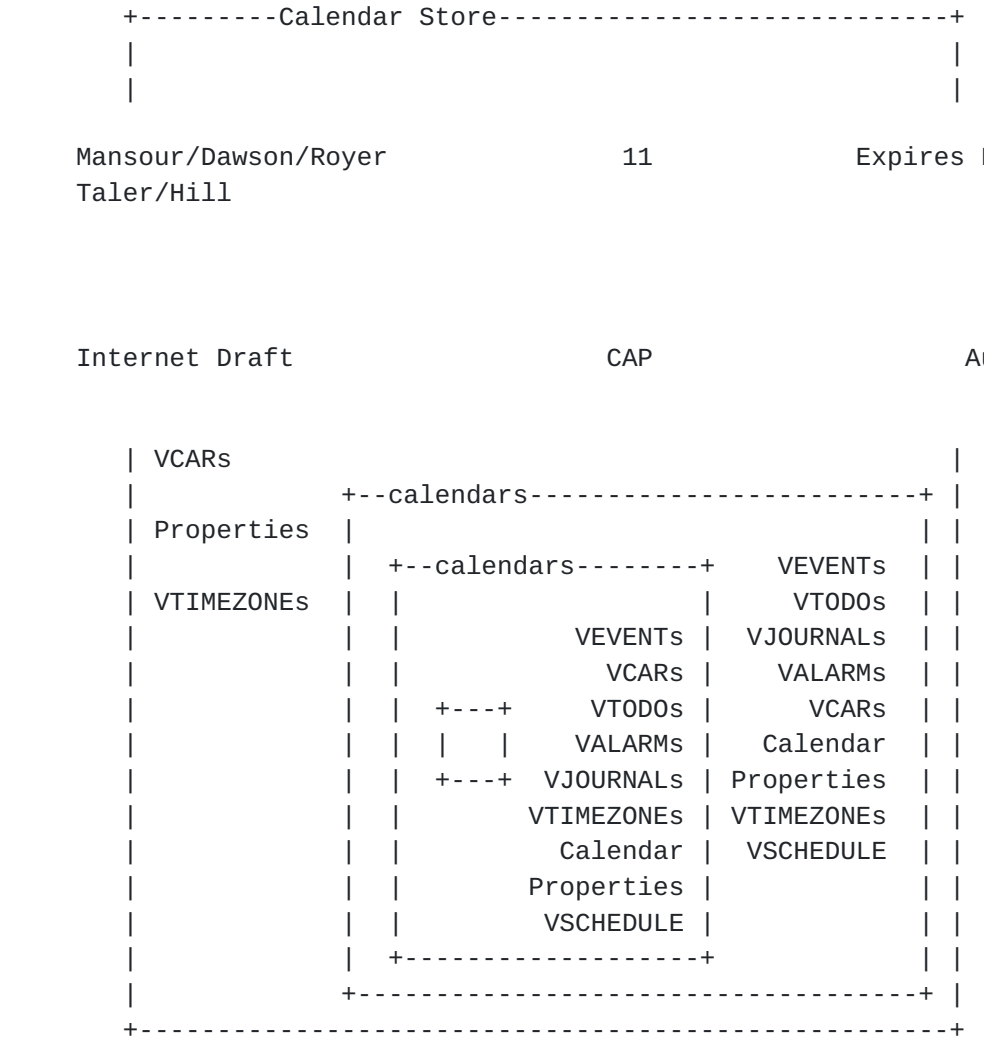


Note that the fanout feature in CAP is a convenience to the CUA. It is perfectly valid for the CUA to assume the responsibility for fanout if it wishes. That is, [RFC2447] messages could also be sent from CUA1 to CUA2.

2.2 Calendar Store Object Model

The conceptual model for a calendar store is shown below. The calendar store contains calendars, VTIMEZONEs, VCARs, and calendar store properties.

Calendars contain VEVENTs, VTODOs, VJOURNALs, VALARMs, VCARs, and calendar properties. Calendars may also contain other calendars.



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

In this model, VSCHEDULE is a queue of scheduling messages that have not yet been applied to the calendar. Items in VSCHEDULE are discussed in more detail below.

2.3 Protocol Model

A generic transport, Calendar Server Transport Protocol (CSTP), is used to move data objects between a CUA and the CS. CSTP commands are listed below and their usage and semantics are defined in section 7 of this document.

CSTP Commands

Command	Description
ABORT	Stop a command whose latency time has been exceeded.
AUTHENTICATE	Authenticate a UPN.
CONTINUE	Continue the execution of a command whose latency time has been exceeded.
IDENTIFY	Set a new identity for calendar access.
DISCONNECT	Terminate a connection with the server.
SENDDATA	Send a data object MIME encapsulated iCalendar.
STARTTLS	Negotiate transport level security using [TLS]

Application-level commands are used to manipulate data on the calendar store. They are listed below and discussed in detail in section 7.

CAP Calendaring Commands

Command	Description
CREATE	Create a new calendar or component

DELETE	Delete a calendar or component
GENERATEUID	Generate one or more unique ids
MODIFY	Change a calendar or component
MOVE	Move a calendar
READ	Read a calendar properties or components

CAP Scheduling Commands

Command	Description
PUBLISH	publish a calendar entry to one or more calendars
REQUEST	schedule a calendar entry with one or more calendars
REPLY	response to a scheduling request

ADD	add one or more instances to an existing calendar entry
CANCEL	cancel one or more instances to an existing calendar entry
REFRESH	a request for the latest version of a calendar entry
COUNTER	a request for a change (a counter-proposal) to a calendar entry
DECLINECOUNTER	decline a counter proposal

2.4 Roles

CAP defines methods for managing [RFC2445] objects on a Calendar Store and exchanging [RFC2445] objects for the purposes of group calendaring and scheduling between "Calendar Users" (CUs). There are two distinct roles taken on by CUs in CAP. The CU who creates an initial event or to-do and invites other CUs as attendees takes on the role of "Organizer". The CUs asked to participate in the group event or to-do take on the role of "Attendee". Note that "role" is also a descriptive parameter to the "ATTENDEE" property. Its use is to convey descriptive context to an "Attendee" such as "chair", "REQ-PARTICIPANT" or NON-PARTICIPANT" and has nothing to do with the scheduling workflow.

2.5 Calendar User

A Calendar User (CU) is an entity that can be authenticated. It is represented in CAP as a UPN. A UPN is the owner of a calendar and the subject of access rights.

Examples:
   user@example.com
   user/cap@example.com

The UPN representation is independent of the authentication mechanism used during a particular CUA / CS interaction. A CU may use one mechanism while using one CUA but the same user may use a different authentication mechanism when using a different CUA, or while connecting from a different location.

For Calendaring and Scheduling systems that are integrated with a directory system the UPN SHOULD be stored in the attribute [TBD] with OID [TBD]. This enables a clear mapping between a UPN and a Distinguished Name. [note: Microsoft's Active Directory is storing UPNs

as the userPrincipalName.] Within a directory service a UPN is a single

valued property.

2.5.1 UPNs and Certificates

When using certificates for purposes of CAP authentication, the SubjectName field of the user's certificate SHOULD contain the user's UPN (for example, "juser@example.com") as the value of the "CN=" component, and the user's email address (often the same as the UPN) as the value of the "E=" component . The altSubjectName will contain the DN of the user's account object in the DS. The Issuer field must be that of a root CA trusted to issue login certificates, or the DN of a lower level CA whose certificate includes an "AuthorizedNamingContext" field that authorizes it to issue certificates for "example.com" (exact field name and validation mechanism TBD).

Note: If a server is validating data received via iMIP, if the "ORGANIZER" or "ATTENDEE" property said (e.g.) "ATTENDEE;CN=Joe Random User:juser@example.com" then the "juser@example.com" part should be checked against the altSubjectName field of the certificate, and the "Joe Random User" part should be checked against the CN component of the altSubjectName DN. This is so the "ATTENDEE" property couldn't be munged to something misleading like "ATTENDEE;CN=Joe Rictus User:juser@example.com" and have it pass validation. This validation will also defeat other attempts at confusion.

2.5.2 CAP session identity

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

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

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

Since the UPN includes realm information it may be used to govern calendar store access rights across realms. However, governing access

rights across realms is only useful if login access is available. This

could be done through a trusted server relationship or a temporary account.

The "IDENTIFY" command 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.

2.6 Calendar Addresses

Calendar addresses are URIs that are modeled after [RFC2396]. CAP uses the following forms of URI.

```
[[<scheme>]://<csid>[:<port>]/]<relativeCALID>
```

where:

<scheme> is "cap"
<csid> is the Calendar Store ID. It is the network address of the computer on which the CAP server is running
<port> is optional. Its default value is 5229. The port must be present if the CAP server does not listen on the default port.
<relativeCALID> is an identifier that uniquely identifies the calendar on a particular calendar store. There is no implied structure in a Relative CALID. It is an arbitrary string of 7 bit ASCII characters. 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. It is recommended that the Relative CALID be globally unique.

If the <scheme> and <csid> are present the calendar address is said to be "qualified". Senders are required to supply the <relativeCALID> portion of the address. A qualified calendar address is required when the <csid> of the target calendar address differs from that of the CAP server receiving the command.

Examples:

```
cap://calendar.example.com/user1
://calendar.example.com/user1
```

user1
cap://calendar.example.com/conferenceRoomA
cap://calendar.example.com/89798-098-zytytasd

For a user currently authenticated to a CAP server on calendar.example.com, the first three addresses refer to the same calendar.

2.7 Finding CAP Servers

Using DNS
Using SLP
Request-Status _ optional text (second field)

2.8 Extensions to iCalendar

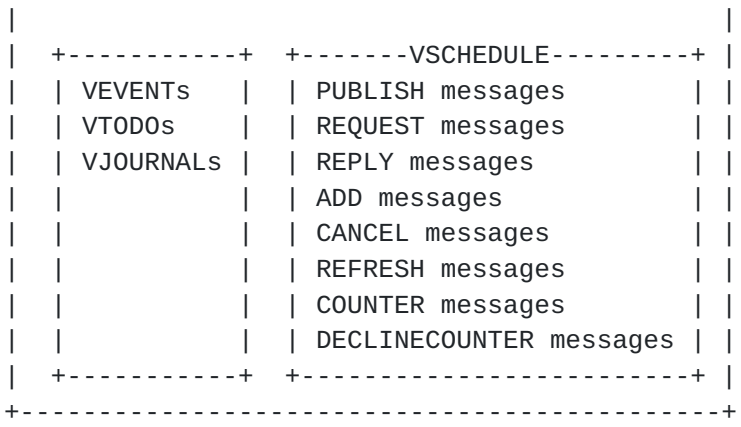
In mapping the CAP command set, query feature, and access rights onto the iCalendar format, several extended iCalendar methods and components are defined by this memo.

The search function is specified with the new iCalendar QUERY method. The QUERY method makes use of a new component, called VQUERY, that contains the search filter. The component consists of a set of new properties: SCOPE, MAXRESULTS, MAXRESULTSSIZE, QUERY and QUERYNAME, that define the search filter.
Access control is specified the the new iCalendar VCAR component.
The iCalendar METHOD property format has been updated with new values.
A new iCalendar component, VCOMMAND, has been added. VCOMMANDs are needed to fully specify CAP commands.
TARGET is a new property within the VCOMMAND component. It indicates a

2.9 Relationship of RFC 2446 (ITIP) to CAP

[RFC2446] describes scheduling methods which result in indirect manipulation of calendar components. CAP methods provide direct manipuation of calendar components. In the CAP calendar store model, scheduling messages are kept separate from other calendar components. This is modeled with the VSCHEDULE queue. Note that this is a conceptual model, the actual storage details are left to implementations. The model is shown pictorially as follows:

+-----------------VCALENDAR-------------------+



The METHOD is saved along with components. Scheduled components become booked components when the METHOD changes from an ITIP method to the CAP storage method. For example, a component whose METHOD is "REQUEST" is scheduled. The component becomes booked when the METHOD is changed to "CREATED".

[ed note: need to clean up the terminology here. We haven't discussed "booked"]

2.10 VCalendar Access Rights (VCARs)

In simple terms, VCARs are used to grant or deny access to a calendar for a Calendar User. Specifically, they grant User Principal Names

(UPNs) the rights to read and write components, properties, and parameters on calendars within a calendar store.

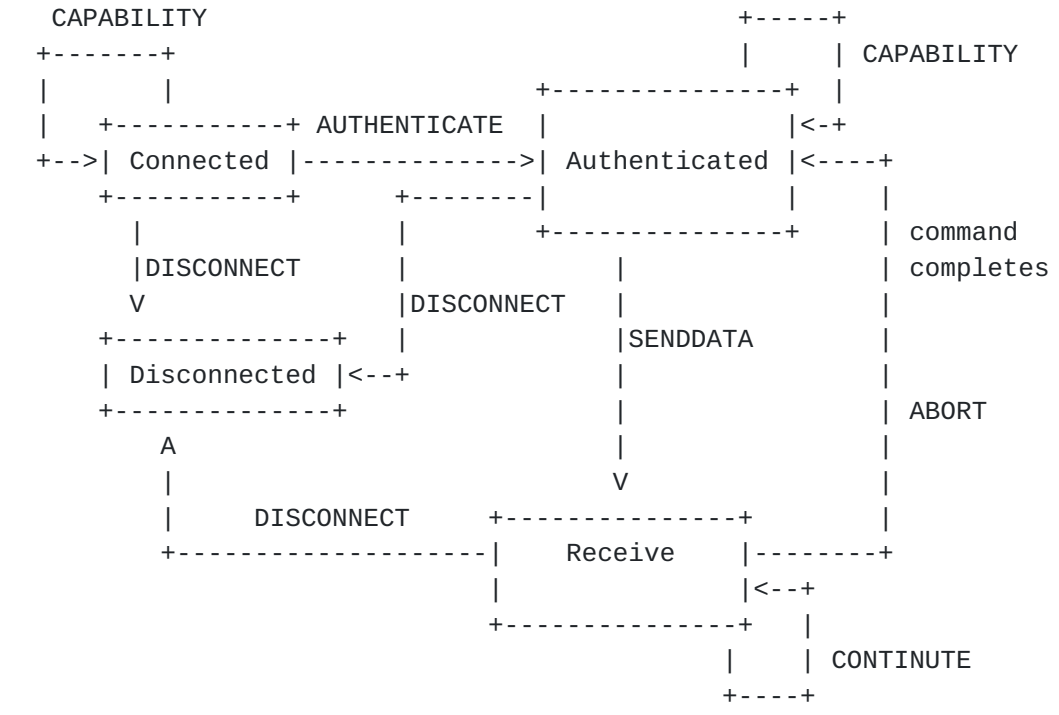
The model does not put any restriction on the sequence in which the object and access rights are created. That is, an event associated with a particular VCAR might be created before or after the actual VCAR is defined. In addition, the VCAR and VEVENT definition might be created in the same iCalendar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

2.11 Query Schema

3. State Diagram

This section describes the states of the transport connection between a CUA and a CS. The state diagram is shown below. State names shown with first letter capitalized. The commands used to switch between states are

shown next to an arrow connecting the states. The commands are listed in all capital letters. A condition that causes a state to change is shown in lower case letters.



The connection begins the Connected state when a CUA connects to a CAP server. The capabilities of the CS are reported in the response from the CS. From this state, the CUA can issue the DISCONNECT command to terminate the connection, the CAPABILITY command, or the AUTHENTICATE command to authenticate a Calendar User. The capabilities of the CS in the authenticated state are returned in the response from the CS. One use of the CAPABILITY command at this stage is to determine the supported authentication mechanisms supported by the server.

If an AUTHENTICATE command is successful, the connection enters the Authenticated state. From here the CUA can issue the CAPABILITY command.

The capabilities the server offers in the Authenticated state may be different than those in the Connected state. The connection remains in the Authenticated state after the CAPABILITY command completes. The CUA can issue the DISCONNECT command to terminate the connection. The SENDDATA command can be used to send a request to read, write, modify,

or delete data on the server.

After the SENDDATA command has been issued the connection enters the Receive state while the CUA awaits and reads a server reply. Normally, the server handles the command, sends a reply which is read by the CUA and the connection returns to the Authenticated state. The CUA may have issued the SENDATA command with a maximum latency time. This informs the server that the CUA expects a response within the maximum latency time, even if the command was not completed. When the server is unable to complete the command in the maximum latency time, it issues an appropriate reply code and waits for the CUA to tell it how to proceed. If the CUA issues a CONTINUE command the server continues processing the command and the connection remains in the Receive state. If the CUA issues the ABORT command the server need not process the command any further and the connection returns to the Authenticated state. The DISCONNECT command can also be issued from the Receive state.

4. Protocol Framework

4.1 CAP Application Layer

The CAP application layer is used for the manipulation of the calendar store. Commands and responses are transmitted between the CUA and CS inside "VCALENDAR" component wrappers. Commands are specified as the value of a "METHOD" property, and responses are specified as the value of a "REQUEST-STATUS" property.

4.2 CAP Transport Layer

The CAP transport layer handles the transmission of CAP application layer messages.

CAP transport layer commands are transmitted across the underlying transport. The transport used is a TCP/IP socket connection between the CUA and the CS. The CS listens for connections on port <xyz>.

Messages sent between the CUA and CS are formatted as a command followed by any associated data:

<command> [<command data>]

4.3 Response Format

Server responses consist of a response code and any parameters:

<response code> [; debug text ; more text]
[<CRLF><application-data>]<CRLF>.CRLF>

The response codes are defined in Section 8. The debug text is human-readable information for protocol debugging.

The optional application-data begins on the next line.

The response is terminated with a <CRLF> "." <CRLF> sequence. Any <CRLF> "." sequences which appear in the transmitted data must be quoted by placing an additional "." between the <CRLF> and the ".". For example, the following sequences of characters in the application data:

```
.
..2
...3
```

are quoted as follows:

```
..
...2
....3
```

No other tagged command sequence can be sent until the special terminating character sequence <CRLF>.<CRLF> has been sent.

4.4 Auto-logout Timer

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

When a timeout occurs, the server drops the connection to the CUA.

4.5 Bounded Latency

[CAP] is designed so that the CUA can either obtain an immediate response from a request or discover within a specified amount of time that the request could not be completed in the requested amount of time. When the CUA initiates a command that the server cannot complete within the specified latency time, the server returns an appropriate response code. The CUA then issues either a CONTINUE or ABORT command. The ABORT command immediately terminates the command in progress and the connection returns to the Authenticated state. The CONTINUE command instructs the server to continue processing the command.

4.6 Data Elements

The data elements for CAP are MIME encapsulated iCalendar objects.

5. Formal Command Syntax

5.1 Searching and Filtering

This section describes CAPs searching and filtering entities within a remote store. It is based on the Standard Query Language (SQL) defined by [SQL].

The QUERY property value MUST be a valid QUERY value type. This new value type is defined to be a "name=value" value type grammar, similar in syntax to the format already in use for the iCalendar RECUR value type. Each "name" is the name of a valid SQL statement component (e.g., SELECT, WHERE, etc.). Each "value" is valid string associated with one of these SQL statement components.

[Editor's note: We need to precisely define what part of SQL we're using and why we chose what we did.]

Examples needed:
Grant someone access to June events
Grant someone access to events during the month of June. (i.e., based on the current system date, if it's prior to June or after June you don't have access)

Example for denying access to a specific property:

DENY:UPN=FOO;ACTION=*;OBJECT=CLASS

*scope vcar to a component
*scope Grant, Deny of a VCAR

5.1.1 Grammar For Search Mechanism

```
SEARCH  = "BEGIN:VQUERY" CRLF
          [scope] [maxresults] [maxsize] querycomp
```

```
          "END:VQUERY" CRLF

scope           = "SCOPE:" comp-name ("," comp-name)*

comp-name       = "VEVENT" / "VTODO" / "VJOURNAL" / "VTIMEZONE"
                / "VALARM" / "VFREEBUSY" / iana-name / x-name

maxresults      = integer

maxsize         = integer

querycomp       = (query) / (queryname query) / queryname

queryname       = "QUERYNAME:" text

query           = "QUERY:" queryrule

queryrule       = select where orderby ...

select          = <any valid SQL string that goes into a SELECT clause>
```



```
where           = <any valid SQL string that goes into a WHERE clause>

orderby         = <any valid SQL string that goes into a ORDERBY
                  clause>
```

6. Access Rights

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

Individual calendar access rights MUST be specifically granted to an authenticated calendar user (i.e., UPN); all rights are denied unless specifically granted.

Properties within an iCalendar object are unordered. This also is the case for the "GRANT", "DENY" and "CARID" properties. Likewise, there is no implied ordering required for components of a "RIGHTS" value type other than that specified by the ABNF.

6.1 VCAR Inheritance

Calendar access rights specified in a calendar store are inherited as default calendar access rights for any calendar in the parent calendar store. Likewise, any calendar access rights specified in a root calendar are inherited as default calendar access rights for any sub-calendar to the root calendar. By implication, calendar access rights specified in a sub-calendar are inherited as default calendar access rights for any calendars that are hierarchically below the sub-calendar.

Calendar access rights specified in a calendar override any default calendar access rights. Calendar access rights specified within a sub-calendar override any default calendar access rights.

7. Commands and Responses

CAP commands and responses are described in this section. Command arguments, identified by "Arguments:" in the command descriptions below, are described by function, not by syntax. The precise syntax of command arguments is described in the Formal Syntax section.

Some commands cause specific server data to be returned; these are identified by "Data:" in the command descriptions below. See the response descriptions in the Responses section for information on these responses, and the Formal Syntax section for the precise syntax of these responses.

The "Result:" in the command description refers to the possible status responses to a command, and any special interpretation of these status responses.

Commands have the general form:

```
<command> [arguments...]
```

where <command> is a command listed in the table above. A command MAY have arguments. Arguments are defined in the detailed command definitions below.

Responses to commands have the following general form:

```
responseCode [sep transportDescr sep [applicationDescr]]
```

CRLF "." CRLF

In the examples below, lines preceded with "S:" refer to the sender and lines preceded with "R:" refer to the receiver. Lines in which the first non-whitespace character is a "#" are editorial comments and are not part of the protocol.

7.1 Transport Protocol Commands

7.1.1 Initial Connection

Arguments: none
Data: noneResult: 2.0 _ success.
8.1 _ server too busy

Upon session startup, the server sends a response of 2.0 to indicate that it is ready to receive commands. A response of 8.1 indicates that the server is too busy to accept the connection. In addition, the general capabilities of the CS are reported in the response from the CS. These capabilities may be different than those reported in the authenticated state.

The supported AUTHentication mechanisms. There may be 1 or more.
CAPVERSION
IRIPVERSION

7.1.2 ABORT Command

Arguments: none

Data: none

Result: 2.0 _ success
2.2 _ no command is in progres

The ABORT command is issued by the CUA to stop a command whoselatency time has been exceeded. When the latency time is specified onthe SENDATA command, the CS must issue a reply to the CUA
within the specified time. It may be a reply code indicating
that the CS has not yet processed the request. The CUA must
then tell the server whether to continue or abort.

The CUA can issue the ABORT command at any time after the SENDATA command has been completed but before receiving a reply.

7.1.3 AUTHENTICATE Command

Arguments: <SASL mechanism name> [<initial data>]

Data: continuation data may be requested

Result: 2.0 - Authenticate completed, now in authenticated state
6.0 - Failed authentication
6.1 - Authorization identity refused.
6.2 - Sender aborted authentication, authentication exchange cancelled
6.3 - Unsupported Authentication Mechanism
9.1 - Unexpected command.

The capabilities of the CS in the authenticated state are reported in the response from the CS. These may be different than the capabilities in the Connected, but unauthenticated state.

The AUTHENTICATE command is used by the CUA to identify the user to the CS. CAP uses the [SASL] specification for authentication. The desired SASL mechanism is specified as the initial argument.

<SASL mechanism name> is a registered SASL authentication mechanism. (Refer to [SASL] for information on obtaining a list of currently registered mechanisms.) CS Supported authentication mechanisms can be discovered using the CAPABILITY command. All implementations MUST support Digest-MD5 authentication using DES and 3DES, as well as DES-56 for link level encryption. Implementations MUST support the SASL Anonymous mechanism, although this may be disabled in installations. Implementations SHOULD implement the External SASL mechanism and the command STARTTLS.

<initial data> is an optional parameter which can be used for mechanisms which require an initial response from the CUA.

The AUTHENTICATE command is followed by an authentication protocol exchange, in the form of a series of CS challenges and CUA responses. These challenges and responses are encoded in Base64 and transmitted with a terminating CRLF. The CS terminates the exchange with a "." <CRLF> sequence followed by a reply code. ("." is not a legal Base64 character.) Possible reply codes are listed above.

CAP does not provide support for SASL authorization identities. If a CUA attempts to use an authorization identity the Calendar Service must return the reply code indicating that the authorization identity was refused.

If the CUA wishes to cancel an authentication exchange it may do so by issuing a "." <CRLF> sequence. Upon receipt of such a sequence the CS MUST terminate the exchange and return the appropriate reply code.

If a security layer was negotiated it comes into effect for the CS starting with the first octet transmitted after the CRLF which follows the 2.0 reply code, and for the CUA starting with the first octet after the CRLF of its last response in the authentication exchange. Encrypted data is transmitted as described in [SASL].

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

The result of the AUTHENTICATE command includes data indicating the identity which has been assigned to the session, derived from the supplied authentication credentials.

A CAP session does not have an identity until the CUA has issued the "AUTHENTCATE" command.

The CUA may not issue the "AUTHENTCATE" command multiple times, even if the first attempt was aborted. If a CUA attempts to do this the CS must terminate the session.

Data returned in response to a successful logon is:

Client implementations SHOULD NOT require any capability name beyond those defined in this specification, and MAY ignore any non-standard, experimental capability names. Non-standard capability names are prefixed with the text "X-". The prefix SHOULD also include a short character vendor identifier For example, "X-FOO-BARCAPABILITY", for the non-standard "BARCAPABILITY" capability of the implementor "FOO". This command may return different results in the Connected state versus the Authenticated state. It may also return different results depending on the UPN.

Capability	Occurs	Description
CAPrev1	1	Revision of CAP, must be

"CAPrev1"

IRIPrev1 0 or 1 Revision of IRIP, MAY be present. If present, it MUST be "IRIPrev1"

CAR 0 or 1 Indicates level of CAR support CAR0, CAR1, CAR2, CAR3

MAXICALOBJECTSIZE 0 or 1 An integer value that specifies The largest ICAL object the server will accept. Objects larger than this will be rejected.

MAXDATE 0 or 1 The datetime value beyond which

the server cannot accept.

MINDATE 0 or 1 The datetime value prior to which the server cannot accept.

The following examples illustrate the various possiblities for an authentication protocol exchange.

Here are examples of a successful authentication:

```
C: AUTHENTICATE KERBEROS_V4
S: AmFYig==
C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
S: or//EoAADZI=
C: DiAF5A4gA+oOIALuBkAAmw==
S: 2.0
S: Content-Type:text/calendar; method=REQUEST; charset=US-ASCII
S: Content-Transfer-Encoding: 7bit
S:
S: BEGIN:VCALENDAR
S: PRODID:-//ACME/CAPserver//EN
S: VERSION:2.1
S: IDENTITY=bill@example.com
S: CAPVERSION=1.0
S: ITIPVERSION=1.0
S: AUTH=KERBEROS_V4
S: AUTH=DIGEST_MD5
```

```
   S: CAR=CAR1  appl
   S: MINDATE=19700101T000000Z  appl
   S: MAXDATE=20370201T000000Z
   S: END:VCALENDAR
   S: .


   C: AUTHENTICATE ANONYMOUS
   S: 2.0
   S: Content-Type:text/calendar; method=REQUEST; charset=US-ASCII
   S: Content-Transfer-Encoding: 7bit
   S:
   S: BEGIN:VCALENDAR
   S: PRODID:-//ACME/CAPserver//EN
   S: VERSION:2.1
   S: CAPVERSION=1.0
   S: ITIPVERSION=1.0
   S: AUTH=KERBEROS_V4
   S: AUTH=DIGEST_MD5
   S: CAR=CAR1
   S: MINDATE=19700101T000000Z
   S: MAXDATE=20370201T000000Z
   S: END:VCALENDAR
   S: .
```

This example shows a failed authentication:


```
   C: AUTHENTICATE KERBEROS_V4
   S: AmFYig==
   C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
   S: .
   S: 6.0
```

7.1.4 CONTINUE Command

```
Arguments:  latency time in seconds (optional)
Data:       noneResult:     results from the command in progress
            2.0.2 _ reply pending.
```

The CONTINUE command is issued by the client in response to a SENDATA timeout. When a timeout value is specified on the SENDDATA command, the server must issue a reply to the client within the specified time. If the latency time has elapsed prior to the server completing the command it returns a timeout response code. If the client wants the server to continue processing the command it responds with the CONTINUE command.

If latencyTime is present, it must be a positive integer that specifies the maximum number of seconds the client will wait for the next response. If it is omitted, the receiver waits an indefinite period of time for the response.

In this example, the client requests a response from the server every 10 seconds.

```
...
C: SENDDATA:10
C: Content-Type:text/calendar; method=READ; component=VEVENT
C:
C: BEGIN:VCALENDAR
#  etc
C: END:VCALENDAR
C: .
#  after 10 seconds...
S: .
S: 2.0.2
C: CONTINUE:10
S: 2.0
S: Content-type:text/calendar; Method=RESPONSE;Component=VDATA;
S:   Optinfo=VERSION:2.1
S:
S: BEGIN:VCALENDAR
S: VERSION:2.1
S: CALID:cap://cal.example.com/relcal2
#  etc.
S: END:VCALENDAR
S: .
```


7.1.5 DISCONNECT Command

Arguments: none

Data:

Result: 2.0
The DISCONNECT command is used by a client to terminate a connection. It always succeeds.
Example:

```
C: DISCONNECT
#  [ed. Note: should the client now wait for a response from the server
#             before disconnecting? ]S: 2.0
C: <drops connection>
S: <drops connection>
```

7.1.6 IDENTIFY Command

Arguments: Identity to assume

Data: None

Result: 2.0
6.4 Identity not permitted

The "IDENTIFY" command allows the CUA to select a new identity to be used for calendar access. This command may only be called in the Authenticated State.

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

7.1.7 SENDDATA Command

Arguments: [latencyTime]

Data: a MIME encapsulated iCalendar object

Result: 2.0.1 - Server will now accept input until <CRLF>.<CRLF> is encountered.

The SENDDATA command is used to send calendar requests and commands to the server. After a response code of 2.0.1 is issued the CUA sends a MIME encapsulated iCalendar object to the server. The end of this MIME data is signaleled by the special sequence <CRLF>.<CRLF> .

7.1.8 STARTTLS Command

Arguments: None

Data: None

Result: 2.0

6.5 TLS not supported

The "STARTTLS" command is issued by the CUA to indicate to the CS that it wishes to negotiate transport level security using [TLS]. If the CS does not support TLS it returns status code 6.5. If the CS supports TLS it issues an initial response of 2.0.12 indicating that the CUA should proceed with TLS negotiation. Once the TLS negotiation is complete the server returns the response code 2.0.

After issuing the "STARTTLS" command the CUA issues the "AUTHENTICATE" command. The SASL external mechanism may be used if the CUA wishes to use the authentication id which was used in the TLS negotiation. If an authentication id was determined during TLS negotiations it MUST NOT be used for the purpose of granting a CAP session identity unless the CUA authenticates using the SASL external mechanism.

The CUA MUST NOT issue a "STARTTLS" if it has already issued an "AUTHENTICATE" or "STARTTLS" command in this session. If a CUA does this the CS must terminate the session.

The following examples illustrate the use of the "STARTTLS" command:

Unsupported TLS:

```
C: STARTTLS
S: 6.5
```

Supported TLS:

```
C: STARTTLS
S: 2.0.12
  <tls negotiation>
S: 2.0
```

7.2 Application Protocol Commands

7.2.1 Calendaring Commands

The following methods provide a set of calendaring commands in CAP. Calendaring commands (or methods) allow a CU to directly manipulate a calendar.

Calendar access rights can be granted for the more generalized access provided by the calendar commands.

7.2.1.1 CREATE Method

Arguments: objtype
Data: no specific data for this command
Result: 2.0 - successfully created the component or calendar
6.0 _ Permission denied
6.1 - Container(s) not found 6.2 - Calendar or component already exists
Bad args

The CREATE method is used to create a new iCalendar object of type objtype. ContainerId1 through ContainerIdn specify the container(s) for the create. When creating a new calendar at the top level, the CSID is specified. Otherwise the container will be a CalID.

7.2.1.1.1 Creating New Calendars

Example to create a new calendar named "Bill's Soccer Team" in several different containers. In the following example, the client is in the Authenticated state with CS cal.example.com.

```
C: SENDDATA
C: CONTENT-TYPE: text/calendar;method=CREATE;component=VCOMMAND
C: Content-Transfer-Encoding:7bit
C:
C: BEGIN:VCALENDAR
C: VERSION:2.1
C: BEGIN:VCOMMAND
C: METHOD:CREATE;VCALENDAR
C: TARGET:cap://cal.example.com/
C: TARGET:relcal4
C: TARGET://bobo.ex.com/
C: TARGET:relcal5
C: TARGET:cap://cal.example.com/relcal8
C: TARGET:relcal9
C: BEGIN:VCALENDAR
C: RELCALID:relcalz
C: NAME:CHARSET=us-ascii;LANGUAGE=EN-us:Bill's Soccer Team
C: OWNER:capcar:bill
C: OWNER:capcar:mary
C: CALMASTER:mailto:bill@example.com
C: PREFERRED-TZID:US_PST
```

```
   C: BEGIN:VCAR
   C: CARID:12345
   C: GRANT;CN="Bill Jones":UPN=capcar:bill;ACTION=ALL;OBJECT=all
   C: GRANT;CN="Mary Jones":UPN=capcar:mary;ACTION=read;OBJECT=all
   C: END:VCAR
   C: END:VCALENDAR
   C: END:VCOMMAND
   C: END:VCALENDAR
   C: .
   S: 6.0 cap://cal.example.com/
   S: 2.0 cap://cal.example.com/relcal4 cap://cal.example.com/relcalz
   S: 3.1.4 cap://bobo.ex.com/
   S: 6.2 cap://cal.example.com/relcal5
   S: 3.1.5 cap://cal.example.com/relcal8
   S: 7.0 cap://cal.example.com/relcal9
```

If the example above, the Relative CALID is specified. The values for this property must be unique on a CS. That is the reason for the 3.1.5 error response.

In the example below, the Relative CalID is not specified. So, the CAP server will generate one for each calendar successfully created. The

value of the Relative CalID appears as the second parameter on the response code.

```
   S: 6.0 cap://cal.example.com/
   S: 2.0 cap://cal.example.com/relcal4 cap://cal.example.com/rand123
   S: 3.1.4 cap://bobo.ex.com/
   S: 6.2 cap://cal.example.com/relcal5
   S: 3.1.4 cap://cal.example.com/relcal8
   S: 2.0 cap://cal.example.com/relcal9 cap://cal.example.com/rand456
```

Example to create a new component.

```
   C: SENDDATA
   C: Content-Type:text/calendar; method=CREATE; charset=US-ASCII
   C: Content-Transfer-Encoding:7bit
   C:
   C: BEGIN:VCALENDAR
   C: VERSION:2.1
   C: CMDID:abcde
```

```
C: METHOD:CREATE
C: TARGET:cap://cal.foo.com/relcal1
C: TARGET:relcal2
C: BEGIN:VEVENT
C: DTSTART:19990307T180000Z
C: UID:abcd12345
C: DTEND:19990307T190000Z
C: SUMMARY:Important Meeting
C: END:VEVENT
C: END:VCALENDAR
C: .
S: 2.0
S: Content-Type:text/calendar; method=RESPONSE; OPTINFO="CMDID:abcde"
S:
S: BEGIN:VCALENDAR
S: VERSION:2.1
S: CMDID:abcde
S: METHOD:RESPONSE
S: BEGIN:VEVENT
S: REQUEST-STATUS:2.0;cap://cal.foo.com/relcal1 abcd12345
S: REQUEST-STATUS:2.0;cap://cal.foo.com/relcal2 abcd12345
S: END:VEVENT
S: END:VCALENDAR
```

[Editors Note: this returns the calendar and UID? Is this right? It could also be UID and RecurrenceID ? what about if the event has an RRULE?]

7.2.1.2 DELETE Method

```
Arguments:  ContainerId1 [;...ContainerIdn]
Data:       no specific data for this command
Result:     2.0 - successfully deleted the component or calendar
            Permission
            Calendar or component not found
```



```
            Bad args
            Container(s) not found
```

The DELETE method is used to delete a calendar or component. ContainerId1 through ContainerIdn specify the container(s) for the delete. When deleting a calendar at the top level, the CSID is specified. Otherwise the container will be a CalID.

Example to delete a calendar at the top level:

```
   C: SENDDATA
   C: Content-Type:text/calendar; method=DELETE; component=VCOMMAND
   C: Content-Transfer-Encoding:7bit
   C:
   C: BEGIN:VCALENDAR
   C: BEGIN:VCOMMAND
   C: METHOD:DELETE
   C: TARGET:cap://cal.foo.com/bill
   C: BEGIN:VQUERY
   C: SCOPE:VEVENT
   C: QUERY SELECT="UID"
   C: WHERE (UID EQ abcd12345)
   C: END:VQUERY
   C: END:VCOMMAND
   C: END:VCALENDAR
   C: .
   S: 2.0 cap://cal.foo.com/bill
```

7.2.1.3 GENERATEUID Method

```
Arguments:  number of uids to generate
Data:       new uids

Result:     2.0
GENERATEUID returns one or more new unique identifier which MUST be
unique on the server's calendar store. It is recommended that the return
value be a globally unique id.
Example:
C: GENERATEUID 2
S: 2.0  abcde1234567-asdf-lkhh abcde1234567-asdf-3455
```

7.2.1.4 MODIFY Method

```
Arguments:  ContainerId1 [...ContainerIdn]
Data:       no specific data for this command
Result:     2.0 - successfully modified the component or calendar
            Permission
            Calendar or component not found
            Bad args
            Container(s) not found
The MODIFY method is used to change an existing calendar or component.
ContainerId1 through ContainerIdn specify the container(s) of the
modification. When modifying a calendar at the top level, the CSID is
specified. Otherwise the container will be a CalID.
```

In the example below, the start and end time of the event with UID abcd12345 is changed and the LOCATION property is removed.

```
C: SENDDATA
C: Content-type:text/calendar; Method=MODIFY; Component=VCOMMAND
C:
C: BEGIN:VCALENDAR
C: VERSION:2.1
C: METHOD:MODIFY;VEVENT
C: TARGET:relcal2
C: BEGIN:VCOMMAND
C: BEGIN:VQUERY
C: SCOPE:VEVENT
C: QUERY SELECT="UID"
C: WHERE (UID EQ abcd12345)
C: END:VQUERY
C: BEGIN:VOLD
C: DTSTART:19990421T160000Z
C: DTEND:19990421T163000Z
C: LOCATION:Joe's Diner
C: END:VOLD
C: BEGIN:VNEW
C: DTSTART:19990421T160000Z
C: DTEND:19990421T163000Z
C: END:VNEW
C: END:VCOMMAND
C: END:VCALENDAR
C: .
S: 2.0 cap://cal.example.com/relcal2
```

7.2.1.5 MOVE Method

Arguments: ContainerId
Data: data as described below

Result: 2.0 _ success
2.2 _ will attempt operation on the remote cap server
Permission
Calendar already exists
Bad args
Parent Calendar(s) not found

This method is used to move a calendar within the CS's hierarchy of calendars.

[Editors Note: there could be VCAR issues with this... if a VCAR's scope of influence is limited to a calendar, we're probably OK. We should

discuss this one]

7.2.1.6 READ Method

Arguments: ContainerId
Data: data as described below

Result: 2.0 _ successful and the requested data follows

2.2 _ will attempt read on the remote cap server
Permission
Calendar already exists
Bad args
Parent Calendar(s) not found

Read Events
In the example below events on March 10,1999 between 080000Z and 190000Z are read. In this case only 4 properties for each event are returned. Two calendars are specified. In the example, the CAP server is capable of

```
C: SENDDATA
C: Content-type:text/calendar; Method=READ; Component=VQUERY
C:
C: BEGIN:VCALENDAR
C: VERSION:2.1
C: METHOD:READ
C: CMDID:xyz12345
C: TARGET:relcal2
C: TARGET:cap://bobo.ex.com/relcal3
C: BEGIN:VQUERY
C: QUERY:SELECT (DTSTART,DTEND,SUMMARY,UID);
C:  FROM VEVENT;
C:  WHERE (DTEND >= 19990310T080000Z AND
C:        DTSTART <= 19990310T190000Z);
C:  ORDERBY (DTSTART ASC, DTEND, UID, SUMMARY)
C: END:VQUERY
C: END:VCALENDAR
C: .
S: 2.0 cap://cal.example.com/relcal2
S: Content-type:text/calendar; Method=RESPONSE;
S:   Optinfo=VERSION:2.1
S: Content-Transfer-Encoding: 7bit
```

```
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.1
   S: METHOD:RESPONSE
   S: BEGIN:VEVENT
   S: DTSTART:19990310T090000Z
   S: DTEND:19990310T100000Z
   S: UID:abcxyz12345
   S: SUMMARY:Meet with Sir Elton
   S: END:VEVENT
   S: BEGIN:VEVENT
   S: DTSTART:19990310T130000Z
   S: DTEND:19990310T133000Z
   S: UID:abcxyz8999
   S: SUMMARY:Meet with brave brave Sir Robin
   S: END:VEVENT
   S: END:VCALENDAR
   S: .
   S: 2.0 cap://bobo.ex.com/relcal3
   S: Content-type:text/calendar; Method=RESPONSE;Component=VDATA;
```



```
   S:   Optinfo=VERSION:2.1
   S: Content-Transfer-Encoding: 7bit
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.1
   S: METHOD:RESPONSE
   S: BEGIN:VDATA
   S: BEGIN:VEVENT
   S: DTSTART:19990310T140000Z
   S: DTEND:19990310T150000Z
   S: UID:123456asdf
   S: SUMMARY:Summer Budget
   S: END:VEVENT
   S: END:VDATA
   S: END:VCALENDAR
   S: .
```

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 events, but has been denied access to all

properties called out in the request, the response will contain a single RESPONSE-CODE property indicating the error. That is, the VEVENT components will be the following:

```
   S: 2.0 cap://bobo.ex.com/sally
   S: Content-type:text/calendar; Method=RESPONSE;Component=VDATA;
   S:   Optinfo=VERSION:2.1
   S: Content-Transfer-Encoding: 7bit
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.1
   S: BEGIN:VDATA
   S: BEGIN:VEVENT
   S: RESPONSE-CODE:3.8
   S: END:VEVENT
   S: END:VDATA
   S: END:VCALENDAR
   S: .
```

If the UPN has no access to any events at all, the response will simply be an empty data set. The response looks the same if there are particular events to which the requester has been denied access.

```
   S: 2.0 cap://bobo.ex.com/sally
   S: Content-type:text/calendar; Method=RESPONSE;Component=VDATA;
   S:   Optinfo=VERSION:2.1
   S: Content-Transfer-Encoding: 7bit
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.1
   S: BEGIN:VDATA
   S: END:VDATA
   S: END:VCALENDAR
```



```
   S: .
```

Find alarms within a range of time.

```
   C: SENDDATA
   C: Content-type:text/calendar; Method=READ; Component=VQUERY
   C:
   C: BEGIN:VCALENDAR
   C: VERSION:2.1
   C: METHOD:READ
```

```
C: CMDID:xyz12345
C: TARGET:relcal2
C: TARGET:cap://bobo.ex.com/relcal3
C: BEGIN:VQUERY
C: QUERY:SELECT (VEVENT.DTSTART,
    VEVENT.DTEND,VEVENT.SUMMARY, VEVENT.UID,
    VALARM.*);
C:  FROM VEVENT,VTODO;
C:  WHERE (VALARM.TRIGGER >= 19990310T080000Z AND
C:         VALARM.TRIGGER <= 19990310T190000Z);
C:  ORDERBY (VALARM.TRIGGER ASC)
C: END:VQUERY
C: END:VCALENDAR
C: .
S: 2.0 cap://bobo.ex.com/relcal3
S: Content-type:text/calendar; Method=RESPONSE;
S:   Optinfo=VERSION:2.1
S: Content-Transfer-Encoding: 7bit
S:
S: BEGIN:VCALENDAR
S: VERSION:2.1
S: METHOD:RESPONSE
S: CMDID:xyz12345
S: TARGET:cap://bobo.ex.com/relcal3
S: BEGIN:VEVENT
S: DTSTART:19990310T130000Z
S: DTEND:19990310T133000Z
S: UID:abcxyz8999
S: SUMMARY:Meet with brave brave Sir Robin
S: BEGIN:VALARM
S: TRIGGER:19990310T132500Z
S: SUMMARY:Almost time..
S: ACTION:DISPLAY
S: END:VALARM
S: END:VEVENT
S: END:VCALENDAR
S: .
S: 2.0 cap://bobo.ex.com/relcal2
S: Content-type:text/calendar; Method=RESPONSE;
S:   Optinfo=VERSION:2.1
S: Content-Transfer-Encoding: 7bit
S:
S: BEGIN:VCALENDAR
S: VERSION:2.1
S: METHOD:RESPONSE
```

```
S: CMDID:xyz12345
S: TARGET:cap://bobo.ex.com/relcal2
S: BEGIN:VEVENT
S: REQUEST-STATUS:2.0
S: END:VEVENT
S: END:VCALENDAR
S: .
```

7.2.2 Scheduling Commands

The following provide a set of scheduling commands (or methods) in CAP. Scheduling commands allow a CU to indirectly manipulate a calendar by asking another CU to perform an operation on their calendar. For example, CU-A can request CU-B to add a meeting to their calendar; in effect inviting CU-B to the meeting.

Calendar access rights can be granted for scheduling commands without granting rights for more generalized access with the calendar commands.

[Editors Note: This section needs to be completed by adding the restriction tables for each of these iTIP methods. The basis for the text is to be taken from [RFC2446].]

7.2.2.1 PUBLISH

7.2.2.2 REQUEST

7.2.2.3 REPLY

7.2.2.4 ADD

7.2.2.5 CANCEL

7.2.2.6 REFRESH

7.2.2.7 COUNTER

7.2.2.8 DECLINECOUNTER

7.2.3 iTIP Examples

The following examples describe scenarios for the handling of incoming iTIP data. An appropriate sort-order for the handling of icoming iTIP is by UID, Recurrence-id, sequence, dtstamp. This processing may be optimized, for instance, REFRESHs could be processed last.

As an update to [RFC2446], data with the "COUNTER" method should be processed even if the Seqeunce number is stale.

7.2.3.1 Sending and Receiving an iTIP request

In this example A invites B and C to a meeting, B accepts the meeting and C rejects it. The calendars for A, B and C are relcal1, relcal2

and relcal3 respectively, and are all on the same server, "cal.foo.com". A lot of these described actions are performed by the CUAs and not the users themselves, the CUAs are called A-c, B-c and C-c respectively.

A wishes to create a meeting with B and C, so A-c uses CAP to send the following iTIP request to relcal2 and relcal3, while logged in to "cal.foo.com".

```
BEGIN:VCALENDAR
VERSION:2.1
CMDID:xhj-dd
METHOD:REQUEST
TARGET:cap://cal.foo.com/relcal2
TARGET:relcal3
BEGIN:VEVENT
UID:abcd12345
DTSTART:19990307T180000Z
DTEND:19990307T190000Z
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE;RSVP=TRUE;PARTSTAT=NEEDS-ACTION:cap://cal.foo.com/relcal2
ATTENDEE;RSVP=TRUE;PARTSTAT=NEEDS-ACTION:cap://cal.foo.com/relcal3
SUMMARY:Important Meeting
END:VEVENT
END:VCALENDAR
```

An incoming event (indicated by the value of the "METHOD" property) then appears in relcal2 and relcal3, with the following data:

```
BEGIN:VEVENT
METHOD:REQUEST
UID:abcd12345
DTSTART:19990307T180000Z
DTEND:19990307T190000Z
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE;RSVP=TRUE;PARTSTAT=NEEDS-ACTION:cap://cal.foo.com/relcal2
ATTENDEE;RSVP=TRUE;PARTSTAT=NEEDS-ACTION:cap://cal.foo.com/relcal3
SUMMARY:Important Meeting
```

```
   END:VEVENT
```

B-c and C-c must search for such incoming events, they do so using the following CAP search:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   METHOD:READ
   CMDID:xhr-de
   TARGET:relcal2
      # or TARGET:relcal3
   BEGIN:VQUERY
   QUERY:SELECT (ALL);
    FROM VEVENT;
    WHERE (METHOD == REQUEST);
   END:VQUERY
   END:VCALENDAR
```


In response to this search they get the above event. B-c and C-c must then crack open the VEVENT, find the UID and determine if there is already an event on their calendar with that UID. To do this they use the following search:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   METHOD:READ
   CMDID:xhr-df
   TARGET:relcal2
   BEGIN:VQUERY
   QUERY:SELECT (ALL);
    FROM VEVENT;
    WHERE (UID == abcd12345);
   END:VQUERY
   END:VCALENDAR
```

We assume that the event is not already in their relcal2 or relcal3, so the read they only returns the original incoming iTIP (the UID matched), but this can be ignored since it is incoming.

B-c prompts B who decides to accept the meeting request, and B-c creates a copy of the event in relcal2, with the "PARTSTAT" parameter set to ACCEPTED. B-c also sends this copy to the Organizer at relcal1 as an

iTIP REPLY, preserving the CMDID:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   CMDID:xhj-dd
   METHOD:REPLY
   TARGET:cap://cal.foo.com/relcal1
   BEGIN:VEVENT
   UID:abcd12345
   DTSTART:19990307T180000Z
   DTEND:19990307T190000Z
   ORGANIZER:cap://cal.foo.com/relcal1
   ATTENDEE;PARTSTAT=ACCEPTED:cap://cal.foo.com/relcal2
   SUMMARY:Important Meeting
   END:VEVENT
   END:VCALENDAR
```

C, on the other hand, decides to decline the meeting, and C-c sends a reply to the Organizer to that effect, as follows:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   CMDID:xhj-dd
   METHOD:REPLY
   TARGET:cap://cal.foo.com/relcal1
   BEGIN:VEVENT
   UID:abcd12345
   DTSTART:19990307T180000Z
   DTEND:19990307T190000Z
```



```
   ORGANIZER:cap://cal.foo.com/relcal1
   ATTENDEE;PARTSTAT=DECLINED:cap://cal.foo.com/relcal3
   SUMMARY:Important Meeting
   END:VEVENT
   END:VCALENDAR
```

It is preferable that C-c store the event in relcal3 even though it has been declined. Storing the event in relcal3 allows subsequent iTIP messages to be interpreted correctly. The "PARTSTAT" parameter indicates that the event was refused, and a tombstone property may be necessary if the user wishes to delete the event.

After receiving the replies from relcal2 and relcal3, A-c updates the

version of the event in relcal1 to indicate the new participation statii:

```
BEGIN:VEVENT
METHOD:REQUEST
UID:abcd12345
DTSTART:19990307T180000Z
DTEND:19990307T190000Z
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE;PARTSTAT=ACCEPTED:cap://cal.foo.com/relcal2
ATTENDEE;PARTSTAT=DECLINED:cap://cal.foo.com/relcal3
SUMMARY:Important Meeting
END:VEVENT
```

A-c then sends a new iTIP request to relcal2 only, indicating the updated information.

7.2.3.2 Handling an iTIP refresh

A little bit later, C is thinking about accepting the event in the previous example, but first wants to check the current state of the event. To find the current state C-c uses the iTIP "REFRESH" method, sending the following to relcal1:

```
BEGIN:VCALENDAR
VERSION:2.1
CMDID:xud-pn
METHOD:REFRESH
TARGET:cap://cal.foo.com/relcal1
BEGIN:VEVENT
UID:abcd12345
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE:cap://cal.foo.com/relcal3
DTSTAMP:19990306T202333Z
END:VEVENT
END:VCALENDAR
```

A-c finds the refresh as an incoming iTIP, and searches for the corresponding event. Having found the event (with no changes since the last example) A-c then verifies that relcal3 is in fact an Attendee of the event and is thus allowed to request a refresh. (In the case of a

published event things are more complicated.) A-c packages the event up

as an iTIP request and sends it to relcal3:

```
BEGIN:VCALENDAR
VERSION:2.1
CMDID: xud-pn
METHOD:REQUEST
TARGET:cap://cal.foo.com/relcal3
BEGIN:VEVENT
UID:abcd12345
DTSTART:19990307T180000Z
DTEND:19990307T190000Z
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE;PARTSTAT=ACCEPTED:cap://cal.foo.com/relcal2
ATTENDEE;PARTSTAT=DECLINED:cap://cal.foo.com/relcal3
SUMMARY:Important Meeting
SEQUENCE:0
DTSTAMP:19990306T204333Z
END:VEVENT
END:VCALENDAR
```

[Ed. - should the CMDID match that of the REFRESH?]

7.2.3.3 Sending and accepting an iTIP counter

Having received the latest copy of the event C wishes to propose a venue for the event, using an iTIP COUNTER. To do this C-c sends the following to relcal1:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   CMDID:zzykjjk
   METHOD:COUNTER
   TARGET:cap://cal.foo.com/relcal1
   BEGIN:VEVENT
   UID:abcd12345
   DTSTART:19990307T180000Z
   DTEND:19990307T190000Z
   ORGANIZER:cap://cal.foo.com/relcal1
   ATTENDEE;PARTSTAT=DECLINED:cap://cal.foo.com/relcal3
   SUMMARY:Important Meeting
   LOCATION:La Belle Province
   COMMENT:My favourite restaurant\, I'll definitely go if it's there.
   END:VEVENT
   END:VCALENDAR
```

Having sent the information to relcal1, C-c shouldn't store the new details in relcal3. If C-c updated the version in relcal3 and relcal1 did not reply to the counter, then relcal3 would have incorrect information. Instead C-c preserves the correct information and waits for a response from relcal1. A CUA implementation may wish to

preserve this information itself, externally to the CS.

In order to receive an iTIP counter A-c follows the same search as for other iTIP data, first find the incoming message, next find any matching events in the calendar store.

Having found the matching event, A reviews the proposed changes and decides to accept the COUNTER. To do this, A-c modifies the version in relcal1 (bumping the sequence number) to:

```
BEGIN:VEVENT
METHOD:CREATE
UID:abcd12345
DTSTART:19990307T180000Z
DTEND:19990307T190000Z
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE;PARTSTAT=ACCEPTED:cap://cal.foo.com/relcal2
ATTENDEE;PARTSTAT=DECLINED:cap://cal.foo.com/relcal3
SUMMARY:Important Meeting
LOCATION:La Belle Province
SEQUENCE:1
END:VEVENT
```

A-c then sends the updated version as a request to both relcal2 and relcal3:

```
BEGIN:VCALENDAR
VERSION:2.1
CMDID:xup-po
METHOD:REQUEST
TARGET:cap://cal.foo.com/relcal2
TARGET:cap://cal.foo.com/relcal3
BEGIN:VEVENT
UID:abcd12345
DTSTART:19990307T180000Z
DTEND:19990307T190000Z
ORGANIZER:cap://cal.foo.com/relcal1
ATTENDEE;RSVP=TRUE;PARTSTAT=NEEDS-ACTION:cap://cal.foo.com/relcal2
ATTENDEE;RSVP=TRUE;PARTSTAT=NEEDS-ACTION:cap://cal.foo.com/relcal3
SUMMARY:Important Meeting
LOCATION:La Belle Province
SEQUENCE:1
```

```
   DTSTAMP:19990307T054339Z
   END:VEVENT
   END:VCALENDAR
```

7.2.3.4 Declining an iTIP counter

B does not like the new location and also counters the event, B-c sends the following iTIP:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   CMDID:xim-ef
   METHOD:COUNTER
```



```
   TARGET:cap://cal.foo.com/relcal1
   BEGIN:VEVENT
   UID:abcd12345
   DTSTART:19990307T180000Z
   DTEND:19990307T190000Z
   ORGANIZER:cap://cal.foo.com/relcal1
   ATTENDEE:cap://cal.foo.com/relcal2
   SUMMARY:Important Meeting
   LOCATION:Au Coin Dor=E9
   END:VEVENT
   END:VCALENDAR
```

However, C does not accept the counter, and C-c replies with a decline counter:

```
   BEGIN:VCALENDAR
   VERSION:2.1
   CMDID:xim-ef
   METHOD:DECLINE-COUNTER
   TARGET:cap://cal.foo.com/relcal2
   BEGIN:VEVENT
   DTSTAMP:19990307T093245Z
   UID:abcd12345
   ORGANIZER:cap://cal.foo.com/relcal1
   SEQUENCE:1
   END:VEVENT
   END:VCALENDAR
```

Fortunately B-c kept the original information when sending the counter, and there is no problem when no information is returned in the DECLINE-COUNTER.

8. Response Codes

Numeric response codes are returned at both the transport and application layer. The same set of codes is used in both cases.

[Editors Note: Do we want to use the same set of codes?]

The format of these codes is described in [RFC2445], and extend in [RFC2446] and [RFC2447]. The following describes new codes added to this set.

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 [RFC2445], to make the accompanying text optional.

Code	Params	Description
2.0	**varies**	Success. The parameters vary with the operation and are specified
2.0.1	**none**	Success, send data, terminate with <CRLF>.<CRLF>
2.0.2		**A reply is pending. It could not be completed in the specified amount of time. The server awaits** a CONTINUE or ABORT command.
2.0.3		**In response to the client issuing an ABORT command, this reply code indicates that any** command currently underway was successfully aborted.
2.0.6		**An operation is being attempted on a remote server. This response indicates that the server** has not yet been contacted but an attempt will be made to contact it after the command has been sent.

3.1.4	**Capability not supported**
4.1	**Calendar store access denied**
6.1	**authenticate failure: unsupported authentication mechanism, credentials rejected**
6.2	**Sender aborted authentication, authentication exchange cancelled**
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 Receiver that prevents the operation from succeeding.**
8.1	**Sent when a session cannot be established because the CAP Server is too busy.**
8.2	**Used to signal that an ICAL object has exceeded the server's size limit.**
8.3	**A DATETIME value was too large 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 id specified is already in use.**
8.6	**ID clash**
9.0	**An unrecongnized command was received.**
10.1	**Accompanied by an alternate address. The**

RECIPIENT specified should be contacted at the given alternate address. The referral address MUST follow the reply code.

10.2	**The server is shutting down.**

10.4 **The operation has not be performed because it would cause the resources (memory, disk,CPU, etc)** to exceed the allocated quota.

10.5 **The ITIP message has been queued too too long. Delivery has been aborted.**

9. Detailed SQL Schema

This section describes a conceptual schema for object model in CAP. It is used as the basis for querying data managed by the CS. This is only a conceptual schema. Implementations can use any schema they like so long as they are prepared to map CAP queries that are expressed in this conceptual schema. Implementations are not required to use SQL database technology. The protocol is designed such that a CUA does not need to handle these queries.

This schema is based on SQL-92 [SQL] along with the [SQLCOM] corrections.

Properties than can occur multiple times are intended to be put in separate tables. For example

```
BEGIN:VEVENT
UID:1
DTSTART:19990326T201400Z
ORGANIZER:mailto:sam@abc.COM
SUMMARY:I have 2 attachments
ATTACHMENT;FMTTYPE=audio/basic:ftp://host.com/pub/sounds/bell.au
ATTACHMENT;FMTTYPE=audio/basic:ftp://host.com/pub/sounds/bell2.au
END:VEVENT
```

There are two ATTACHMENT properties each having a unique value. These are kept in separate tables. This is diagrammed below. The diagram is not a complete representation of the VEVENT table. It is an abbreviated table used to illustrate how properties that can occur multiple times are intended to be represented.

ABBREVIATED VEVENT TABLE

UID	DTSTART	ORGANIZER	SUMMARY	ATTACH_LIST
1	19990326T201400Z	mailto:sam@abc.com	I have 2 attachments	123
999	19700101T000000Z	mailto:usr@host.com	I have no attachments	

+----+----------------+------------------+------------+------------+

ABBREVIATED ATTACH_LIST TABLE

ATTACH_LIST	VALUE	INLINE_BLOB
123	ftp://host.com/pub/sounds/bell.au	
123	ftp://host.com/pub/sounds/bell2.au	
234		MIICajCCAdO- gAwIBAgICBEU <...remainder of "BASE64" encoded binary data...>

9.1 iCalendar Store Schema

The following defines the schema for an iCalendar object and the components, properties, and parameters defined in [RFC2445].

```
Create table VCALENDAR {
     RELATIVECALID                  VARCHAR(256) PRIMARY KEY,
     CALMASTER                      VARCHAR(256),
     CHARSET                        VARCHAR(256),
     CHILDREN                       VARCHAR(256)
     LANGUAGE                       CHAR(5)
     LAST_MODIFIED
     NAME                           VARCHAR(256),
     OWNERS
     PARENT                         CHAR(16),
     PATH
     SCHEDULABLE_HOURS
     TOMBSTONE
     TZID
     LAST_MODIFIED_BY
};

create table VEVENT {
     ATTACH_LIST                    INTEGER,
     ATTENDEE_LIST                  INTEGER,
     /* CATEGORIES may contain a comma seperated list */
     CATEGORIES                     VARCHAR(len?),
```

```
     CLASS                          INTEGER,
     CLASS_PARAMS                   INTEGER,
     COMMENT                        VARCHA,
     COMMENT_PARAMS                 INTEGER,
     CONTACT_LIST                   INTEGER,
     CREATED                        TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     CREATED_PARAMS                 INTEGER,
     DESCRIPTION                    VARCHAR(len?),
```



```
     DESCRIPTION_PARAMS             INTEGER,
     DTEND                          TIMESTAMP,
     DTEND_PARAMS                   INTEGER,
     DTSTAMP                        TIMESTAMP NOT NULL,
     DTSTAMP_PARAMS                 INTEGER,
     DTSTART                        TIMESTAMP NOT NULL,
     DTSTART_PARAMS                 INTEGER,
     DURATION                       <?type?>,
     DURATION_PARAMS                INTEGER,
     EXDATE_LIST                    INTEGER,
     EXRULE_LIST                    INTEGER,
     GEO_LAT                        NUMBER,
     GEO_LON                        NUMBER,
     GEO_PARAMS                     INTEGER,
     LAST_MODIFIED                  TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     LAST_MODIFIED_PARAMS           INTEGER,
     LOCATION                       VARCHA,
     LOCATION_PARAMS                INTEGER,
     METHOD                         VARCHAR(len20?),
     ORGANIZER                      VARCHAR(len?) NOT NULL,
     ORGANIZER_PARAMS               INTEGER,
     PRIORITY                       INTEGER,
     PRIORITY_PARAMS                CHAR(1),
     RECURRENCE_ID                  VARCHAR(len?),
     RECURRENCE_ID_PARAMS           INTEGER,
     RDATE_LIST                     INTEGER,
     RELATED_TO_LIST                INTEGER,
     /* RESOURCES may contain a comma seperated list */
     RESOURCES                      VARCHAR(len?),
     RESOURCES_PARAMS               INTEGER,
     RRULE_LIST                     INTEGER,
```

```
     SUMMARY                        VARCHAR(len?) NOT NULL DEFAULT "",
     SUMMARY_PARAMS                 INTEGER,
     SEQUENCE                       INTEGER NOT NULL DEFAULT 0,
     SEQUENCE_PARAMS                INTEGER,
     STATUS                         INTEGER,
     STATUS_PARAMS                  CHAR(1),
     TRANSP                         CHAR(1),
     TRANSP_PARAMS                  INTEGER,
     UID                            VARCHAR(len?) NOT NULL,
     UID_PARAMS                     INTEGER,
     URL                            VARCHA,
     URL_PARAMS                     INTEGER,
     X_PROP_LIST                    INTEGER,
     VALARM_LIST                    INTEGER,
};

create table VTODO {
     ATTENDEE_LISTINTEGER,
     ATTACH_LIST                    INTEGER,
     /* CATEGORIES may contain a comma separated list */
     CATEGORIES                     VARCHAR(len?),
     CLASS                          INTEGER,
```



```
     CLASS_PARAMS                   INTEGER,
     COMMENT                        VARCHAR(len?),
     COMMENT_PARAMS                 INTEGER,
     CONTACT_LIST                   INTEGER,
     CREATED                        TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     CREATED_PARAMS                 INTEGER,
     DESCRIPTION                    VARCHAR(len?),
     DESCRIPTION_PARAMS             INTEGER,
     DTSTAMP                        TIMESTAMP NOT NULL,
     DTSTAMP_PARAMS                 INTEGER,
     DTSTART                        TIMESTAMP NOT NULL,
     DTSTART_PARAMS                 INTEGER,
     DUE                            TIMESTAMP,
     DUE_PARAMS                     INTEGER,
     DURATION                       <?type?>,
     DURATION_PARAMS                INTEGER,
     EXDATE_LIST                    INTEGER,
     EXRULE_LIST                    INTEGER,
```

```
     GEO_LAT                        NUMBER,
     GEO_LON                        NUMBER,
     GEO_PARAMS                     INTEGER,
     LAST_MODIFIED                  TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     LAST_MODIFIED_PARAMS           INTEGER,
     LOCATION                       VARCHA,
     LOCATION_PARAMS                INTEGER,
     METHOD                         VARCHAR(len20?),
     ORGANIZER                      VARCHAR(len?) NOT NULL,
     ORGANIZER_PARAMS               INTEGER,
     PERCENT_COMPLETE               INTEGER,
     PERCENT_COMPLETE_PARAMSLETE    INTEGER
     PRIORITY                       INTEGER NOT NULL,
     PRIORITY_PARAMS                INTEGER,
     RDATE_LIST                     INTEGER,
     RECURRENCE_ID                  VARCHAR(len?),
     RECURRENCE_ID_PARAMS           INTEGER,
     /* RESOURCES may contain a    comma seperated list */
     RESOURCES                      VARCHAR(len?),
     RESOURCES_PARAMS               INTEGER,
     RRULE_LIST                     INTEGER,
     SEQUENCE                       INTEGER NOT NULL DEFAULT 0,
     SEQUENCE_PARAMS                INTEGER,
     SUMMARY                        VARCHAR(len?) NOT NULL DEFAULT "",
     SUMMARY_PARAMS                 INTEGER,
     UID                            VARCHAR(len?) NOT NULL,
     UID_PARAMS                     INTEGER,
     URL                            VARCHAR(len?)
     URL_PARAMS                     INTEGER,
     X_PROP_LIST                    INTEGER
     VALARM_LIST                    INTEGER,
};

create table VJOURNAL {
```



```
     ATTACH_LIST                    INTEGER,
     /* CATEGORIES may contain a comma seperated list */
     CATEGORIES                     VARCHAR(len?),
     CLASS                          INTEGER,
     CLASS_PARAMS                   INTEGER,
     COMMENT                        VARCHAR(len?),
```

```
     COMMENT_PARAMS                INTEGER,
     CONTACT_LIST                  INTEGER,
     CREATED                       TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     CREATED_PARAMS                INTEGER,
     DESCRIPTION                   VARCHAR(len?) NOT NULL DEFAUT "",
     DESCRIPTION_PARAMS            INTEGER,
     DTSTAMP                       TIMESTAMP NOT NULL,
     DTSTAMP_PARAMS                INTEGER,
     DTSTART                       TIMESTAMP NOT NULL,
     DTSTART_PARAMS                INTEGER,
     EXDATE_LIST                   INTEGER,
     EXRULE_LIST                   INTEGER,
     LAST_MODIFIED                 TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     METHOD                        VARCHAR(len20?),
     LAST_MODIFIED_PARAMS          INTEGER,
     ORGANIZER                     VARCHAR(len?) NOT NULL,
     ORGANIZER_PARAMS              INTEGER,
     RDATE_LIST                    INTEGER,
     RECURRENCE_ID                 VARCHAR(len?),
     RECURRENCE_ID_PARAMS          INTEGER,
     RELATED_TO_LIST               INTEGER,
     RRULE_LIST                    INTEGER,
     SEQUENCE                      INTEGER NOT NULL DEFAULT 0,
     SEQUENCE_PARAMS               INTEGER,
     STATUS                        INTEGER,
     STATUS_PARAMS                 CHAR(1),
     SUMMARY                       VARCHAR(len?) NOT NULL DEFAULT "",
     SUMMARY_PARAMS                INTEGER,
     UID                           VARCHAR(len?) NOT NULL,
     UID_PARAMS                    INTEGER,
     X_PROP_LIST                   INTEGER
};
```

An implementation may not actually have a VFREEBUSY table as the information may be produced dynamicly. However a CS MUST appear to provide this table as this may be how a CUA chooses to query for VFREEBUSY information while using [CAP]. Example, it probabily would not make any sense for ATTENDEE to exist in this table, yet a CUA may wish to ask for the VFREEBUSY for an ATTENDEE.

```
create table VFREEBUSY {
     ATTENDEE_LIST      VARCHAR(len?),
     COMMENT            VARCHAR(len?),
     COMMENT_PARAMS     INTEGER,
     CONTACT_LIST       INTEGER,
```

```
     DTEND              TIMESTAMP NOT NULL,
     DTEND_PARAMS       INTEGER,
     DTSTAMP            TIMESTAMP NOT NULL,
     DTSTAMP_PARAMS     INTEGER,
     DTSTART            TIMESTAMP NOT NULL,
     DTSTART_PARAMS     INTEGER,
     FREEBUSY_LIST      INTEGER NOT NULL,
     METHOD                       VARCHAR(len20?),
     ORGANIZER          VARCHAR(len?) NOT NULL,
     ORGANIZER_PARAMS   INTEGER,
     X_PROP_LIST        INTEGER
     URL                VARCHAR(len?)
};

create table VTIMEZONE {
     DAYLIGHT_LIST   INTEGER, /* In TZ_LIST table */
     STANDARD_LIST   INTEGER, /* In TZ_LIST table */
     TZID            VARCHAR(len?) NOT NULL,
     TZID_PARAM      INTEGER,
     TZURL           VARCHAR(len?) NOT NULL,
     TZURL_PARAM     INTEGER,
     X_PROP_LIST     INTEGER
};

create table TZ_LIST {
     /* Maps to DAYLIGHT_LIST   or STANDARD_LIST in VTIMEZONE table */
     TZ_KEY                     INTEGER,
     COMMENT                    VARCHAR(len?),
     COMMENT_PARAMS             INTEGER,
     DTSTART                    TIMESTAMP NOT NULL,
     DTSTART_PARAMS             INTEGER,
     LAST_MODIFIED              TIMESTAMP NOT NULL DEFAULT
     CURRENT_DATE,
     LAST_MODIFIED_PARAMS       INTEGER,
     RDATE_LIST                 INTEGER,
     RRULE_LIST                 INTEGER,
     TZNAME                     VARCHAR(len?),
     TZOFFSET                   <?type?> NOT NULL,
     TZOFFSETFROM               <?type?> NOT NULL,
     TZOFFSETTO                 <?type?> NOT NULL,
};

create table VALARM_LIST {
     /* Maps to VALARM_LIST   in other tables */
```

```
     VALARM_KEY              INTEGER,
     ACTION                  INTEGER NOT NULL,
     ACTION_PARAMS           INTEGER,
     ATTACH_LIST             INTEGER,
     DESCRIPTION             VARCHAR(len?) NOT NULL DEFAUT "",
     DESCRIPTION_PARAMS      INTEGER,
     DURATION                <?type?>,
     DURATION_PARAMS         INTEGER,
     REPEAT                  INTEGER,
     REPEAT_PARAMS           INTEGER,
```



```
     SUMMARY                 VARCHAR(len?) NOT NULL DEFAULT "",
     SUMMARY_PARAMS          INTEGER,
     TRIGGER_DT              TIMESTAMP,
     TRIGGER_DURATION        <?type?>,
     X_PROP_LIST             INTEGER
};
```

10. Examples

For all the examples in this section, the authenticated user is user@example.com.

10.1 Authentication Examples

10.1.1 Login Using Kerberos V4

Use Kerberos V4 to authenticate as bill@example.com to the CAP server on cal.example.com.

```
   C: <connect to cal.example.com on port ...>
   S: 2.0
   S: CAPVERSION=1.0
   S: ITIPVERSION=1.0
   S: AUTH=KERBEROS_V4
   S: AUTH=DIGEST_MD5
   S: .
   C: AUTHENTICATE KERBEROS_V4
   S: AmFYig==
   C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
   S: or//EoAADZI=
   C: DiAF5A4gA+oOIALuBkAAmw==
   S: 2.0
   S: IDENTITY=bill@example.com
```

```
   S: CAPVERSION=1.0
   S: ITIPVERSION=1.0
   S: AUTH=KERBEROS_V4
   S: AUTH=DIGEST_MD5
   S: CAR=CAR1  appl
   S: MINDATE=19700101T000000Z  appl
   # who knows this date (end of the 32 bit number)?
   S: MAXDATE=20370201T000000Z
   S: .
```

10.1.2 Error Scenarios

Use of SASL Authorization Identity is not supported. Use the IDENTITY command instead. If you attempt to use the Authorization Identity, an error status will be returned.

```
   C: AUTHENTICATE KERBEROS_V4
   S: AmFYig==
   C: BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IJmrMG+25a4DT
   S: or//EoAADZI=
   C: DiAF5A4gA+oOIALuBkAAmw==
   S: 6.1
```



```
   S: .
```

Sender aborted authentication:

```
   C: AUTHENTICATE KERBEROS_V4
   S: AmFYig==
   C: .
   S: 6.2
   S: .
```

Unsupported mechanism:

```
   C: AUTHENTICATE Experimental_Auth
   S: 6.3
   S: .
```

10.2 Read Examples

10.2.1 Read From A Single Calendar

In this example bill@example.com reads a day's worth of events from cap://cal.example.com/opaqueid99.

```
   C: SENDDATA
   C: Content-type:text/calendar; Method=READ; Component=VQUERY
   C:
   C: BEGIN:VCALENDAR
   C: VERSION:2.1
   C: METHOD:READ
   C: CMDID:xyz12345
   C: TARGET:cap://cal.example.com/opaqueid99
   C: BEGIN:VQUERY
   C: QUERY:SELECT (VEVENT.DTSTART,VEVENT.DTEND,SUMMARY,UID);
   C:  FROM VEVENTTABLE;
   C:  WHERE (VEVENT.DTEND >= 19990714T080000Z AND
   C:         VEVENT.DTSTART <= 19990715T080000Z);
   C:  ORDERBY (VEVENT.DTSTART ASC, VEVENT.DTEND, UID, SUMMARY)
   C: END:VQUERY
   C: END:VCALENDAR
   C: .

   # this response code means that the transport successfully
   # delivered the data.
   S: 2.0 ; got the request OK ; I swear
   S: Content-type:text/calendar; Method=RESPONSE;
   S:   Optinfo=VERSION:2.1
   S: Content-Transfer-Encoding: 7bit
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.1
   S: METHOD:RESPONSE
   S: TARGET:cap://cal.example.com/opaqueid99
   S: CMDID:xyz12345
```



```
   # we have not yet discussed response-status

   S: RESPONSE-STATUS:2.0
   S: BEGIN:VEVENT
   S: DTSTART:19990714T200000Z
   S: DTEND:19990714T210000Z
   S: UID:000444888929922
   S: SUMMARY:Blah bla
```

```
   S: END:VEVENT
   S: BEGIN:VEVENT
   S: UID:003484809803888898944 3
   S: SUMMARY:meeting
   S: DTEND:19990714T233000Z
   S: DTSTART:19990714T223000Z
   S: END:VEVENT
   S: END:VCALENDAR
   S: .
```

10.2.2 Read From Multiple Calendars

In this example bill@example.com reads a day's worth of events from cap://cal.example.com/opaqueid101 and cap://cal.example.com/opaqueid103

```
   C: SENDDATA
   C: Content-type:text/calendar; Method=READ; Component=VQUERY
   C:
   C: BEGIN:VCALENDAR
   C: VERSION:2.1
   C: METHOD:READ
   C: CMDID:xyz12346
   C: TARGET:cap://cal.example.com/opaqueid101
   C: TARGET:opaqueid103
   C: BEGIN:VQUERY
   C: QUERY:SELECT (DTSTART,DTEND,SUMMARY,UID);
   C:  FROM VEVENT;
   C:  WHERE (DTEND >= 19990714T080000Z AND
   C:         DTSTART <= 19990715T080000Z);
   C:  ORDERBY (DTSTART ASC, DTEND, UID, SUMMARY)
   C: END:VQUERY
   C: END:VCALENDAR
   C: .
   S: 2.0
   S: Content-Type:multipart/mixed;boundary="--FEE3790DC7E35189CA67"
   S:
   S: ----FEE3790DC7E35189CA67
   S: Content-type:text/calendar; Method=RESPONSE;
   S:   Optinfo=VERSION:2.1
   S: Content-Transfer-Encoding: 7bit
   S:
   S: BEGIN:VCALENDAR
   S: VERSION:2.1
   S: METHOD:RESPONSE
   S: TARGET:cap://cal.example.com/opaqueid103
   S: CMDID:xyz12346
   S: RESPONSE-CODE:2.0
```

```
S: BEGIN:VEVENT
S: UID:0034848098038888989443
S: SUMMARY:meeting
S: DTEND:19990714T233000Z
S: DTSTART:19990714T223000Z
S: END:VEVENT
S: END:VCALENDAR
S:
S: ----FEE3790DC7E35189CA67CE2C
S: Content-type:text/calendar; Method=RESPONSE;
S:   Optinfo=VERSION:2.1
S: Content-Transfer-Encoding: 7bit
S:
S: BEGIN:VCALENDAR
S: VERSION:2.1
S: METHOD:RESPONSE
S: TARGET:cap://cal.example.com/opaqueid101
S: CMDID:xyz12346
S: RESPONSE-CODE:4.1 ; access denied
S: END:VCALENDAR
S:
S: ----FEE3790DC7E35189CA67CE2C
S: .
```

10.2.3 Timeouts

In this example bill@example.com attempts to read a calendar but the latency time he supplies is not sufficient for the server to complete the command.

```
C: SENDDATA 3
C: Content-type:text/calendar; Method=READ; Component=VQUERY
C:
C: BEGIN:VCALENDAR
C: VERSION:2.1
C: METHOD:READ
C: CMDID:xyz12346
C: TARGET:cap://cal.example.com/opaqueid101
C: TARGET:opaqueid103
C: BEGIN:VQUERY
C: QUERY:SELECT (DTSTART,DTEND,SUMMARY,UID);
C:  FROM VEVENT;
C:  WHERE (DTEND >= 19990714T080000Z AND
C:         DTSTART <= 19990715T080000Z);
C:  ORDERBY (DTSTART ASC, DTEND, UID, SUMMARY)
C: END:VQUERY
C: END:VCALENDAR
```

```
   C: .
   S: 7.0 ; timeout
   S: .
```

If Bill wants to continue and give the server more time he would issue a CONTINUE command:

```
   C: CONTINUE 10
```


If Bill wants to abort the command and not wait any further he would issue an ABORT command:

```
   C: ABORT
   S: 2.0
   S: .
```

10.2.4 Using the Calendar Parent, Children Properties

10.2.5 An example that depends on VEVENT.DTSTART and VALARM.DTSTART

11. Implementation Issues

1. What are the minimum component properties set required to create a new VEVENT, VTODO and VJOURNAL?. PROPOSAL: DTSTART, SUMMARY and UID.

2. What is the state of all undefined properties? PROPOSAL: Not defined. So a query will not return them, if they are selected.

12. Properties

[Editors Note: These extensions/changes to iCalendar need to be reformatted to conform to the IANA registration process defined in section 7 of [RFC2445].]

12.1 Calendar Store Properties

Name	Read Only	Description
DEFAULT-VCARS	N	The default VCARs for newly created toplevel calendars
MAXDATE	Y	The date/time in the future beyond which the server cannot represent.
MINDATE	Y	The date/time in the past prior to which

		the server cannot represent.
TIME	Y	Current server time. This is returned as a localtime and TZID

[Editors Note: Should there be something here about how the server handles RRULES and EXRULES? For example, can/MUST the server unzip RRULES/EXRULES? Does it even support RRULES? Can it deal with unbounded RRULES?]

12.2 Calendar Properties

Name	Read Only	Description
CHARSET	N	the default charset for localized strings in this calendar
CHILDREN	Y	the sub-calendars belonging to this calendar.

CREATED	Y	the timestamp of the calendar's create date
LANGUAGE	N	the default language for localizable strings in this calendar
LAST-MODIFIED	N	the timestamp when the properties of the calendar were last updated.
NAME	N	the display name for this calendar. It is a localizable string.
OWNERS	N	a multi instanced property indicating the calendar owner.
PARENT	N	maintained by a CAP server.
PATH	Y	?? human readable path of name. ?? [editors note: I think this is going to be really problematic. Can we do away with this? Or perhaps make it optional?]
RELATIVECALID	N	a unique name for the calendar. It is made

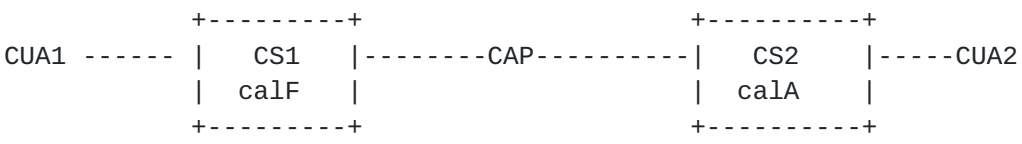
		up of 7 bit ASCII characters.
SCHEDULABLE-HOURS	N	the preferred time range for scheduling events on this calendar.
TOMBSTONE	N	a marker indicating that this calendar has been Deleted.
TZID	N	the id of the timezone associated with this calendar
LAST-MODIFIED-BY	Y	UPN of the person or process that last modified the calendar properties.

13. Security Considerations

For the mandatory SASL mechanism that CAP specifies, the mechanism support is:

MUST authentication
MUST authorization
MAY impersonation

The security issue:



UserListX is not an owner of calF

UserListX has been given ACTONBEHALF of rights to calF by an owner of calF, UserY
UserX authenticates to CS1 as UserX
UserX wants to update the attendee status of an event on calA
An owner of calA has granted access to UserY to update an event they have been invited to
How do we grant UserX access to do this?

[Editors Note: This needs further work and examples.]

14. Changes to iCalendar

[Editors Note: These extensions/changes to iCalendar need to be reformatted to conform to the IANA registration process defined in section 7 of [RFC2445].]

14.1 RIGHTS Value Type

Value Name: RIGHTS

Purpose: This value type is used to identify properties whose value is a calendar access rights.

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

```
rights  = [princ] (policy / carref / cardef) CRLF

princ = "UPN" "=" (text / all / "OWNER" / "NONOWNER")

policy  = ";" "POLICY" "=" policyname

policyname      = "READBUSYTIMEINFO" / "ACTONBEHALFOF" /
"REQUESTONLY"
                / "UPDATEPARTSTATUS" / "OWNER" / iana-name

carref  = ";" "CARREF" "=" text *("," text)

cardef  = action object

action  = ";" "ACTION" "=" act-type *("," act-type)

act-type        = ("CREATE" / "MODIFY" / "DELETE" / "READ" / all)

object  = ";" "OBJECT" "=" (csprop *("," csprop) [propvalue])
                / (calprop *("," calprop) [propvalue])
                / (component *("," component)) [compvalue]
                / (compprop *("," compprop) [propvalue])
                / (compparam *("," compparam) [paramvalue])

csprop          = csprop2 / all / iana-name

csprop2         = <any calendar store property defined in [CAP]>

propvalue       = propvalue2 / all / iana-name
```

```
propvalue2      = <any value appropriate for the named property>

calprop         = calprop2 / all / iana-name

calprop2        = <any calendar property name defined in [RFC2445] or
                    [CAP]>

component       = component2 / all / iana-name

component2      = <any calendar component defined in [RFC2445] or
                  [CAP]>

compprop        = compprop2 / all / iana-name

compprop2       = <any component property name defined in [RFC2445] or
                    [CAP]>

compparam       = compparm2 / all / iana-name

compparm2       = <any component parameter name defined in [RFC2445] or
                    [CAP]>

compvalue       = ";" "VALUE" "=" ((component2 *("," component2))
                / all / iana-name)

paramvalue      = paramvalue2 / all / iana-name

paramvalue2     = <any value appropriate for the named parameter>

all             = "ALL"

iana-name       = <A name registered with IANA>
```

Description: The value type is a structured value consisting of a list of one or more access control rights rule parts. Each rule part is defined by a "NAME=VALUE" pair. The rule parts are separated from each other by the SEMICOLON character (US-ASCII decimal 59). The rule parts are not ordered in any particular sequence, unless otherwise specified by the ABNF. Individual rule parts MUST only be specified once.

The UPN rule part specifies the authenticated calendar user that the calendar access rights applies to. The value of this rule part is either a quoted text specifying a UPN or an unquoted text specifying a keyword enumerating a standard authenticated user type. If the value is the keyword is ALL, then the rule applies to all authenticated calendar users (i.e., all UPNs). If the value is the keyword OWNER, then the rule applies to any of the owners of the calendar store or calendar. If the value is the keyword NONOWNER, then the rule applies to a UPN that is not the owner of the calendar store or calendar. If this rule part is not specified in the value, then the calendar access rights do not apply

to any UPN. In this case, the calendar access rights can be defined for reference by another instance of a calendar access rights. For example, a complex set of calendar access rights can be defined once and

referenced many times in the rights specified for individual calendar users.

The POLICY rule part specifies a standard calendar access policy. Calendar access policies are individual sets of well-defined calendar access rights that can be referenced by their policy name.

NOTE: Possible calendar access policy that may be standardized by CAP include:

READBUSYTIMEINFO - Specifies rights for reading busy time data.

ACTONBEHALFOF - Specifies rights for any CAP function taken on PUBLIC or PRIVATE calendar components. However, no CAP function can be taken on CONFIDENTIAL classified calendar components.

REQUESTONLY - Specifies rights for creating new event invitations, to-do assignments and journal entries.

UPDATEPARTSTATUS - Specifies rights for modifying ones own participation status.

OWNER - Specifies the same rights given to the owner of the calendar store or calendar.

The CARREF rule part specifies a reference to a particular "VCAR" calendar component. The text is matched to a CARID property value within a "VCAR" calendar component. This allows for a particular set of calendar access rights to be defined once and referenced multiple times. The "VCAR" identifier specified by this rule part is unique to the calendar store.

The ACTION rule part defines one or more CAP actions that are allowed for the UPN. The valid values are CREATE, COPY, DELETE, MODIFY, MOVE, READ, corresponding to the calendar commands; PUBLISH, REQUEST, REPLY, ADD, CANCEL, REFRESH, COUNTER, DECLINECOUNTER, corresponding to the scheduling commands; and ALL, meaning all of calendaring commands and scheduling commands. Multiple ACTION enumerations can be specified as a COMMA character (US-ASCII decimal 44) separated list of ACTION

enumerated values. The text ALL is the same as specifying the enumerated values "CREATE, MODIFY, DELETE, READ".

The OBJECT rule part defines the calendar store property, calendar property, calendar component, component property, or parameter that the ACTION is restricted to. Multiple OBJECT enumerations can be specified as a COMMA character (US-ASCII decimal 44) separated list of OBJECT enumerated values. The value ALL specifies any and all valid objects.

The VALUE rule part specifies the restricted values for the OBJECT rule part. Multiple VALUE strings can be specified as a COMMA character (US-ASCII decimal 44) separated list of VALUE strings. The text ALL specifies any and all valid values. If an OBJECT rule part is specified but no corresponding VALUE rule part is specified, then the rule applies to any and all valid values of the specified OBJECT(s).

Example: The following is a rule which specifies access rights for "foo" calendar user to read busy time values:

UPN="foo@host.com";ACTION=READ;OBJECT=DTSTART,DTEND

14.2 VCAR Calendar Component

Component Name: "VCAR"

Purpose: Provide a grouping of calendar access rights.

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

```
aclc    = "BEGIN" ":" "VCAR" CRLF
                  carprop
          "END" ":" "VCAR" CRLF

carprop = carid 1*(grant / deny)
```

Description: A "VCAR" calendar component is a grouping of calendar access rights component properties.

The "CARID" property specifies the local identifier for the "VCAR" calendar component. The "GRANT" property specifies calendar access

rights granted to an UPN. The "DENY" property specifies calendar access rights denied from an UPN.

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

```
BEGIN:VCAR
CARID:"View Start and End Times"
GRANT:UPN="foo@host.com";ACTION="READ";OBJECT=DTSTART,DTEND
END:VEVENT
```

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

```
BEGIN:VCAR
CARID:"View Start and End Times 2"
GRANT:UPN=ALL;ACTION=READ;OBJECT=DTSTART,DTEND
END:VCAR
```

In this example, rights are specified for all UPNs to read components classified as PUBLIC:

```
BEGIN:VCAR
CARID:"View PUBLIC Start and End Times"
```



```
GRANT:UPN=ALL;ACTION=READ;OBJECT=DTSTART;DTEND
DENY:UPN=ALL;ACTION=READ;OBJECT=CLASS;VALUE=PUBLIC,
 CONFIDENTIAL
END:VCAR
```

In this example, rights are specified for all UPNs to read or modify existing components classified as PUBLIC:

```
BEGIN:VCAR
CARID:"Read and Modify PUBLIC Calendar Entries"
GRANT:UPN=ALL;ACTION=READ,MODIFY;OBJECT=ALL
DENY:UPN=ALL;ACTION=READ,MODIFY;OBJECT=CLASS;VALUE=PRIVATE,
 CONFIDENTIAL
END:VCAR
```

In this example, rights are given to a standard calendar access right

policy of "viewing" (i.e., READ) busy time information:

```
BEGIN:VCAR
CARID:"View Busy Time Information"
GRANT:UPN=ALL;POLICY=READBUSYTIMEINFO
END:VCAR
```

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

```
BEGIN:VCAR
CARID:"Only OWNER or ADMIN Settable CARs"
GRANT:UPN=OWNER;ACTION=ALL;OBJECT=ALL
GRANT:UPN="cal-admin@host.com";ACTION=ALL;
 OBJECT=VCAR,CARID,GRANT,DENY
END:VCAR
```

In this example, rights to create, read, modify or delete calendar access rights are denied to all UPNs. This example would disable providing different access rights to the calendar store or calendar. This calendar access rights should not be specified, as they the ability to change calendar access; even for the owner or administrator:

```
BEGIN:VCAR
CARID:"No CAR At All"
DENY:UPN=ALL;OBJECT=VCAR,CARID,GRANT,DENY
```

14.3 GRANT Component Property

Property Name: GRANT

Purpose: This property specifies those access rights granted to a UPN.

Value Type: RIGHTS

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

Conformance: This property can only be specified in "VCAR" calendar

component.

Description: This property is used to grant calendar access rights to a UPN.

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

```
grant   = "GRANT" rightsparam ":" rights CRLF
rightparam      = *(";" xparam)
```

Example: In the following example, a hypothetical "guest@host.com" UPN is granted rights to view busy time information. These rights are specified by referencing a standard calendar access rights policy, by name:

```
   GRANT:UPN="guest@host.com";POLICY="READBUSYTIMEINFO"
```

14.4 DENY Component Property

Property Name: DENY

Purpose: This property specifies those access rights denied from a UPN.

Value Type: RIGHTS

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

Conformance: This property can only be specified in "VCAR" calendar component.

Description: This property is used to deny calendar access rights to a UPN.

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

```
DENY    = "DENY" rightsparam ":" rights CRLF
rightsparam      = *(";" xparam)
```

Example: In the following example, any UPN who is not the owner is denied rights to create, modify or delete entries:

```
   DENY:UPN=NONOWNER;ACTION=CREATE,MODIFY,DELETE;OBJECT=ALL
```

14.5 VCAR Identifier Component Property

Property Name: CARID

Purpose: This property specifies the identifier for a "VCAR" calendar component.

Value Type: TEXT

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

Conformance: This property can be specified in "VCAR" calendar component.

Description: This property permits previously defined sets of calendar access rights to be specified with a reference. This capability facilitates repetitively specifying calendar access rights.

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

```
CARID   = "CARID" textparam ":" text CRLF
```

Example: The following is an example of this property:

```
   CARID:"Restrict Guests From Creating ALARMs On Events"
```

14.6 REQUEST-STATUS property

This description is a revision of the REQUEST-STATUS property for VCALENDAR version 2.1.

```
rstatus    = "REQUEST-STATUS" rstatparam ":"
             statcode [";" statdesc [";" extdata]]

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

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

                (";" xparam)

                )

     statcode   = 1*DIGIT *("." 1*DIGIT)
```

```
     ;Hierarchical, numeric return status code

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

     extdata    = text
     ;Textual exception data. For example, the offending property
```



```
     ;name and value or complete property line.
```

Example: The following are some possible examples of this property. The COMMA and SEMICOLON separator characters in the property value are BACKSLASH character escaped because they appear in a text value.

```
     REQUEST-STATUS:2.0;Success

     REQUEST-STATUS:2.0;Success despite braindead LDAP implementation

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

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

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

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

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

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

15. CAP Entities Registration

This section provides the process for registration of new or modified CAP entities.

15.1 Registration of New and Modified CAP Entities

New CAP entities are registered by the publication of an IETF Request for Comment (RFC). Changes to a CAP entity are registered by the publication of a revision of the RFC defining the method.

15.2 Registration of New Entities

This section defines procedures by which new entities (i.e., components, properties, parameters, enumerated values or restriction tables) for a CAP entity can be registered with the IANA.

Non-standard, experimental entities can be used by bilateral agreement, provided the associated properties names follow the "X-" convention. Such non-standard entities are non-IANA entities and need not be registered using this process.

The procedures defined here are designed to allow public comment and review of new CAP entities, while posing only a small impediment to the definition of new properties.

Registration of a new CAP entity is accomplished by the following steps.

15.2.1 Define the Entity

A CAP entity is defined by completing the following template.

To: ietf-calendar@imc.org

Subject: Registration of CAP entity XXX
Entity name:
Entity purpose:
Description:
CAP terminology changes:
CAP data model changes:
CAP system model changes:
Conformance considerations:
Format definition:
Examples:

The meaning of each field in the template is as follows.

Entity name: The name of the entity.

Entity purpose: The purpose of the entity (e.g., Extends the CAP command set to poll for notifications, etc.). Give a short but clear description.

Description: Any special notes about the entity, how it is to be used, etc.

CAP terminology changes: Any change or additions to the existing CAP

terminology needs to be specified.

CAP data model changes: Any of the valid property parameters for the property needs to be specified.

CAP system model changes:

Conformance: A clear summary of how and where this CAP entity extension MUST, MAY, SHOULD or can be used. Any changes or impact to the existing conformance definition for CAP should be explained. The impact to implmentations conforming to the existing CAP specification should be clearly described.

Format definition: The ABNF for each element of the CAP entity needs to be specified.

Examples: One or more examples of instances of the CAP entity and each of its usage scenarios needs to be specified.

15.2.2 Post the entity definition

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

15.2.3 Allow a comment period

Discussion on the new entity MUST be allowed to take place on the list for a minimum of two weeks. Consensus MUST be reached on the property before proceeding to the next step.

15.2.4 Submit the entity for approval

Once the two-week comment period has elapsed, and the proposer is convinced consensus has been reached on the entity, the registration

application should be submitted to the Method Reviewer for approval. The Method Reviewer is appointed by the Application Area Directors and can either accept or reject the entity registration. An accepted registration should be passed on by the Method Reviewer to the IANA for inclusion in the official IANA method registry. The registration can be rejected for any of the following reasons. 1) Insufficient comment period; 2) Consensus not reached; 3) Technical deficiencies raised on the list or elsewhere have not been addressed. The Method Reviewer's decision to reject an entity can be appealed by the proposer to the IESG, or the objections raised can be addressed by the proposer and the entity resubmitted.

[Ed note: John Stracke to review any updates]

15.3 Property Change Control

Existing CAP entities can be changed using the same process by which they were registered.

1. Define the change
2. Post the change
3. Allow a comment period
4. Submit the entity for approval

Note that the original author or any other interested party can propose a change to an existing CAP entity, but that such changes should only be proposed when there are serious omissions or errors in the published memo. The Method Reviewer can object to a change if it is not backward compatible, but is not required to do so.

CAP entity definitions can never be deleted from the IANA registry, but entities which are no longer believed to be useful can be declared OBSOLETE by adding this text to their "Entity purpose" field.

16. IANA Considerations

This memo defines IANA registered extensions to the attributes defined by iCalendar, as defined in [RFC2445], and iTIP, as defined in [RFC2426].

IANA registration proposals for iCalendar and iTIP are to be emailed to the registration agent for the "text/calendar" MIME content-type, <MAILTO: ietf-calendar@imc.org> using the format defined in section 7 of [RFC2445].

17. Acknowledgments

The following have individuals were major contributors in the drafting and discussion of this memo:

Mario Bonin, Andre Courtemanche, Dave Crocker, Pat Egen, Gilles Fortin, Alex Hoppman, Bruce Kahn, Lisa Lippert, David Madeo, Bob Mahoney, Pete O'Leary, Richard Shusterman, Tony Small, John Stracke.

18. Bibliography

[RFC1521] N. Borenstein and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Internet Draft UTF-825 July 1996 Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993.

[TLS] Dierks, Allen, "The TLS Protocol", RFC 2246, January 1999

[RFC2396] Berners-Lee, Fielding, Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2445] Dawson, Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998

[RFC2446] Silverberg, Mansour, Dawson, Hopson, "iCalendar Transport-Independent Interoperability Protocol (iTIP)", RFC 2446, November 1998

[RFC2447] Dawson, Mansour, Silverberg, "iCalendar Message-Based Interoperability Protocol (iMIP)", RFC 2445, November 1998

[SQL] "Database Language _ SQL", ANSI/ISO/IEC 9075: 1992, aka ANSI X3.135-1992, aka FiPS PUB 127-2

[SQLCOM] ANSI/ISO/IEC 9075:1992/TC-1-1995, Technical corrigendum 1 to ISO/IEC 9075: 1992, also adopted as Amendment 1 to ANSI X3.135.1992

[UNICODE] The Unicode Consortium, "The Unicode Standard --Worldwide Character Encoding -- Version 1.0", Addison-Wesley, Volume 1, 1991, Volume 2, 1992. UTF-8 is described in Unicode Technical Report #4.

[US-ASCII] Coded Character Set--7-bit American Standard Code for Information Interchange, ANSI X3.4-1986.

19. Author's Address

The following address information is provided in a vCard v3.0, the RFC **2426 electronic business card format.**

```
BEGIN:VCARD
VERSION:3.0
N:Dawson;Frank
FN:Frank Dawson
ORG:Lotus Development Corporation
ADR;TYPE=WORK,POSTAL,PARCEL:;;6544 Battleford Drive;Raleigh;NC;
 27613-3502;US
TEL;TYPE=PREF,WORK,MSG:+1-617-693-8728
TEL;TYPE=WORK,MSG:+1-919-676-9515
TEL;TYPE=WORK,FAX:+1-919-676-9515
```

```
   EMAIL;TYPE=INTERNET,PREF:Frank_Dawson@Lotus.com
   EMAIL;TYPE=INTERNET:fdawson@earthlink.net
   URL;TYPE=X-HOME:http://home.earthlink.net/~fdawson
   END:VCARD

   BEGIN:VCARD
   VERSION:3.0
```



```
   N:Mansour;Steve
   FN:Steve Mansour
   ORG:Netscape
   ADR;TYPE=WORK,POSTAL,PARCEL:;;501 E Middlfield Road;Mountain
    View;CA;94043;US
   TEL;WORK;MSG:+1-650-937-2378
   TEL;WORK;FAX:+1-650-937-2103
   EMAIL;INTERNET:sman@netscape.com
   END:VCARD

   BEGIN:VCARD
   VERSION:3.0
   FN:Doug Royer
   N:Royer;Doug
   ORG:Sun Microsystems
   ADR;TYPE=WORK,POSTAL,PARCEL:MS MPK17-105;;901 San Antonio Road;
    Palo Alto;CA;94303-4900
   TEL;TYPE=WORK,VOICE:650-786-7599
   TEL;TYPE=FAX:650-786-7994
   EMAIL;TYPE=INTERNET:doug.royer@sun.com
   END:VCARD

   BEGIN:VCARD
   VERSION:3.0
   FN:Alexander Taler
   N:Taler;Alexander
   ORG:CS&T
   ADR;TYPE=WORK,POSTAL,PARCEL:;;3333 Graham Boulevard;Montreal;QC;
    H3R 3L5;Canada
   TEL;TYPE=WORK,VOICE:514-733-8500
   TEL;TYPE=FAX:514-733-8878
   EMAIL;TYPE=INTERNET:alext@cst.ca
   END:VCARD
```

20. Full Copyright Statement