Calendaring Interoperability Protocol

Status of this Memo

Abstract

The Calendaring Interoperability Protocol (CIP) provides a standard mechanism for exchanging events and other information between scheduling systems. This allows users to schedule meetings with anyone else, no matter what scheduling software they use.

CIP is one of three standards under development by the Calendaring and Scheduling Working Group of the IETF. The other two are the Core Object Specification (COS), a standard data format for representing scheduled events, and the Calendaring Access Protocol (CAP), a standard mechanism that scheduling user agents may use to access user calendars (analogous to POP).

Table of Contents

Status of this Memo 1
Abstract 1
Table of Contents 1
1.0 Protocol Overview 2
1.1 Design Goals 3

1.2 Terminology 4
1.3 Event Object Format 4
1.4 Commands and Responses 5
1.5 Server Identifiers 7
1.6 User Identifiers 7
1.7 Event Identifiers 8
1.8 Event Versioning 8
1.9 Contacting Servers 9
2.0 Protocol Commands 9
2.1 AUTHENTICATE Command 9
2.2 SENDEVENT Command 10
2.3 RSVP Command 11
2.4 FREEBUSY Command 12
3.0 Example Session 13
4.0 Formal Grammar 16
5.0 Security Considerations 16
6.0 References 16
7.0 Open Issues 17
8.0 Author's Addresses 18

1.0 Protocol Overview

The Calendaring Interoperability Protocol (CIP) provides a standard mechanism for exchanging events and other information between scheduling systems. This allows users to schedule meetings with anyone else, no matter what scheduling software they use.

There are many kinds of scheduling systems, ranging from a centralized database serving thousands of users to a single-user Personal Information Manager (PIM) or a handheld device. All of these systems should be able to use CIP successfully.

Here is a simple example with many details omitted:

Tom Jones of Jumbo Corp. wants to invite Mary Weaver of Tiny, Inc. to a meeting. He uses his scheduling software to create a meeting and adds Mary to the list of guests, using her CIP user id (probably the same as her email address, say mary@tiny.com). Tom's scheduling system opens a session with the CIP server for tiny.com and sends the event. Mary's scheduling system delivers

the event to her and she accepts the invitation. Mary's scheduling system contacts the CIP server for jumbo.com and sends a response indicating that she plans to attend. Tom's scheduling system communicates this to him.

Several key concepts are illustrated by this example. CIP is used only for interoperability between the scheduling systems, not for

calendar access when Tom or Mary are accessing their personal calendars. Each scheduling system functions both as a client and as a server. As a server, it has a server id that is a domain name (but not necessarily the DNS host name of the server). Each user has a user id that is often the same as their email address. Users send and receive invitations with their normal scheduling software. Their scheduling system is responsible for communicating with other scheduling systems.

Each CIP server has certain responsibilities. First, it MUST have a unique server identifier (described in more detail in a later section). Second, it SHOULD have several user identifiers that include that server identifier. All CIP traffic destined for those user identifiers will be sent to that CIP server. Third, it SHOULD be available to CIP traffic most of the time. CIP depends on establishing a TCP connection to the CIP server. If this connection cannot be made, CIP traffic will be halted. Fourth, it SHOULD have several.

Each CIP client has certain responsibilities. First, it MUST contact CIP servers and conduct the CIP session according to the protocol described here. Second, if it cannot contact a CIP server, it MAY attempt the connection later using an exponential backoff algorithm.

All characters sent as part of the CIP protocol are members of the character set ISO 8859-1. The COS makes provisions for using other character sets for event properties, such as event descriptions, and
encoding them so they can be transmitted using ISO 8859-1. Therefore, the only items that are substantially affected by this character set restriction are those that appear outside of the COS: the server
identifier and user identifier.

1.1 Design Goals

The primary goal of CIP is quite simple: to allow people to schedule events with each other without worrying about incompatibilities between their scheduling systems. All other goals are secondary to that one.

CIP is designed to be simple. With only four commands and a very simple syntax, it should be easy to implement. Most of the effort will be in parsing and generating the COS format for representing events.

CIP is designed to be secure. The AUTHENTICATE command

allows for mutual authentication between client and server and encryption of the subsequent protocol exchanges.

CIP is designed to be international. With the exception of server and user identifiers (which are restricted by older RFCs), all data items are contained within the COS and can therefore be expressed with almost any character set.

1.2 Terminology

This document uses the following terms:

event: An event is a data item that refers to a specific date and/or time. For the purposes of this document, the term event also refers to todos.

user: For the purposes of this specification, a user is a person, location, or resource whose calendar is managed by a scheduling system.

scheduling system: A scheduling system is a software program that manages the scheduling of events. There are many types of scheduling systems, from a centralized database serving thousands of users to a single-user Personal Information Manager (PIM) or a handheld device. All of them should be able to function as CIP servers or clients.

CIP client: A CIP client is a scheduling system using CIP to send commands to a CIP server.

CIP server: A CIP server is a scheduling system that accepts

commands from CIP clients.

1.3 Event Object Format

CIP relies on the existence of a text-based representation of calendar information, which is not described in this document. The examples below employ vCalendar, a calendar information format authored by the Versit consortium (see [1]).

CIP does depend on certain information being present in the data being transported. Some properties that are defined as "optional" in the specification of vCalendar are mandatory when transported via CIP:

ATTENDEE - CIP uses this property to determine the intended recipient of SENDEVENT
ATTENDEE; ROLE = OWNER: this property MUST be present so that event recipients can locate the originator before using the RSVP command. The user's address given in the OWNER attendee MUST be the calendar host name that attendees can RSVP to.

SEQUENCE - CIP uses this property for event versioning, as described below.
UID - CIP relies on this property to locate events when an RSVP command is issued The originator of the meeting generates the UID and MUST send it along with the first SENDEVENT command. The UID also MUST be sent along with the RSVP command.

1.4 Commands and Responses

The CIP session begins when the client initiates a connection with a CIP server on port IANA registered port xxx. The server responds to the client connection with the following statement containing the name of the protocol - "CIP" - a version number and the DNS name of the server:

S: OK CIP 1.0 hostname

The version number given is comprised of two numeric values

delimited by a period ".". The first number given is the "major" revision number and the second is the "minor" revision number. If the client initiating the protocol session receives a major revision number other than what it expects, the client should close the current session immediately. Major revisions of CIP are not guaranteed to be compatible with any other revision. A client which encounters a minor revision number from the server which is lower than that supported by the client may choose to either close the connection or initiate commands defined in the lower-numbered version of CIP only. Higher-numbered minor revisions are not a problem for the client, the server must support all commands and responses defined by the lower revision.

The client drives the session by sending commands to the server. All commands are terminated with a carriage return and line feed pair (CRLF). No command is considered complete by the server until the CRLF is received. Certain command require sending additional data after to command itself has been sent. All such commands finish with a plus character "+" as the last character on the line before the CRLF. This convention makes it easy for implementers to determine when additional data is to be received. The server should send a single "+" character on a line followed by CRLF before it returns any data back to the client.

Data is sent one line at a time, each line followed by CRLF. The end of the data is marked by a single period character "." on a line followed by a CRLF. If a single period on a line must be sent as part of the data, two periods can be sent in a row.

The client must wait for the last command issued to complete, as described below, before issuing a new command.

There are two types of responses which may be given by the server after the client has initiated a command: informational responses and command completion responses.

Informational responses are of the form:

"*" information_code information

All informational responses begin with "*" as the first character

on the line, followed by an information code. The server may send as many informational responses as necessary. The exact content of the informational response depends on the command currently in progress. See the description of the various command below for more information.

Command completion responses are of the form:

[OK | NO] result_code text_information

The first part of the command completion response indicates the overall result of the command:

OK - The command completed successfully.
NO - The command failed and no action was taken.

Note: These result codes are intended to convey the overall results of the command and are the only part of the command completion response which a client must rely on. The remainder of the command completion response is for informational purposes only!

The result_code is a three-digit numerical value which gives additional information about why a command completed the way that it did.

The text_information part of the response can consist of any information that the server wants to send. This should be a human-readable description of the command result; the client may elect to display this information to the user in some way.

Result Codes:

000 No additional information given.
001 Command was invalid or parameters are bad: given along with NO response

002 Operation not allowed - access control.
003 Command completed partially - see specific command for more details

Other result codes are defined below under their respective

commands.

1.5 Server Identifiers

Each CIP server is identified by a server identifier, which is a domain name (as defined by RFC 1034 [4]). This name may be used to contact the CIP server, as described in section 1.9, Contacting Servers. Each server identifier MUST be used by only one server.

If a server changes its server identifier, it is a good idea to have a transition period during which both the old and the new server identifiers are supported. Before the old server identifier is retired, all events whose event identifier includes the old server identifier SHOULD be cancelled and sent out with an event identifier that includes the new server identifier.

1.6 User Identifiers

Each CIP user is identified by a user identifier of the form local-user-id@server-id. Server-id is a server identifier, as described in section 1.4. Local-user-id is a local identifier specific to that server.

The syntax for user identifiers is very similar to the email address syntax defined in RFC 822 [2], but somewhat simplified. For many users, their CIP user identifier will be the same as their Internet email address, which will be very convenient. Of course, they are used quite differently. Instead of using SMTP to send email, scheduling systems will use CIP to send and manage events.

If a user changes their user identifier (either the local-user-id or the server-id), their old server SHOULD reserve the old local-user-id for a period of time and provide REDIRECT informational responses whenever it receives a SENDEVENT or FREEBUSY command that refers to the old user identifier. This will allow CIP clients to update their databases.

CIP does not provide any inherent support for groups. A scheduling system can provide a user identifier that refers to a group, but CIP doesn't provide any special support for it. If scheduling systems do provide such support, they SHOULD

avoid loops. If a user identifier on their system is mapped to a user identifier on another system, which is mapped back to the original user identifier on their system, a loop is formed. There are more complicated versions, too.

1.7 Event Identifiers

In order to manage events easily, each event that is sent or referred to with CIP has a unique event identifier of the form local-event-id@server-id. Server-id is the server identifier of the CIP server that owns the event. Local-event-id is a local identifier assigned by that server.

Each event sent and referred to with CIP MUST have one and only one event identifier. This event identifier SHOULD NOT change, even if the event changes (including changes to event title, owner, and even event cancellation.). It is permissible for a CIP server to cancel one event and create a new one with a different event identifier. This may be useful if, for instance, an event should now be owned by a different server. As far as CIP is concerned, this is really a different event and all CIP clients will regard it as such.

A single event identifier SHOULD NOT be used for different events, even if one event is cancelled and another created. Otherwise, CIP clients may become confused about which event is which.

Event identifiers are used in several places. In the SENDEVENT command, they are stored in the UID property within the vCalendar representation of the event. In the RESPOND command, one of the parameters is the event identifier of the event being responded to.

1.8 Event Versioning

In order to manage multiple versions of events, each CIP server must assign an event version to each event it owns. This version number is a monotonically increasing positive integer, starting at 1 when the event is created. Whenever the server changes an event, it increments the event's version. CIP clients and servers SHOULD use the event version to check version compatibility, as described in the descriptions of the SENDEVENT and RESPOND commands.

If the version reaches 2^31 (two to the thirty-first power), the event SHOULD be cancelled and a new event created. This avoids problems caused by wraparound. Of course, this should be

very rare.

1.9 Contacting Servers

In order to establish a connection with a CIP server, a CIP client must map a server identifier to an IP address. Instead of performing a simple DNS host name resolution, a CIP client MUST use the technique described in RFC 2052 (DNS SRV) [3] to locate a server with service equal to CIP, protocol equal to TCP, and name equal to the server identifier. If no SRV record is found, this technique backs off to a simple DNS host name resolution of the server identifier. Once an IP address has been found, a TCP connection MUST be opened to a port number that will be assigned later by the IANA. This mapping process MUST be followed every time a CIP client tries to connect to a CIP server.

The primary advantage of this technique is that it allows server identifiers to be quite simple and allows a user's CIP user identifier to often be the same as their email address. This means the user has one less item to remember and write down on their business cards.

2.0 Protocol Commands

In the following command descriptions, "C:" indicates a command line initiated by the client. "S:" indicates a response from the server.

2.1 AUTHENTICATE Command

C: AUTHENTICATE authenticate_code authenticate_data

C: AUTHENTICATE LOGIN identity password

The AUTHENTICATE command can be used by the protocol client to establish it's identity for purposes of access control. It is not required for a client to login or otherwise authenticate itself in order to initiate commands. Servers should enforce whatever access control policies are appropriate for their application. Note that the use of the term "client" here is not meant to imply that the protocol client is associated with a particular user: the client

may, in fact, be a server.

The AUTHENTICATE command takes a code as its first parameter which indicates the type of authentication mechanism to use. The simplest AUTHENTICATE code is LOGIN which takes a user name and plain-text password as its parameters

followed by CRLF. Additional AUTHENTICATE codes are specified in xxx.

Responses:

Command completion response of OK or NO only. For security purposes, it is not a good idea for the server to return any additional information for a failed login.

Example:

```
C: AUTHENTICATE LOGIN fredsbaking.com fromblotz
S: OK 000 Login complete.

C: AUTHENTICATE LOGIN clearblue.com foobatz
S: NO 000 Access denied.
```

2.2 SENDEVENT Command

```
C: SENDEVENT "+" CRLF
C: event_information
C: "." CRLF
```

The SENDEVENT command is used by the client to pass a new event to the server. The client sends the command, followed by a "+" and CRLF. The client then sends the event data, one line at a time, ending with a single period character on a line. The server then replies with 0 or more informational responses followed by the command completion response.

Responses:

NOTFOUND - This informational response should be followed by a user name. The given user was not found on the current server and the SENDEVENT request could not be completed.

TENTATIVE - This informational response should be followed
by a user name. The event has been stored for the given user but
they have not yet indicated whether they plan to attend the event.
See the RSVP command below for a description of how a user
would respond to the event.

PERMDENIED - This response is given with a user name when
the SENDEVENT failed due to access control restrictions
imposed by an attendee.

REDIRECT - This informational response informs the client that
a user identifier has changed. It only applies to user identifiers

that are listed as an ATTENDEE in the event and whose server
identifier maps to this CIP server. This response is equivalent to
the NOTFOUND response, but provides more information. The
client SHOULD resend the event to the new user identifier. The
client MAY update its database to always use the new user
identifier instead of the old.

* REDIRECT USER old-id IS NOW new-id CRLF

Completion Codes:

003 Not all attendees received the event

Example:

```
C: SENDEVENT +
C: DTSTART: 19961231T2000-0500
C: DTEND: 19970101T0600-0500
C: DESCRIPTION: New year's eve party in New York. The Big
2000!
C: ATTENDEE; ROLE=OWNER: Pete O'Leary
<poleary@clearblue.com>
C: ATTENDEE: Bill Clinton <president@whitehouse.gov>
C: ATTENDEE: Jen Yip <jen@yip.com>
C: SEQUENCE: 1
C: UID: 23454321abcdabcd@clearblue.com
C: .
S: * NOTFOUND <president@whitehouse.gov>
```

S: NO 000 Not all users are present on this server.

2.3 RSVP Command

C: RSVP "+" CRLF
C: event_information
C: "." CRLF

The RSVP command is used by the client to update the status of an event. The command is normally used by the recipient (via SENDEVENT) of an event to indicate to the event's originator whether the recipient plans to attend the event in question.

The client sends the command, followed by a "+" and CRLF. The client then sends the event data, one line at a time, ending with a single period character on a line. The server then replies with 0 or more informational responses followed by the command completion response.

The client is not required to send all of the information which

constitutes a full event, only the information to be updated in the original event. The server may choose to ignore any information in the event which cannot be modified by the client, based on access control restrictions.

Responses:

CONFIRMED - Can be sent by the server to indicate that other attendees have confirmed the meeting it was last updated or received by the attendee performing the RSVP.

REDIRECT - This informational response informs the client that an event identifier has changed. It only applies to the event id listed in the RSVP command. This response is equivalent to the NOTFOUND response, but provides more information. The client MAY resend the RSVP to the new event identifier. The client MAY update its database to always use the new event identifier instead of the old.

* REDIRECT EVENT old-id IS NOW new-id CRLF

Example:

```
C: RSVP +
C: DTSTART: 19961231T2000-0500
C: DTEND: 19970101T0600-0500
C: DESCRIPTION: New year's eve party in New York. The Big
2000!>
C: ATTENDEE; STATUS=ACCEPTED: Jen Yip
<jen@yip.com>
C: .
S: OK 000 RSVP accepted.
```

2.4 FREEBUSY Command

```
C: FREEBUSY "+" CRLF
C: <busy_request> CRLF
C: "." CRLF
S: "+" CRLF
S: <response_information> CRLF
S: "." CRLF
```

The FREEBUSY command requests that the server gather free or busy time information according to the request information submitted by the client. The client sends the command, followed by a "+" and CRLF. The client then sends the request data, one line at a time, ending with a single period character on a line. The server then replies with the response data, then 0 or more

informational responses followed by the command completion response. The server may choose to send informational responses and a completion response only if the free/busy request cannot be completed for any reason. The client can distinguish between the different responses based on the first character on the line received: "+" for data, "*" for informational responses.

Response:

NOTFOUND - This informational response should be followed by a user name. The given user was not found on the current server and no free/busy information for the user was returned.

REDIRECT - This informational response informs the client that

a user identifier has changed. It only applies to user identifiers that are listed in the free_busy_request and whose server identifier maps to this CIP server. This response is equivalent to the NOTFOUND response, but provides more information. The client MAY resend the FREEBUSY request to the new user identifier. The client MAY update its database to always use the new user identifier instead of the old.

* REDIRECT USER old-id IS NOW new-id CRLF

Example:

C: FREEBUSY +
S: +
C: fill in example here

3.0 Example Session

Scenario:
A group meeting is setup in C&S system A that contains one attendee, Bob, in system B. System A is represented by a CIP server at a.com and system B is represented by a CIP server at b.com.

CIP server a.com does a DNS lookup to find the IP address of b.com and opens a TCP connection on port CIP_PORT

S: OK CIP 1.0 b.com
C: AUTHENTICATE LOGIN a.com PASSWORD_A
S: OK 000 LOGIN COMPLETED
C: SENDEVENT+
C: UID: unique identifier for event
C: SEQUENCE: 1
C: DTSTART: start date and time

C: DTEND: end date and time
C: DESCRIPTION: Project X review meeting
C: ATTENDEE; ROLE=OWNER: Pete O'Leary
<poleary@a.com>
C: ATTENDEE: Alex SystemA <alex@a.com>
C: ATTENDEE: Bob SystemB <bob@b.com>
C: .

```
S: OK 000 SENDEVENT COMPLETED

CSIP server a.com closes connection

Scenario:
Bob on system B accepts the meeting invitation.

CIP server b.com does a DNS lookup to find the IP address of
a.com and opens a TCP connection on port CIP_PORT

S: OK CIP 1.0 a.com
C: AUTHENTICATE LOGIN b.com PASSWORD_B
S: OK 000 LOGIN COMPLETED
C: RSVP+
C: UID unique identifier of event
C: SEQUENCE: 1
C: RESPONSE-SEQUENCE: 1
C: ATTENDEE: Bob SystemB <bob@b.com>
C: STATUS:  CONFIRMED
C: .
S: OK 000 RSVP COMPLETED

CIP server b.com closes connection

The scenario is a simple meeting between two people.  Bill
schedules it from
his PC client.  To test out some error handling, it fails the first
time
under anonymous authentication.

Client (bill's PC):

Connects to abc.com

C: SENDEVENT +
C: CATEGORIES: Meeting
C: DTSTART: 19961126T1530-0500
C: DTEND:  19961126T1630-0500
C: SUBJECT: Important Meeting
C: DESCRIPTION: We need to talk about important stuff.
C: CLASS: Private
```

```
C: UID: 0289AF8G7DD@abc.com
C: SEQUENCE: 1
C:
ATTENDEE;ROLE=OWNER;STATUS=CONFIRMED:bill@ab
c.com
C:
ATTENDEE;ROLE=ATTENDEE;STATUS=TENTATIVE:mary
@xyz.com
C: .
S: * PERMDENIED bill@abc.com
S: * NOTFOUND  mary@xyz.com
S: NO 000  Quite a bummer, this failed altogether.

C: AUTHENTICATE LOGIN bill aardvark
S: OK 000
C: SENDEVENT +
C: CATEGORIES: Meeting
C: DTSTART: 19961126T1530-0500
C: DTEND:  19961126T1630-0500
C: SUBJECT: Important Meeting
C: DESCRIPTION: We need to talk about important stuff.
C: CLASS: Private
C: UID: 0289AF8G7DD@abc.com
C: SEQUENCE: 1
C:
ATTENDEE;ROLE=OWNER;STATUS=CONFIRMED:bill@ab
c.com
C:
ATTENDEE;ROLE=ATTENDEE;STATUS=TENTATIVE:mary
@xyz.com
C: .
S: * NOTFOUND  mary@xyz.com
S: OK 000  Things came out better this time

Disconnect from abc.com
Connect to xyz.com

C: SENDEVENT +
C: UID: 0289AF8G7DD@abc.com
C: SEQUENCE: 1
C: CATEGORIES: Meeting
C: DTSTART: 19961126T1530-0500
C: DTEND:  19961126T1630-0500
C: SUBJECT: Important Meeting
C: DESCRIPTION: We need to talk about important stuff.
C: CLASS: Private
C:
ATTENDEE;ROLE=OWNER;STATUS=CONFIRMED:bill@ab
```

```
c.com
C:
ATTENDEE;ROLE=ATTENDEE;STATUS=TENTATIVE:mary
@xyz.com
C: .
S: * TENTATIVE mary@xyz.com
S: * NOTFOUND  bill@abc.com
S: OK 000 Stored something anyway.
```

Now, from Mary's PC. Somehow, she's seen the event...

Mary connects to abc.com.

```
C: RSVP +
C: UID: 0289AF8G7DD@abc.com
C: SEQUENCE: 1
C:
ATTENDEE;ROLE=ATTENDEE;STATUS=CONFIRMED:mar
y@xyz.com
S: * CONFIRMED mary@xyz.com
S: * CONFIRMED bill@abc.com
S: OK 000 Stored all we could.
```

Mary disconnects. Note that this did not require authentication.

4.0 Formal Grammar

Insert lots of BNF here.

5.0 Security Considerations

CIP protocol transactions, including event data, are sent in the clear over the network unless encryption is negotiated using the AUTHENTICATE command.

A server response for a command that fails due to insufficient permissions should not detail why the permissions are insufficient.

Additional security considerations are discussed in the section discussing the AUTHENTICATE command.

The LOGIN authentication mechanism described in this document sends a plaintext password. In the absence of

transport-level encryption, this creates to possibility that the password can be intercepted enroute.

6.0 References

[1] versit Consortium, "Electronic Calendaring and Scheduling (vCalendar) Specification", 03/29/1996, http://www.versit.com/

[2] D. Crocker, "Standard for the format of ARPA Internet text messages", 08/13/1982, http://ds.internic.net/rfc/rfc822.txt

[3] A. Gulbrandsen, P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", 10/31/1996, http://ds.internic.net/rfc/rfc2052.txt

[4] P. Mockapetris, "Domain names - concepts and facilities", 11/01/1987, http://ds.internic.net/rfc/rfc1034.txt

7.0 Open Issues

Should have a section on Requirements that defines the meanings of MUST, MUST NOT, SHOULD, and SHOULD NOT.

Should move description of C: and S: to a section early on. At the moment, they appear before they're explained.

Should talk about gateways.

Need to make sure that AUTHENTICATE allows both client and server to establish the identity of the other.

Should give more info about loops and how to avoid them.

Should update numeric result codes so they're more consistent.

Can one event go by several ids?

Should use references throughout.

If we add the SELECT SERVER command, add a comment to the Server Identifiers section that tells people to use it and says

they should provides redirects there when a server identifier changes.

Need to coordinate with e-mail (MIME) based calendar interoperability protocol. Need to coordinate with Common Object Specification.

Should talk about email delivery as an alternative transmission mode.

Should provide some examples of how a couple of different kinds

of scheduling systems would implement this (perhaps in a separate section). Maybe we should just implement it to provide a good example!

Should say that each event is owned by a single server.

8.0 Author's Addresses

Anik Ganguly
Campbell Services Inc.
anik@ontime.com
(810) 559-5955 x4646

Steve Hanna
On Technology Corporation
hanna@world.std.com
(617) 692-3153

Peter O'Leary
Clear Blue Network Systems, Inc.
poleary@clearblue.com
(415) 323-3380

Bill Spencer
Phase2 Software Corporation
bill@p2software.com
(508) 481-8496 x41