Oracle® Calendar API

Developer's Guide

Release 2.5

August, 2002

Part No. B10097-01

This guide contains considerations and reference material for the use of the Oracle Calendar API.

ORACLE®

Oracle Calendar API Developer's Guide, Release 2.5

Part No. B10097-01

Contents

2 Function Reference

Send Us Your Comments

Oracle Calendar API Developer's Guide, Release 2.5

Part No. B10097-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 633-3836 Attn: Oracle Collaboration Suite Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Collaboration Suite Documentation Manager
  500 Oracle Parkway, Mailstop 2op5
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual contains considerations and reference material for the use of the Oracle Calendar API, a collection of C-language function calls that provides access to Oracle's calendar server.

Intended Audience

This manual is intended for any programmers and developers who intend to use the Oracle Calendar API to create custom applications for calendar access.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

This manual contains five chapters:

Chapter 1

This chapter contains an overview of various elements of the Oracle Calendar API, as well as items to consider before implementation.

Chapter 2

This chapter contains detailed documentation on the functions provided with this development kit.

Chapter 3

This chapter contains information on the configuration parameters that can be supplied to the Oracle Calendar API.

Chapter 4

This chapter contains a variety of information on type definitions, constants and capabilities.

Chapter 5

This chapter contains an alphabetical list and brief explanation of every status code CAPI provides as feedback.

Related Documents

For more information, see the following manuals in the Oracle Collaboration Suite documentation set:

- *Oracle Calendar API Release Notes*
- *Oracle Calendar Server Administrator's Guide*

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1

Implementation Considerations

This chapter discusses a variety of factors to be taken into consideration in your Oracle Calendar API (CAPI) implementations:

- Character Sets
- iCalendar Support
- Security Model
- Alarms
- Event Recurrences
- User identification
- Data Streams
- Access Control

Character Sets

CAPI only supports text encoded in UTF-8. Strings passed to CAPI_Logon and CAPI_GetHandle must be in UTF-8. All output data is in UTF-8. Input must be in UTF-8; this means that MIME entities which specify a character set must specify UTF-8 or US-ASCII (MIME defaults to US-ASCII if no character set is specified).

iCalendar Support

CAPI uses the iCalendar format (as specified in RFC 2445) for dealing with calendar data. iCalendar information saved via CAPI can be retrieved later. However, not all iCalendar data is actively supported by this revision of CAPI. In particular, VTODO, VFREEBUSY and VJOURNAL components are not supported.

iCalendar input

The essential iCalendar data is mapped to native data structures. Data for these properties will not always be completely preserved. Some properties are stored only per event, rather than per instance, so only one value is preserved. The following are affected:

DTSTART, DTEND and DURATION

If DTEND is present, it will be used to calculate the event duration; the actual end time is not stored. As event times are measured in minutes, the start time and duration will have their 'seconds' component set to zero.

SUMMARY

This property is mapped to the event title. When using a 5.0 calendar server this value will be truncated to 64 bytes.

PRIORITY

This property is mapped to one of the calendar server's 5 priority values. This property is stored per event.

CLASS

This property is mapped to access level. The mapping between iCalendar and the calendar server's access levels is as follows:

From iCalendar to calendar server:

- PUBLIC->PUBLIC
- PRIVATE->PERSONAL
- CONFIDENTIAL->CONFIDENTIAL

From calendar server to iCalendar:

- PUBLIC->PUBLIC
- PERSONAL->PRIVATE
- X-CST-NORMAL -> NORMAL
- NORMAL->PRIVATE
- CONFIDENTIAL->CONFIDENTIAL

This property is stored per event.

STATUS

A value of TENTATIVE indicates a tentative event. Any other value (even CANCELLED) will indicate a non-tentative event.

LOCATION

This is stored in the location field. When using a 5.0 calendar server, this value will be truncated to 32 bytes.

ATTENDEE

When storing "ATTENDEE" properties an attempt will be made to correlate attendee properties with the people to whom they refer by comparing them with the supplied handles. If an attendee property represents a calendar user and there is no corresponding handle for that user then the calendar user will not be invited to the event.

ATTACH

Currently this property is ignored.

DESCRIPTION

This is set to the Event's details. It will be truncated if it is longer than 32 Kb. This property is stored per event.

CATEGORIES

This is mapped to the event's type. Possible values are "APPOINTMENT", "DAILY NOTE", "DAY EVENT", and "HOLIDAY". The event type is stored per event.

VALARM

Alarms are preserved separately for each attendee of an event. Users will not be able to see each other's alarms.

UID

If a UID is not specified in stored data the server will assign a UID. These assigned UIDs are accessible through the CAPI_GetLastStoredUIDs function described below.

Other iCalendar data will be preserved and returned unchanged.

iCalendar output

When data created with Oracle calendar clients is output, the following is a list of available properties, and how they are obtained. Other properties stored with CAPI are also available.

Instances of an event are all returned in separate VEVENT objects.

ORGANIZER

The event owner.

CLASS

The event access level.

PRIORITY

The event priority.

STATUS

A tentative event will have a TENTATIVE status. Non-tentative events will be marked as CONFIRMED. No other STATUS values are generated.

DESCRIPTION

The event details.

CATEGORIES

The event type.

DURATION or DTEND

The duration of the event.

SUMMARY

The event title.

DTSTART

The start time of the event, in GMT.

UID

An id generated from internal ids.

ATTENDEE

A property is generated for each ATTENDEE. The parameters PARTSTAT, ROLE, CUTYPE, and CN are obtained from the attendee and user information.

RESOURCES

Converted from native resource attendees.

VALARM

Converted from native reminders.

Security Model

There are two parts to the security model: storing and fetching events. These are handled by different security paradigms.

The owner of an event can add or delete any property of that event. When an "ATTENDEE" property is created for a calendar user, and a handle has been supplied for that user, the property is created with default values for its parameters. The owner of the event cannot modify the parameters of that property, only the user to whom it corresponds can do that.

When a user is updating their "ATTENDEE" property no error will be returned if there is an attempt to modify other event data, but the modifications will not occur.

It is possible for a user to refuse invitations from another user. In that case an "ATTENDEE" property will not be created for that user and the status for that user's handle will indicate that the invitation was refused. This may also occur when attempting to double book resources.

When fetching events the security model is based on the iCalendar classification of the event. Users grant other users different access levels to different classes of events. The three access levels are: no read access, read the start and end times of the event only, and read all details of the event. When fetching events with CAPI this results in some events for which only the "DTSTART", "DURATION" and "DTEND" properties will be returned. All other events will be invisible or all of their properties will be returned.

CAPI does not allow users to modify the security records which govern this behaviour.

Alarms

Alarms are considered private to each user, so users cannot read or write alarms for each other. Since users cannot read each others alarms it is not possible for users to do fetches by alarm range on each other's calendars. Any user may set an alarm for an event which they are attending, so the same events can have a different alarm when fetched by a different user.

Alarms will be returned by a fetch by id on the current user's agenda, and in fetches by alarm range. The alarm always applies to the logged in user.

Event Recurrences

All event recurrences are expanded into separate recurrence dates. This affects events which are stored with "RRULE" properties. Once an event is stored any "RRULE" and "EXRULE" properties will be lost.

Format of returned iCalendar

When fetching events, iCalendar data is returned with a separate "VEVENT" component for each instance of a recurring event. This may result in a lot of repeated data for events with many instances. This behaviour may change in the future.

User identification

Users are identified to CAPI by a userid string, or by using a search string specifying, for example, the user's name. The string format is flexible and allows the caller to specify a number of optional parameters. Depending on the server configuration, some of these options (such as the Node ID) may be required in the identification string. The same user identification string format is used both at logon and when obtaining a handle, however not all options will be applicable in both cases.

Logging into the server as a resource is not supported.

All options are specified using key-value pairs. The entire string is a collection of such pairs. The userid is separated from the extended data by an ASCII '?' character. The character immediately following this one is the delimiter of each subsequent

field value pair. The delimiter may be any ASCII character except a digit, a letter, NUL, '*' or '='. The rest of the string consists of field-value pairs separated by the chosen delimiter. The string is terminated by a delimiter followed by a NUL character. Field value pairs consist of a field name, followed by an equal sign ('='), followed by the value. The value is a string which does not contain the delimiter character, the NUL character, and for user identification strings, the slash ('/') (aka solidus) character.

For example, the field name G denotes the given name, and S denotes the surname. The following is a sample legal string for identifying a user. No userid is specified, so the optional parameters would used to search for the user. (Note that if a search results in multiple matches, CAPI will return an error to the caller; a userid is the best method of specifying a user, if it is available.) Even with no userid, we still have the question mark '?' character separating the userid from the extended string. The character immediately following, in this case a slash '/', is used as the delimiter. Note that the string ends with the delimiter character, and is NUL terminated.

```
?/S=Bunny/G=Bugs/
```

Any field used for identifying a user may be terminated with a '*', which is used as a wildcard. This is not available for specifying nodes. The following will also match the above user:

```
?/S=Bu*/G=Bugs/
```

Remember that if multiple users match a given search string, the CAPI call will return an error.

Resources have a different name structure; they are identified with the single field RS which indicates resource name.

The following grammar (in ABNF form, as described in RFC 2234) describes legal logon strings:

The description below diverges from ABNF in that values in double quotes are case-sensitive, ie. field names must be in uppercase. Also, the delimiter character must be the same in all cases in a single string.

```
logon-string = userid "?" [ DELIMITER ext-string ] %x00

userid = *( ALPHA / DIGIT / "-")

ext-string = 1*( field )

field = ( node / company-domain / surname / given-name / initials / generation /
org-unit / organization / country / admin / private / resource-name ) DELIMITER
```

Specifying a particular field more than once is, while silly, still legal, although only the last field will be used.

```
node = "ND=" node-number
node-number = 1*DIGIT
company-domain = "CD=" 1*VALUE-CHAR
surname = "S=" 1*VALUE-CHAR ["*"]
given-name = "G=" 1*VALUE-CHAR ["*"]
initials = "I=" 1*VALUE-CHAR ["*"]
generation = "X=" 1*VALUE-CHAR ["*"]
org-unit = ( "OU1" / "OU2" / "OU3" / "OU4" ) "=" 1*VALUE-CHAR ["*"]
organization = "O=" 1*VALUE-CHAR ["*"]
country = "C=" 1*VALUE-CHAR ["*"] DELIMITER
admin = "A=" 1*VALUE-CHAR ["*"] DELIMITER
private = "P=" 1*VALUE-CHAR ["*"] DELIMITER
resource-name = "RS=" 1*VALUE-CHAR ["*"] DELIMITER

DELIMITER = %x01-%x29 / %x2B-%x2F / %x3A-%x3C / %x3E-%x40 / %x5B-%x60 /
%x7B-%x7F
```

Note also that the DELIMITER cannot be used as a VALUE-CHAR.

```
VALUE-CHAR = %x01-29 / %x2B-2E / %x30-7F
```

Data Streams

CAPI deals with MIME (see RFC 2045) encapsulated iCalendar objects for both input and output. A single request may fetch data from a list of calendars. A reply to such a request will consist of a separate iCalendar object for each calendar in the list, inside separate MIME parts. That is, a request for events from calendarA and calendarB results in a MIME stream of this form:

```
... MIME envelope
--MIMEBOUNDARYasdfasdf
Content-type: text/calendar
Content-Transfer-Encoding: quoted-printable
BEGIN:VCALENDAR
... events from calendarA
END:VCALENDAR

--MIMEBOUNDARYasdfasdf
Content-type: text/calendar
Content-Transfer-Encoding: quoted-printable
```

```
BEGIN:VCALENDAR
... events from calendarB
END:VCALENDAR

--MIMEBOUNDARYasdfasdf--
```

The order of the iCalendar objects corresponds to the order of the calendars in the request list. If a request results in an empty solution set, the return stream will be an empty iCalendar object. If there is any sort of error with a calendar the iCalendar reply object corresponding to that calendar will be empty.

For example, in the example above if calendarB was deleted by someone else just prior to an attempt to fetch data, the resulting stream would have the following form:

```
Content-type: text/calendar
Content-Transfer-Encoding: 7bit
BEGIN:VCALENDAR
... events from calendarA
END:VCALENDAR
Content-type: text/calendar
Content-Transfer-Encoding: 7bit
BEGIN:VCALENDAR
END:VCALENDAR
```

On a successful fetch the "VCALENDAR" will contain many "VEVENT" components, each containing the requested properties, if available. iCalendar allows these different components to contain information about different instances of the same event. The returned data may use any of the following methods to give instance specific information:

- Data for each instance can be placed in a different "VEVENT" component, with a different "DTSTART".
- Data for multiple instances can be placed in a single "VEVENT" by identifying instances with the properties "RRULE", "RDATE", "EXRULE" and "EXDATE"
- Hybrids of the above two methods allow grouping of multiple instances which share all properties except their start time in a single "VEVENT" component, and returning many such components.

Please note that the "DTSTART" property returned indicates the start time of the first instance identified in the "VEVENT" component in which it resides and not the start time of the first instance of the event in the Calendar Store. Furthermore the number of "VEVENT" components returned in the calendar has no relation to the

number of instances of the event. Consequently, when fetching events, if the recurrence identifying properties are not requested, there will be no way to determine how many instances exist, and to which instances each returned property applies.

When storing, data supplied to CAPI must consist of a single "VCALENDAR" component inside a single MIME part. The calendar may contain many "VEVENTS", but these must all be information about a single event. For example, this is a valid input:

```
Content-type: text/calendar
Content-Transfer-Encoding: 7bit
BEGIN:VCALENDAR
BEGIN:VEVENT
event properties
END:VEVENT
BEGIN:VEVENT
event properties
END:VEVENT
END:VCALENDAR
```

Access Control

Access to data through CAPI is controlled by the calendar server. It is based on the requester's identity and the data / operation being requested. CAPI provides an interface to request reading any combination of properties. Properties that the requesting user is not authorized to read will not be returned in the CAPI stream and an appropriate status code will be returned by the function call. However, if the user is allowed to read the proprty but it does not exist in the events requested, the property will not be present in the CAPI stream and the return status is not affected. For example, if the VEVENT.GEO property is requested, and it exists, but the requester does not have permission to read it, the resulting CAPI stream will not contain the VEVENT.GEO property in the data stream and an access violation status code will be returned. If the requester has permission to read the VEVENT.GEO property, but the components do not have a GEO property value, the data stream will not contain the VEVENT.GEO property but no error will be returned because of the missing property.

Users will only have access to modify the events to which they are invited, or which they own. If the user is the owner of the event they will have full privileges to modify the event (except for modifying other users' attendance information), otherwise if they are invited to the event they will have restricted privileges to

modify information relating to their own attendance, such as acceptance and alarms.

CAPI will silently ignore attempts to modify properties that the user is not permitted to modify. This simplifies the implementation of modification in these cases.

Errors may occur for specific agendas when attempting to modify events or when creating events. These errors will be returned using a supplied array of status values, allowing the rest of the operation to proceed.

2

Function Reference

This chapter contains detailed information on the following functions:

- CAPI_Logon
- CAPI_SetConfigFile
- CAPI_SetIdentity
- CAPI_StoreEvent

CAPI_AuthenticateAsSysop

```
CAPIStatus CAPI_AuthenticateAsSysop  (  const char *    in_password,
                                        const char *    in_host,
                                        const char *    in_nodeName,
                                        CAPIFlag    in_flags,
                                        CAPISession *    io_session
                                     )
```

Logon as sysop.

Once logged on, the sysop can assume the identity of any user by calling CAPI_SetIdentity().

A node should always be specified since masternode and calendar-domain functionality is not available during logon as sysop.

If the host parameter specifies ACE mechanisms, these will be ignored. The admin default ACE settings from the calendar server will be used.

Sysop authentication is only available with version 5.3 and newer servers. An error will be returned if the provided host does not support this feature. A calendar server may be configured to refuse sysop logon via CAPI in which case a security error will be returned.

The operations available to sysops are limited to:

- logging off by calling CAPI_Logoff()
- switching identity to a user by calling CAPI_SetIdentity()

Once the identity has been set to a user, all operations will be performed as if that user had logged in.

See CAPI_Connect() for the format of the in_host parameter.

Parameters:

- in_password : Sysop's password. May be NULL.
- in_host : Calendar server host name (optional port no.)

- in_nodeName : node ID or alias to connect to as sysop. May be NULL for default node.
- in_flags : Bit flags modifying behaviour. This must be CAPI_FLAG_NONE currently.
- io_session : Session opened by CAPI_Connect

Returns:

CAPIStatus

Cleanup Required:

The sessions created by calling this routine must be destroyed by calling CAPI_Logoff.

Example: Connect to myNode on the server running on the default port of calserver.acme.com, to authenticate as sysop:

```
{
    CAPIStatus  myStatus  = CAPI_AuthenticateAsSysop("theSysopPassword",
                                                    "calserver.acme.com",
                                                    "myNode",
                                                    CAPI_FLAG_NONE,
                                                    &mySession);
}
```

Example: Connect to the default node on the server running on the default port of calserver.acme.com, to authenticate as sysop:

```
{
    CAPIStatus  myStatus  = CAPI_AuthenticateAsSysop("theSysopPassword",
                                                    "calserver.acme.com",
                                                    NULL,
                                                    CAPI_FLAG_NONE,
                                                    &mySession);
}
```

CAPI_Connect

```
CAPIStatus CAPI_Connect  (  const char *    in_host,
                            CAPIFlag    in_flags,
                            CAPISession *    out_session
                         )
```

Establish a connection with a calendar service.

The session obtained in this manner can only be used with CAPI_GetCapabilities, CAPI_Logon or CAPI_Logoff. The session cannot be used to perform any other calendar operations until a user has authenticated using CAPI_Logon.

The format of the in_host parameter is:

```
host-string = hostname "?" ext-param x00
```
Nul-terminated string

```
hostname = *( ALPHA / DIGIT / "." ) [":" port-number]
```
Identifies a calendar store, or calendar domain server (CDS)

```
port-number = 1*DIGIT
```
Identifies the port on which the server is listening (optional). In most cases this should be left out so as to use the default port number

Parameters:

- in_host : calendar server host (with optional port #)
- in_flags : bit flags
- out_session : returns new session

Returns:

CAPIStatus

Example:

```
{
     CAPIStatus  myStatus = CAPI_STAT_OK;
     CAPISession mySession = NULL;
     myStatus = CAPI_connect("calserver.acme.com", CAPI_FLAG_NONE, &mySession);
 }
```

Example: Connect to port 12345 on host calserver.acme.com:

```
 {
     CAPIStatus  myStatus = CAPI_STAT_OK;
     CAPISession mySession = NULL;
    myStatus = CAPI_connect("calserver.acme.com:12345", CAPI_FLAG_NONE, &mySession);
 }
```

CAPI_CreateCallbackStream

```
CAPIStatus CAPI_CreateCallbackStream  (  CAPISession    in_session,
                                         CAPIStream *    out_stream,
                                         CAPICallback    in_sendCallback,
                                         void *    in_sendUserData,
                                         CAPICallback    in_recvCallback,
                                         void *    in_recvUserData,
                                         CAPIFlag    in_flags
                                      )
```

A callback stream can be used to either supply data to, or receive data from CAPI.

C function pointers are given to CAPI for each action (send, receive) which CAPI will call to either read or send data.

During a CAPI_Store...() call, CAPI will call the function in_sendCallback passing in the value in_sendUserData (which is typically used to store some context to be used by the callback function).

During a CAPI_Fetch...() call, CAPI will call the function in_recvCallback passing in the value in_recvUserData (which is typically used to store some context to be used by the callback function).

Both types of callback functions use the same function signature:

```
 typedef int (*CAPICallback)(
      void *      in_userData,    // user-defined data (the value supplied in
                                  // CAPI_CreateCallbackStream)
      char *      io_data,        // buffer to read or write
      size_t      in_dataSize,    // the number of characters to read or write
      size_t *    out_datSize);   // the number of characters read or written
)
```

The return values from the callbacks must be:

Send callback:

- CAPI_CALLBACK_CONTINUE: there is more data to be read from the stream
- CAPI_CALLBACK_DONE: there is no more data to be read from the stream
- a positive integer: an error has occurred. This positive integer will be returned as part of the CAPIStatus returned in bit 5 with the value CAPI_STAT_API_CALLBACK_ERROR

Receive callback:

- CAPI_CALLBACK_CONTINUE: no error
- a positive integer: an error has occurred (e.g. the stream cannot receive any more data). This positive integer will be returned as part of the CAPIStatus returned in bit 5 with the value CAPI_STAT_API_CALLBACK_ERROR

When CAPI has finished writing data to the receive callback, the callback will be called with in_dataSize == 0.

In many applications, it is easier to use either a memory stream, or file stream than a callback stream.

Parameters:

- in_session : login session handle
- out_stream : on output, will point to new stream.
- in_sendCallback : send data callback
- in_sendUserData : a value which will be passed to in_sendCallback
- in_recvCallback : receive data callback
- in_recvUserData : a value which will be passed to in_recvCallback
- in_flags : bit flags (must be CAPI_FLAG_NONE at this time)

Cleanup Required:

The stream returned by this function must be destroyed by calling CAPI_ DestroyStreams()

Returns:

CAPIStatus

Return values:

CAPI_STAT_API_NULL : both supplied callbacks were NULL

See also:

CAPI_CreateMemoryStream() , CAPI_CreateFileStreamFromFilenames()

CAPI_CreateFileStream

```
CAPIStatus CAPI_CreateFileStream  (  CAPISession    in_session,
```

```
    CAPIStream *    out_stream,
    FILE *    in_readFile,
    FILE *    in_writeFile,
    CAPIFlag    in_flags
)
```

Creates file stream to allow CAPI to read from or write to open files.

For compatibility reasons, it is safer to use CAPI_CreateFileStreamFromFilenames which will prevent the need for passing FILE * variables between your application and CAPI.

Files must have been opened using fopen() with an appropriate mode.

Parameters:

- in_session : login session handle
- out_stream : in output, will point to new stream
- in_readFile : FILE * to read from
- in_writeFile : FILE * to write to
- in_flags : bit flags (must be CAPI_FLAG_NONE at this time)

Returns:

CAPIStatus

Example: Store events from the file "events.ics":

```
FILE * myFileFullOfMIMEEncapsulatediCal = fopen("events.ics", "rb");
if (myFileFullOfMIMEEncapsulatediCal != NULL)
{
    CAPIStream myInputStream = NULL;
    CAPIStatus status = CAPI_CreateFileStream(mySession,
                                              &myInputStream,
                                              myFileFullOfMIMEEncapsulatediCal,
                                              NULL,  // no output file
                                              CAPI_FLAG_NONE);
    if (status == CAPI_STAT_OK)
    {
        status = CAPI_StoreEvent(mySession,
                                 myHandles,
                                 numHandles,
                                 handleStatus,
                                 CAPI_STORE_REPLACE,
                                 myInputStream);
```

```
    }
    fclose(myFileFullOfMIMEEncapsulatediCal);
    CAPI_DestroyStreams(mySession,
                        &myInputStream,
                        1,
                        CAPI_FLAG_NONE);
}
```

Example: Fetch events and write them to the file "myAgenda.ics":

```
FILE * outputFile = fopen("myAgenda.ics", "wb");
if (outputFile != NULL)
{
    CAPIStream myOutputStream = NULL;
    CAPIStatus status = CAPI_CreateFileStream(mySession,
                                              &myOutputStream,
                                              NULL,  // no input file
                                              outputFile,
                                              CAPI_FLAG_NONE);
    if (status == CAPI_STAT_OK)
    {
        status = CAPI_FetchEventsByRange(mySession,
                                         myHandles,
                                         numHandles,
                                         handleStatus,
                                         CAPI_FLAG_NONE,
                                         "20020722T000000",
                                         "20020722T235900",
                                         NULL,
                                         0,
                                         myOutputStream);
    }
    fclose(outputFile);
    CAPI_DestroyStreams(mySession,
                        &myOutputStream,
                        1,
                        CAPI_FLAG_NONE);
}
```

CAPI_CreateFileStreamFromFilenames

```
CAPIStatus CAPI_CreateFileStreamFromFilenames  (  CAPISession    in_session,
                                                  CAPIStream *    out_stream,
                                                  const char *    in_readFileName,
                                                  const char *    in_readMode,
                                                  const char *    in_writeFileName,
```

```
    const char *    in_writeMode,
    CAPIFlag    in_flags
)
```

Creates file stream to allow CAPI to read from or write to files.

Parameters:

- in_session : login session handle
- out_stream : in output, will point to new stream
- in_readFileName : name of file to read from
- in_readMode : mode to pass to fopen() while opening in_readFileName
- in_writeFileName : name of file to write to
- in_writeMode : mode to pass to fopen() while opening in_writeFileName
- in_flags : bit flags (must be CAPI_FLAG_NONE at this time)

Returns:

CAPIStatus

Return values:

- CAPI_STAT_SERVICE_FILE_MODE : an invalid mode was passed in
- CAPI_STAT_SERVICE_FILE_OPEN : failed to open a file

Example: Store events from the file "events.ics":

```
CAPIStream myInputStream = NULL;
CAPIStatus status = CAPI_CreateFileStreamFromFilenames(mySession,
                                                       &myInputStream,
                                                       "events.ics",
                                                       "rb",
                                                       NULL,    // no output file
                                                       NULL,    // no output file mode
                                                       CAPI_FLAG_NONE);
if (status == CAPI_STAT_OK)
{
    status = CAPI_StoreEvent(mySession,
                             myHandles,
                             numHandles,
                             handleStatus,
                             CAPI_STORE_REPLACE,
                             myInputStream);
```

```
    CAPI_DestroyStreams(mySession,
                        &myInputStream,
                        1,
                        CAPI_FLAG_NONE);
}
```

Example: Fetch events and write them to the file "myAgenda.ics":

```
CAPIStream myOutputStream = NULL;
CAPIStatus status = CAPI_CreateFileStreamFromFilenames(mySession,
                                                       &myOutputStream,
                                                       NULL,  // no input file
                                                       NULL,   // no input file mode
                                                       "myAgenda.ics",
                                                       "wb",
                                                       CAPI_FLAG_NONE);
if (status == CAPI_STAT_OK)
{
    status = CAPI_FetchEventsByRange(mySession,
                                     myHandles,
                                     numHandles,
                                     handleStatus,
                                     CAPI_FLAG_NONE,
                                     "20020722T000000",
                                     "20020722T235900",
                                     NULL,
                                     0,
                                     myOutputStream);
    CAPI_DestroyStreams(mySession,
                        &myOutputStream,
                        1,
                        CAPI_FLAG_NONE);
}
```

CAPI_CreateMemoryStream

```
CAPIStatus CAPI_CreateMemoryStream  (  CAPISession    in_session,
                                       CAPIStream *    out_stream,
                                       const char *    in_readBuffer,
                                       const char **    out_writeBufferPtr,
                                       CAPIFlag    in_flags
                                    )
```

A memory stream uses data buffers to pass data between your application and CAPI.

This is often the simplest type of stream to use.

Read buffers are read by CAPI during CAPI_Store... calls and write buffers are written to by CAPI during CAPI_Fetch... calls. The read buffers are managed by your application, whereas CAPI will allocate and free the write buffers. The write buffer is freed by CAPI when the memory stream is destroyed.

Parameters:

- in_session : login session handle
- out_stream : on output, will point to new stream.
- in_readBuffer : buffer for CAPI to read from
- out_writeBufferPtr : This address will point to the buffer CAPI is writing into.
- in_flags : bit flags (must be CAPI_FLAG_NONE at this time)

Cleanup Required:

The stream returned by this function must be destroyed by calling CAPI_ DestroyStreams.

Returns:

CAPIStatus

Return values:

CAPI_STAT_API_NULL : both supplied buffers were NULL

Example: Store events from the buffer "events"

```
const char events[] = "MIME-Version: 1.0\n"
                      "Content-Type: text/calendar\n"
                      "Content-Transfer-Encoding: quoted-printable\n\n"
                      "BEGIN:VCALENDAR\n"
                      "VERSION:2.0\n"
                      ...etc
                      "END:VCALENDAR\n";
CAPIStream myInputStream = NULL;
CAPIStatus status = CAPI_CreateMemoryStream(mySession,
                                            &myInputStream,
                                            events,
                                            NULL, // no write buffer
                                            CAPI_FLAG_NONE);
if (status == CAPI_STAT_OK)
{
```

```
    status = CAPI_StoreEvent(mySession,
                             myHandles,
                             numHandles,
                             handleStatus,
                             CAPI_STORE_REPLACE,
                             myInputStream);
    CAPI_DestroyStreams(mySession,
                        &myInputStream,
                        1,
                        CAPI_FLAG_NONE);
}
```

Example: Fetch events and write them to a buffer

```
const char * todaysEvents = NULL;
CAPIStream myOutputStream = NULL;
CAPIStatus status = CAPI_CreateMemoryStream(mySession,
                                            &myOutputStream,
                                            NULL, // no read buffer
                                            &todaysEvents,
                                            CAPI_FLAG_NONE);
if (status == CAPI_STAT_OK)
{
    status = CAPI_FetchEventsByRange(mySession,
                                     myHandles,
                                     numHandles,
                                     handleStatus,
                                     CAPI_FLAG_NONE,
                                     "20020722T000000",
                                     "20020722T235900",
                                     NULL,
                                     0,
                                     myOutputStream);
    if (status == CAPI_STAT_OK)
    {
        printf("Today's events:\n%s", todaysEvents);
    }
    CAPI_DestroyStreams(mySession,
                        &myOutputStream,
                        1,
                        CAPI_FLAG_NONE);
}
```

CAPI_DeleteEvent

```
CAPIStatus CAPI_DeleteEvent  (  CAPISession    in_session,
                                CAPIHandle *    in_handles,
                                int    in_numHandles,
                                CAPIStatus *    io_status,
                                CAPIFlag    in_flags,
                                const char *    in_UID,
                                const char *    in_RECURRENCEID,
                                int    in_modifier
                             )
```

This function deletes the specified event from the specified agendas.

The events to delete are identified by the UID, recurrence ID and modifiers. If the recurrence ID is not specified it is assumed that all recurrences should be deleted. The event is only removed from the agendas specified by the supplied CAPIHandles.

Different agendas may have different events with the same UID, at most one of which the current user may be able to modify. Each element in the array of errors may be set to indicate that there was no such UID, that the current user cannot modify that event, or that the event was found, and deleted from the agenda (OK). If no event could be deleted from any agenda the returned status will be fatal. If any agenda had a security error, the returned status will be a security error. If no agenda had a security error, but at least one had a no such UID error, then the returned status will be no such UID. Otherwise the returned status will be OK, (Unless an error occured in processing).

Parameters:

- in_session : login session handle
- in_handles : calendar(s) from which to delete events
- in_numHandles : number of handles in in_handles
- io_status : array (preallocated) to hold 1 status / handle
- in_flags : bit flags
- in_UID : UID of the event to delete
- in_RECURRENCEID : recurrence-id, NULL means ignore. Must be a NUL-terminated string or NULL.
- in_modifier : one of:

- CAPI_THISINSTANCE
- CAPI_THISANDPRIOR
- CAPI_THISANDFUTURE

Returns:

CAPIStatus

CAPI_DestroyHandles

```
CAPIStatus CAPI_DestroyHandles  (  CAPISession    in_session,
                                   CAPIHandle *    io_handles,
                                   int    in_numHandles,
                                   CAPIFlag    in_flags
                                )
```

Destroy handles returned by CAPI_GetHandle().

Parameters:

- io_session : login session handle
- io_handles : Array of handles (returned by CAPI_GetHandle) to destroy
- in_handleCount : The size of the handle array
- in_flags : bit flags (none at this time; set to CAPI_FLAG_NONE)

Returns:

CAPIStatus

Example:

```
{
    CAPIHandle h1, h2;
    CAPI_GetHandle(mySession, "arthur", CAPI_FLAG_NONE, &h1);
    CAPI_GetHandle(mySession, "tim...", CAPI_FLAG_NONE, &h2);
    ...
    CAPIHandle handles[] = {h1, h2};
    CAPI_DestroyHandles(mySession, handles, 2, CAPI_FLAG_NONE);
}
```

CAPI_DestroyStreams

```
CAPIStatus CAPI_DestroyStreams  (  CAPISession    in_session,
                                   CAPIStream *    in_streams,
                                   int    in_numStreams,
                                   CAPIFlag    in_flags
                                )
```

This function destroys streams created by the various CAPI_CreateXXXStream functions.

Parameters:

- in_session : the session to which streams are associated. Must be a session returned by CAPI_Logon
- in_streams : array of streams to destroy.
- in_numStreams : the number of streams in in_streams to destroy
- in_flags : bit flags modifying behavior. Must be CAPI_FLAG_NONE at this time.

Returns:

CAPIStatus

CAPI_FetchEventByID

```
CAPIStatus CAPI_FetchEventByID  (  CAPISession    in_session,
                                   CAPIHandle    in_handle,
                                   CAPIFlag    in_flags,
                                   const char *    in_UID,
                                   const char *    in_RECURRENCEID,
                                   int    in_modifier,
                                   const char **    in_requestProperties,
                                   int    in_numProperties,
                                   CAPIStream    in_stream
                                )
```

This function fetches an event from the server using its iCalendar UID.

First the event with the specified UID is retrieved from the specified agenda. If the property list is not NULL the event is stripped to include only those specified properties. The event is then encapsulated in a calendar object inside a MIME object and written to the supplied stream.

If the event cannot be found an error is returned and nothing is written to the stream. If a property is requested, but cannot be returned for security reasons a non-fatal error is returned.

Parameters:

- in_session : login session handle
- in_handle : calendar from which to fetch events
- in_flags : bit flags
- in_UID : UID of the event to fetch
- in_RECURRENCEID : recurrence-id, NULL means ignore
- in_modifier : one of:
  - CAPI_THISINSTANCE
  - CAPI_THISANDPRIOR
  - CAPI_THISANDFUTURE only used if recurrence-id is non-NULL

Parameters:

- in_requestProperties : array of iCalendar properties to return (NULL ==> return default set of properties)
- in_numProperties : number of properties in in_requestProperties
- in_stream : stream for CAPI to write into

Returns:

CAPIStatus

CAPI_FetchEventsByAlarmRange

```
CAPIStatus CAPI_FetchEventsByAlarmRange  (  CAPISession    in_session,
                                            CAPIHandle *    in_handles,
                                            int    in_numHandles,
                                            CAPIStatus *    io_status,
                                            CAPIFlag    in_flags,
                                            const char *    in_DTSTART,
                                            const char *    in_DTEND,
                                            const char **    in_requestProperties,
                                            int    in_numProperties,
                                            CAPIStream    in_stream
```

)

This function downloads events with alarms which fall in the specified time range and returns them as MIME-Encapsulated iCalendar data via the CAPIStream.

For each handle; events are downloaded from that handle's associated agenda, as long as the event has an alarm which falls in the time range between the indicated start and end times. If in_requestProperties is non-NULL the events are stripped to include only the indicated properties. The objects are then sent to the CAPIStream.

A single MIME multipart object is written to the stream. A part is created in the MIME object for each handle which was passed. Each part contains a single Calendar object. The calendar object will contain any events that were found on the indicated agenda. The order of the returned calendar objects corresponds to the order of the received handles.

If any of the indicated properties are not available for security reasons a non-fatal error will be returned.

Parameters:

- in_session : login session handle
- in_handles : calendar(s) from which to fetch events
- in_numHandles : number of handles in in_handles
- io_status : array (preallocated) to hold 1 status / handle
- in_flags : bit flags
- in_DTSTART : range start time eg "20010709T000000Z"
- in_DTEND : range end time eg "20020709T000000Z"
- in_requestProperties : array of iCalendar properties to return (NULL ==> return default set of properties)
- in_numProperties : number of properties in in_requestProperties
- in_stream : stream for CAPI to write into

Returns:

CAPIStatus

CAPI_FetchEventsByRange

```
CAPIStatus CAPI_FetchEventsByRange  (  CAPISession    in_session,
                                       CAPIHandle *    in_handles,
                                       int    in_numHandles,
                                       CAPIStatus *    io_status,
                                       CAPIFlag    in_flags,
                                       const char *    in_DTSTART,
                                       const char *    in_DTEND,
                                       const char **    in_requestProperties,
                                       int    in_numProperties,
                                       CAPIStream    in_stream
                                    )
```

This function downloads events which fall in the specified time range and returns them as MIME-Encapsulated iCalendar data via the CAPIStream.

For each handle; events are downloaded from that handle's associated agenda, as long as the event overlaps the time range between the indicated start and end times. If in_requestProperties is non-NULL the events are stripped to include only the indicated properties. The objects are then sent to the CAPIStream.

A single MIME multipart object is written to the stream. A part is created in the MIME object for each handle which was passed. Each part contains a single Calendar object. The calendar object will contain any events that were found on the indicated agenda. The order of the returned calendar objects corresponds to the order of the received handles.

If any of the indicated properties are not available for security reasons a non-fatal error will be returned.

Please note that some vendors may not support the complete range of dates that iCalendar supports. If this is the case then the CAPI should behave as if the server supports the full range of iCalendar dates and that there are no events outside the server supported range. However CAPI should indicate if a fetch range overlapped the range not supported by the server by returning a non-fatal error. The server supported date ranges can be obtained through a call to CAPI_GetCapabilities.

Parameters:

- in_session : login session handle
- in_handles : calendar(s) from which to fetch events
- in_numHandles : number of handles in in_handles
- io_status : array (preallocated) to hold 1 status / handle

- in_flags : bit flags
- in_DTSTART : range start time eg "20010709T000000Z"
- in_DTEND : range end time eg "20020709T000000Z"
- in_requestProperties : array of iCalendar properties to return (NULL ==> return default set of properties)
- in_numProperties : number of properties in in_requestProperties
- in_stream : stream for CAPI to write into

Returns:

CAPIStatus

CAPI_GetCapabilities

```
CAPIStatus CAPI_GetCapabilities  (  CAPISession    in_session,
                                    CAPICapabilityID    in_capabilityID,
                                    CAPIFlag    in_flags,
                                    const char **    out_value
                                 )
```

Returns information on this CAPI release and / or the calendar server.

Parameters:

- in_session : session. If NULL, then no server capabilities can be requested.
- in_capabilityID : ID for a capability (see CAPI_CAPAB_* in ctapi.h)
- in_flags : CAPI_FLAG_NONE at this time
- out_value : information is returned in this param. The values are returned as read-only strings and are only valid until the next CAPI call which uses the same session.

Changes:

CAPI 2.5: type of in_capabilityID was changed from "long" to "CAPICapabilityID"

CAPI_GetHandle

```
CAPIStatus CAPI_GetHandle  (  CAPISession    in_session,
                              const char *    in_user,
```

```
    CAPIFlag    in_flags,
    CAPIHandle *    out_handle
)
```

This function returns a handle to a particular user's calendar store.

With this handle subsequent calls can access items in this agenda. If an error is returned no CAPIHandle will be allocated and no cleanup is required.

The logon string follows the same format as that of the string used by CAPI_Logon.

A handle to the current user is returned if in_user is NULL.

This function is blocked for sysop that has not assumed the identity of a user.

Parameters:

- in_session : login session handle
- in_user : user as defined for CAPI_Login. May be NULL in which case a handle to the current user is returned.
- in_flags : bit flags (none at this time; set to CAPI_FLAG_NONE)
- out_handle : handle for in_user. Must point to NULL on entry.

Returns:

CAPIStatus

Return values:

- CAPI_STAT_OK
- CAPI_STAT_DATA_USERID
- CAPI_STAT_SERVICE_MEM
- CAPI_STAT_SERVICE_FILE
- CAPI_STAT_SERVICE_NET
- CAPI_STAT_API_FLAGS
- CAPI_STAT_API_NULL
- CAPI_STAT_API_HANDLE
- CAPI_STAT_API_SESSION
- CAPI_STAT_LIBRARY

Cleanup Required:

This function allocates a handle which must be cleaned up with a call to CAPI_DestroyHandles. If an error is returned no handle is allocated and no clean up is required.

Example: Get a handle for a user whose userid is "roger":

```
 {
     CAPIHandle shrubber = NULL;
     stat = CAPI_GetHandle(mySession, "roger", CAPI_FLAG_NONE, &shrubber);
 }
```

Example: Get a handle for a user named "Arnold Layne" (Surname Layne, Given name Arnold):

```
{
    CAPIHandle arnold = NULL;
    stat = CAPI_GetHandle(mySession, "?/S=Layne/G=Arnold/, CAPI_FLAG_NONE,
&arnold);
}
```

Example: Get a handle for a resource named "keg" on node "1234":

```
{
    CAPIHandle keg = NULL;
    stat = CAPI_GetHandle(mySession, "?/RS=keg/ND=1234/", CAPI_FLAG_NONE, &keg);
}
```

Example: Get a handle for the current user:

```
 {
     CAPIHandle currUser = NULL;
     stat = CAPI_GetHandle(mySession, NULL, CAPI_FLAG_NONE, &currUser);
 }
```

Changes:

CAPI 2.5: Resource names must be an exact match. (There used to be an implicit * at the end of the string.)

CAPI_GetLastStoredUIDs

```
CAPIStatus CAPI_GetLastStoredUIDs  (  CAPISession    in_session,
                                      char const *const **    out_UIDs,
                                      unsigned long *    out_count,
```

```
                                        CAPIFlag    in_flags
                                        )
```

This function returns the UID(s) of the last event(s) stored by a successful call to CAPI_StoreEvent, or no UID if the there was no call to CAPI_StoreEvent or the last call was not successful.

The UIDs returned are static read-only strings. The strings are only valid until the next call to CAPI_StoreEvent; hence, it may be desirable to copy (e.g. strcpy()) the UIDs to another variable. These UIDs are appropriate for use in subsequent CAPI calls.

Parameters:

- in_session : login session handle
- out_UIDs : on output, will point to an array of read-only strings
- out_count : number of UIDs in out_UIDs
- in_flags : bit flags (none at this time; set to CAPI_FLAG_NONE)

Returns:

CAPIStatus

Return values:

- CAPI_STAT_OK
- CAPI_STAT_API_SESSION_NULL : Indicating that in_session was NULL
- CAPI_STAT_API_NULL : Indicating that out_UIDs was NULL
- CAPI_STAT_API_FLAGS : Indicating that in_flags was not CAPI_FLAG_NONE

Changes:

CAPI 2.5 : type of "out_count" changed from "long *" to "unsigned long *"

Example:

```
{
      const char ** newUIDs = NULL;
      unisigned long   numUIDs = 0;
      CAPI_StoreEvent(mySession, ...);
      stat = CAPI_GetLastStoredUIDs(mySession, &newUIDs, &numUIDs, CAPI_FLAG_NONE);
      for (unisigned long u = 0; u < numUIDs; u++)
```

```
    {
        cout << "Stored UID:" << newUIDs[u] << endl;
    }
}
```

CAPI_GetStatusLevels

```
void CAPI_GetStatusLevels  (  CAPIStatus    in_status,
                              unsigned long *    out_level1,
                              unsigned long *    out_level2,
                              unsigned long *    out_level3,
                              unsigned long *    out_level4,
                              unsigned long *    out_level5
                           )
```

This function decomposes a CAPIStatus into its sub-parts.

Each part of the status code specifies more precisely the actual error.

Parameters:

- in_status : CAPI status
- out_level1: will contain the int result for level1
- out_level2: will contain the int result for level2
- out_level3: will contain the int result for level3
- out_level4: will contain the int result for level4
- out_level5: will contain the int result for level5

Changes:

CAPI 2.5 : types of "out_level[12345]" changed from "int *" to "unsigned long *"

CAPI_GetStatusString

```
void CAPI_GetStatusString  (  CAPIStatus    in_status,
                              const char **    out_string
                           )
```

This function returns a read-only string representation of a CAPIStatus.

This is generally more useful to a programmer than the numeric representation.

Parameters:

- in_status : CAPI status
- out_string: will contain const pointer to the result string

Cleanup Required:

None. The string returned is a const string that cannot be freed

CAPI_HandleInfo

```
CAPIStatus CAPI_HandleInfo  (  CAPISession    in_session,
                               CAPIHandle    in_handle,
                               CAPIFlag    in_flags,
                               const char **    out_info
                            )
```

This function returns information about the agenda of the supplied handle.

Three pieces of information can be returned, chosen by the value of in_flags. The information is returned as a pointer to a static read-only string.

CAPI_HANDLE_TYPE indicates the type of the handle, this can be "user" or "resource" and indicates what type of agenda this is. CAPI_HANDLE_NAME returns the name of the agenda owner, or resource, in the form of a sequence of field-value pairs, separated by "/". This string, when prepended with a '?' is of an appropriate format to be passed to CAPI_GetHandle. A description of this format is given in "User identification" section of this manual. CAPI_HANDLE_MAILTO returns the email address of who the agenda belongs to. Since not all users (and no resources) will have e-mail addresses set on the calendar server, an error (CAPI_STAT_DATA_EMAIL_NOTSET) will be returned when no e-mail address is set.

Parameters:

- in_session : login session handle
- in_handle : handle to get info for
- in_flags : CAPI_HANDLE_TYPE, CAPI_HANDLE_NAME or CAPI_HANDLE_MAILTO
- out_info : read-only handle information

Returns:

CAPIStatus

Changes:

CAPI 2.5: now returns CAPI_STAT_DATA_EMAIL_NOTSET if no e-mail address is set on the server.

Example: Print the name of the logged in user:

```
{
    CAPIHandle  loginUser = NULL;
    const char * fullName = NULL;
    stat = CAPI_GetHandle(mySession, NULL, CAPI_FLAG_NONE, &loginUser);
    stat = CAPI_HandleInfo(mySession, loginUser, CAPI_HANDLE_NAME, &fullName);
    cout << "Currently logged in as " << fullName << endl;
    CAPI_DestroyHandles(mySession,
                        &loginUser,
                        1,
                        CAPI_FLAG_NONE);
}
```

Example: Print out Doctor Winston's e-mail address:

```
{
    CAPIHandle   doctor = NULL;
    const char * email  = NULL;
    stat = CAPI_GetHandle(mySession, "drwinston", CAPI_FLAG_NONE, &doctor);
    stat = CAPI_HandleInfo(mySession, doctor, CAPI_HANDLE_MAILTO, &email);
    cout << "drwinston's email address is " << email << endl;
    CAPI_DestroyHandles(mySession,
                        &doctor,
                        1,
                        CAPI_FLAG_NONE);
}
```

CAPI_Logoff

```
CAPIStatus CAPI_Logoff  (  CAPISession *    io_session,
                           CAPIFlag    in_flags
                           )
```

Use CAPI_Logoff to either de-authenticate, or completely close your CAPISession.

To select which action to perform, use the in_flags parameter with either of these values:

CAPI_MODE_NONE: De-authenticate and disconnect from the calendar server

CAPI_LOGOFF_STAY_CONNECTED: De-authenticate but stay connected. The session can still be used to read capabilities and can be used to re-authenticate (as the same user or a different user) using CAPI_Logon().

Parameters:

- io_session : login session handle. Must be a session returned by CAPI_Logon() or CAPI_Connect()
- in_flags : Bit flags modifying behaviour: CAPI_MODE_NONE or CAPI_LOGOFF_STAY_CONNECTED

Returns:

CAPIStatus

Return values:

- CAPI_STAT_OK
- CAPI_STAT_SERVICE_MEM
- CAPI_STAT_SERVICE_FILE
- CAPI_STAT_SERVICE_NET
- CAPI_STAT_API_FLAGS
- CAPI_STAT_API_NULL
- CAPI_STAT_API_SESSION
- CAPI_STAT_LIBRARY

Cleanup Required:

Since the "stay connected" mode does not destroy the session, a final call to CAPI_Logoff is needed to destroy the sesssion in this case.

Example: Disconnect from a server:

```
{
    ...
    stat = CAPI_Logoff(&mySession, CAPI_MODE_NONE);
    // mySession should now be NULL
}
```

Example: Re-authenticate as a different user:

```
{
    ...
    stat = CAPI_Logoff(&mySession, CAPI_LOGOFF_STAY_CONNECTED);
    ...
    stat = CAPI_Logon("manfromscene24", "blue", "", CAPI_FLAG_NONE, &mySession);
}
```

CAPI_Logon

```
CAPIStatus CAPI_Logon  (  const char *    in_user,
                          const char *    in_password,
                          const char *    in_host,
                          CAPIFlag    in_flags,
                          CAPISession *    io_session
                       )
```

Establish a session with the Calendar Store.

No session will be returned if an error occurs.

The session parameter can be used in 2 ways:

- Preinitialize the session to NULL. In this case, the function will connect to the calendar service, authenticate as the given user and return a new session.
- Reuse an active session obtained from a previous call to CAPI_Connect (or CAPI_Logon, when you use CAPI_Logoff without disconnecting) and authenticate as a new user. Since the connection to a calendar service has already been established, the hostname may be omitted.

With calendar servers that support the ACE framework, CAPI_Logon will try to use the default ACE mechanisms set by the server. For authentication mechanisms that don't require user credentials at each logon (e.g. gssapi:kerberos5 or web:CAL), an empty string may be passed for the user and password parameters.

The in_host parameter is the same as documented in CAPI_Connect except it may have extended parameters as shown:

```
host-string = hostname "?" ext-param x00

ext-param = DELIMITER fields
      the extended string is used to specify ACE mechanisms

fields = [authentication-mech] [compression-mech] [encryption-mech]

authentication-mech = "AUTH=" ACE-mechanism DELIMITER
```

```
compression-mech = "COMP=" ACE-mechanism DELIMITER

encryption-mech = "ENCR=" ACE-mechanism DELIMITER
      To find out which ACE-mechanisms the server support
      use CAPI_Connect followed by CAPI_GetCapabilities

 DELIMITER = x01-29 / x2B-2F / x3A-3C / x3E-40 / %5B-60 / %7B-7F
      Everything except NUL, "*", DIGIT, "=", ALPHA
```

Please refer to the section on User Identification for the format of the in_user parameter.

Parameters:

- in_user : Must be a null-terminated string
- in_password : User's password. May be NULL.
- in_host : Calendar server host name (optional port no.)
- in_flags : Bit flags modifying behaviour. This must be 0 currently.
- io_session : Session opened by CAPI_Connect, or NULL if opening a new connection

Returns:

CAPIStatus

Cleanup Required:

The sessions created by calling this routine must be destroyed by calling CAPI_ Logoff.

Example: Connect to a server running on the default port of calserver.acme.com, to authenticate as userid keithm using default ACE settings (when no node is specified, either a master node or default node must be configured on the specified host):

```
{
    CAPISession mySession = NULL;
    myStatus = CAPI_connect("calserver.acme.com", CAPI_FLAG_NONE, &mySession);
    if (myStatus == CAPI_STAT_OK)
    {
        myStatus = CAPI_Logon("keithm",
                              "abcdefg",
                              "",
```

```
                                CAPI_FLAG_NONE,
                                &mySession);
    }
}
```

Example: Connect to a server running on the default port of calserver.acme.com, to authenticate as user "Keith MacDonald" using default ACE settings:

```
{
    CAPISession mySession = NULL;
    myStatus = CAPI_connect("calserver.acme.com", CAPI_FLAG_NONE, &mySession);
    if (myStatus == CAPI_STAT_OK)
    {
        myStatus = CAPI_Logon("?/S=MacDonald/G=Keith/ND=200/",
                                "abcdefg",
                                "",
                                CAPI_FLAG_NONE,
                                &mySession);
    }
}
```

Example: Connect to a server running on the default port of calserver.acme.com, to authenticate as userid keithm on node 200 using default ACE settings:

```
{
    CAPISession mySession = NULL;
    myStatus = CAPI_connect("calserver.acme.com", CAPI_FLAG_NONE, &mySession);
    if (myStatus == CAPI_STAT_OK)
    {
        myStatus = CAPI_Logon("keithm?/ND=200/",
                                "abcdefg",
                                "",
                                CAPI_FLAG_NONE,
                                &mySession);
    }
}
```

Example: Connect to a server running on calserver.acme.com, using gssapi:kerberos5 authenticatication:

```
{
    CAPISession mySession = NULL;
    myStatus = CAPI_connect("calserver.acme.com", CAPI_FLAG_NONE, &mySession);
    if (myStatus == CAPI_STAT_OK)
    {
```

```
        myStatus = CAPI_Logon("",
                              "",
                              "?/AUTH=gssapi:kerberos5/",
                              CAPI_FLAG_NONE,
                              &mySession);
    }
}
```

(Note: for web authentication, replace "gssapi:kerberos5" with "web:CAL")

Example: Authenticate as user keithm using the company domain MYDOMAIN hosted by the domain server "mycds.myasp.com"

```
{
    CAPISession mySession = NULL;
    CAPIStatus  myStatus  = CAPI_Logon("keithm?/CD=MYDOMAIN/",
                                       "abcdefg",
                                       "mycds.myasp.com"
                                       CAPI_FLAG_NONE,
                                       &mySession);
}
```

CAPI_SetConfigFile

```
CAPIStatus CAPI_SetConfigFile  (  const char *    in_configFileName,
                                  const char *    in_logFileName
                               )
```

Calling this function will allow CAPI to read configuration settings which control error logging, and the other configuration parameters listed in the "Configuration" section of this manual.

Note: To use the "web" ACE authentication module, you MUST call this function since the web authentication reads configuration settings from this file.

Note: If called, this function should be the first CAPI function called by your process and should not be called by each thread.

Parameters:

- const char * in_configFileName : A null-terminated string containing the filename of the config file.
- const char * in_logFileName : the name of a file to write log messages to

Returns:

CAPIStatus

Return values:

CAPI_STAT_API_NULL : one of the input parameters was NULL

See also:

The Configuration section.

Example: Create a file "capi.ini" with the contents:

```
[LOG]
log_activity = true
log_modulesinclude = { CAPI }
```

and call CAPI_SetConfigFile:

```
CAPIStatus stat = CAPI_SetConfigFile("capi.ini", "capi.log");
```

This will turn on "activity" level logging in CAPI and the output will go into capi.log.

CAPI_SetIdentity

```
CAPIStatus CAPI_SetIdentity  (  CAPISession    in_session,
                                const char *    in_user,
                                CAPIFlag    in_flags
                             )
```

Allow an authenticated user (CAPI_Logon) to work on behalf of another calendar user or resource.

For this to work, full designate rights must have been set in advance; otherwise a security error will be returned.

The format of the in_user parameter is the same as in the CAPI_Logon function. The authenticated user may revert to her original identity by using NULL as username.

If you've logged in as sysop (CAPI_AuthenticateAsSysop), then designate rights are ignored and you will be able to work as any calendar user or resource. All calendar operations will appear to have been done by the user, rather than on behalf of the user by a designate.

Parameters:

- in_session : valid CAPI session
- in_user : person (or resource) to work as - an X400 or uid
- in_flags : CAPI_FLAG_NONE at this time

Returns:

CAPIStatus

Example:

```
myStatus = CAPI_SetIdentity(mySession, "keithm", CAPI_FLAG_NONE);
myStatus = CAPI_SetIdentity(mySession, "?/S=MacDonald/G=Keith/", CAPI_FLAG_NONE);
myStatus = CAPI_SetIdentity(mySession, "?/RS=Conference Room/ND=1234/", CAPI_FLAG_NONE);
```

Changes:

CAPI 2.5: Resource names must be an exact match. (There used to be an implicit * at the end of the string.)

CAPI_StoreEvent

```
CAPIStatus CAPI_StoreEvent  (  CAPISession    in_session,
                               CAPIHandle *    in_handles,
                               int    in_numHandles,
                               CAPIStatus *    io_status,
                               CAPIFlag    in_flags,
                               CAPIStream    in_stream
                            )
```

Store events in the supplied list of calendars.

If an event is already in the Calendar (as identified by its UID and Recurrence-ID) it will be updated. CAPI is not responsible for enforcing iCalendar security or data integrity as defined by the "ORGANIZER", "SEQUENCE" and "DTSTAMP" properties. It is the responsibility of the CAPI user to be familiar with this model and to enforce it. CAPI_StoreEvent is capable of overwriting these properties.

When this function returns, the buffer for CAPIStatus return values, io_status, will contain the status associated with each handle relative to the store operation. That is, io_status[i] holds the status of the store operation for the calendar addressed by in_handles[i]. io_status must be large enough to hold in_numHandles CAPIStatus

codes. A failure for one particular handle will not cause a failure for the entire operation.

CAPI_StoreEvent allows data to passed in several VEVENT components. Each VEVENT may or may not refer to the same event. These will be processed as if they had been supplied to many consecutive calls to CAPI_StoreEvent, one per call, in the order in which they appear in the "VCALENDAR". Despite the fact that behaviour is the same, it is preferable to make only one call to CAPI_StoreEvent, as this is likely to be more efficient than many calls. A huge collection of VEVENTs will however require more memory to process.

CAPI_StoreEvent can create new instances, either of a new event or of an event which is already on the server. It can also be used to modify groups of instances, either all, one or a range of instances of an existing event. The appropriate behaviour is chosen through the specification of the properties "UID", "RECURRENCE-ID", "DTSTART", "RDATE", "RRULE", "EXDATE" and "EXRULE".

If "UID" does not match an event that already exists on the server then it is a request to create a new event on the server. In this case a "RECURRENCE-ID" property may not be specified, and a "DTSTART" property must be specified.

If the "UID" property matches an event that already exists on the server, the "RECURRENCE-ID" property is not present, and at least one of "DTSTART", "RDATE", "RRULE", "EXDATE" and "EXRULE" is present then the request is to add new instances to the event. If any instances on the server are identified by these dates they will be overwritten.

If the "UID" property matches an event that already exists on the server, and none of "RECURRENCE-ID", "DTSTART", "RDATE", "RRULE", "EXDATE" or "EXRULE" are present, then this is a request to modify all of the instances of an event. In this case the "DTEND" property may not be present.

If the "UID" property matches an event that already exists on the server, and the "RECURRENCE-ID" property is present, and it has no "range" parameter, then this is a request to modify a single instance. None of the properties "RDATE", "RRULE", "EXDATE" and "EXRULE" can be specified in this case, but the "DTSTART" property can be used to modify the start time of the particular instance.

If the "UID" property matches an event that already exists on the server, and the "RECURRENCE-ID" property is present, and it has a "range" parameter, then this is a request to modify a range of instances. None of the properties "DTSTART", "RDATE", "RRULE", "EXDATE" and "EXRULE" can be specified in this case. Also, "DTEND" may not be specified, although "DURATION" may be.

The iCalendar "RRULE" property is a concise way of specifying many instances of an event, however CAPI expands recurrence rules to specific dates, storing only these dates, and discarding the expanded recurrence rule. In the case of a recurrence rule that goes for ever, the list of created dates will be limited to a server-defined number of instances.

E-mail and wireless notification of newly created and modified events can be sent using the CAPI_NOTIFY_EMAIL and CAPI_NOTIFY_SMS flags.

E-mail, wireless and audio reminders to be triggered in advance of an event are also supported. Specify them using VALARM objects in the following format:

```
BEGIN:VALARM
ACTION:<name>
TRIGGER:-PT2H1M
END:VALARM
```

where <name> can take one of the following values:

- "EMAIL" for an e-mail reminder
- "X-STELTOR-SMS" for a wireless reminder, or AUDIO for an audio reminder.
- TRIGGER specifies the time in advance of the event start time at which to set the reminder.

CAPI_StoreEvent also provides the following mutually exclusive bit flags which govern the interpretation of the rest of the specified properties:

- CAPI_STORE_REPLACE
- CAPI_STORE_UPDATE
- CAPI_STORE_DELPROP

CAPI_STORE_REPLACE

This mode can be used when creating new instances or modifying existing ones. When creating new instances they are created with all the properties as specified in the data. When existing instances are being modified all their properties are replaced with the ones specified in the supplied data.

Example: Event as it currently appears in the calendar store, prior to calling CAPI_StoreEvent (applies to all examples):

```
BEGIN:VEVENT
ORGANIZER:Mailto:A@example.com
ATTENDEE;ROLE=CHAIR;PARTSTAT=ACCEPTED;CN=BIG A:Mailto:A@example.com
```

```
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=B:Mailto:B@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=C:Mailto:C@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=Hal:Mailto:D@example.com
ATTENDEE;RSVP=FALSE;TYPE=ROOM:conf_Big@example.com
ATTENDEE;ROLE=NON-PARTICIPANT;RSVP=FALSE:Mailto:E@example.com
DTSTAMP:19980611T190000Z
DTSTART:19980701T200000Z
DTEND:19980701T2100000Z
SUMMARY:Conference
LOCATION:The Big Conference Room
UID:calsrv.example.com-873970198738777@example.com
SEQUENCE:0
STATUS:CONFIRMED
END:VEVENT
```

Call CAPI_StoreEvent with the following ICAL stream, which modifies a particular instance:

```
BEGIN:VCALENDAR
PRODID:-//ACME//NONSGML DesktopCalendar//EN
VERSION:2.0
BEGIN:VEVENT
ORGANIZER:Mailto:A@example.com
ATTENDEE;ROLE=CHAIR;PARTSTAT=ACCEPTED;CN=BIG A:Mailto:A@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=B:Mailto:B@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=C:Mailto:C@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=Hal:Mailto:D@example.com
DTSTAMP:19980611T190000Z
RECURRENCE-ID:19980701T200000Z
DURATION:PT1H
SUMMARY:Conference
UID:calsrv.example.com-873970198738777@example.com
SEQUENCE:1
STATUS:CONFIRMED
END:VEVENT
END:VCALENDAR
```

Upon successful completion of the CAPI_StoreEvent call, the event stored on the server will have its properties replaced the net effect being:

- Two ATEENDEE properties were removed (conf_Big@example.com and E@example.com)
- The LOCATION property was removed
- The SEQUENCE-NUMBER property was bumped.

CAPI_STORE_UPDATE

This mode can be used when creating new instances or modifying existing ones. When creating new instances they are created with all the properties as specified in the data. When existing instances are being modified, this mode only affects the specified properties. Any property that is updated must be completely updated. Effectively, all instances of any supplied property are completely deleted from the event on the calendar store and replaced with supplied properties. Thus, even to update a single ATTENDEE property, all ATTENDEE properties must be supplied.

Example: Call CAPI_StoreEvent with the following ICAL stream:

```
BEGIN:VCALENDAR
PRODID:-//ACME//NONSGML DesktopCalendar//EN
VERSION:2.0
BEGIN:VEVENT
RECURRENCE-ID:19980701T200000Z
DTEND:19980701T2200000Z
DESCRIPTION:We need to discuss the project schedule. Please come prepared.
UID:calsrv.example.com-873970198738777@example.com
SEQUENCE:1
END:VEVENT
END:VCALENDAR
```

Call CAPI_StoreEvent with the following ICAL stream:

```
BEGIN:VCALENDAR
PRODID:-//ACME//NONSGML DesktopCalendar//EN
VERSION:2.0
BEGIN:VEVENT
ATTENDEE;ROLE=CHAIR;PARTSTAT=ACCEPTED;CN=BIG A:Mailto:A@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=B:Mailto:B@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=C:Mailto:C@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=Hal:Mailto:D@example.com
ATTENDEE;RSVP=FALSE;TYPE=ROOM:conf_Big@example.com
ATTENDEE;ROLE=NON-PARTICIPANT;RSVP=FALSE:Mailto:E@example.com
RECURRENCE-ID:19980701T200000Z
DTEND:19980701T2200000Z
DESCRIPTION:We need to discuss the project schedule. Please come prepared.
UID:calsrv.example.com-873970198738777@example.com
SEQUENCE:1
END:VEVENT
END:VCALENDAR
```

When CAPI_StoreEvent is called with either one of the above ICAL streams, the end time of the event recurrence will be moved, a description will be added, and the sequence number will be bumped. The calendar store copy will be as follows:

```
BEGIN:VEVENT
ORGANIZER:Mailto:A@example.com
ATTENDEE;ROLE=CHAIR;PARTSTAT=ACCEPTED;CN=BIG A:Mailto:A@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=B:Mailto:B@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=C:Mailto:C@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=Hal:Mailto:D@example.com
ATTENDEE;RSVP=FALSE;TYPE=ROOM:conf_Big@example.com
ATTENDEE;ROLE=NON-PARTICIPANT;RSVP=FALSE:Mailto:E@example.com
DTSTAMP:19980611T193000Z
DTSTART:19980701T200000Z
DTEND:19980701T2200000Z
SUMMARY:Conference
DESCRIPTION:We need to discuss the project schedule. Please come prepared.
LOCATION:The Big Conference Room
UID:calsrv.example.com-873970198738777@example.com
SEQUENCE:1
STATUS:CONFIRMED
END:VEVENT
```

CAPI_STORE_DELPROP

This mode can only be used when modifying existing instances of an event. The non-indexing properties are deleted from the event. Some properties may appear many times. To delete one of these an exact match is required on the property, and its value, but not necessarily its parameters. For properties like "DURATION", which do not have a text value, the string output by CAPI must be matched. To delete all appearances of a particular property the property name only is specified, with no value and no parameters.

Example: Call CAPI_StoreEvent with the following ICAL stream:

```
BEGIN:VCALENDAR
PRODID:-//ACME//NONSGML DesktopCalendar//EN
VERSION:2.0
BEGIN:VEVENT
LOCATION:The Big Conference Room
SUMMARY:Bla Bla Bla
UID:calsrv.example.com-873970198738777@example.com
END:VEVENT
END:VCALENDAR
```

Upon successful completion of the CAPI_StoreEvent call, the LOCATION property will be deleted. The SUMMARY property will not be touched because it does not match the current summary. The calendar store copy will be as follows:

```
BEGIN:VEVENT
ORGANIZER:Mailto:A@example.com
ATTENDEE;ROLE=CHAIR;PARTSTAT=ACCEPTED;CN=BIG A:Mailto:A@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=B:Mailto:B@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=C:Mailto:C@example.com
ATTENDEE;RSVP=TRUE;TYPE=INDIVIDUAL;CN=Hal:Mailto:D@example.com
ATTENDEE;RSVP=FALSE;TYPE=ROOM:conf_Big@example.com
ATTENDEE;ROLE=NON-PARTICIPANT;RSVP=FALSE:Mailto:E@example.com
DTSTAMP:19980611T190000Z
DTSTART:19980701T200000Z
DTEND:19980701T2100000Z
SUMMARY:Conference
UID:calsrv.example.com-873970198738777@example.com
SEQUENCE:0
STATUS:CONFIRMED
END:VEVENT
```

Parameters:

- in_session : login session handle
- in_handles : array of handles to store into
- in_numHandles : number of handles in in_handles
- io_status : array (preallocated) to hold 1 status / handle
- in_flags : Flags modifying behavior:
  - CAPI_NOTIFY_EMAIL
  - CAPI_NOTIFY_SMS
  - CAPI_STORE_REPLACE (in low 2 bits)
  - CAPI_STORE_UPDATE (in low 2 bits)
  - CAPI_STORE_DELPROP (in low 2 bits)

Parameters:

in_stream : stream for CAPI to read data from

Returns:

CAPIStatus

3

Configuration Settings

This chapter contains information on the following configuration parameters that can be supplied to the Oracle Calendar API through the CAPI_SetConfigFile function.

client_name

Used to set the application name that will be visible in the server stats.

Section:

CAPI

Values:

any string

Default Value:
""

client_version

Used to set the application version that will be visible in the server stats.

Section:
CAPI

Values:
any string

Default Value:
""

cncachesize

Used to set the maximum number of entries to hold in the common name cache.

Section:
CACHE

Values:
[0..U32MAX]

Default Value:
512

emailcachesize

Used to set the maximum number of entries to hold in the email address cache.

Section:
CACHE

Values:
[0..U32MAX]

Default Value:
512

itemcachesize

Used to set the maximum number of entries to hold in the item record cache.

Section:
CACHE

Values:
[0..U32MAX]

Default Value:
256

log_activity

Used to enable "activity" (high-level) logging.

Section:
LOG

Values:
true / false

Default Value:
false

See also:
log_modulesinclude

log_modulesinclude

Used to control which modules have logging enabled.

This should be set to "{CAPI}", otherwise no logging will be performed even if it is enabled (e.g. via log_activity = true)

Section:
LOG

Values:
"" or "{ CAPI }"

Default Value:
""

tzcachesize

Used to set the maximum number of entries to hold in the timezone record cache.

Section:
CACHE

Values:
[0..U32MAX]

Default Value:
256

4

Types, Constants and Capabilities

Types

Typedefs

```
typedef void *  CAPISession
typedef void *  CAPIHandle
typedef void *  CAPIStream
typedef unsigned long  CAPIStatus
typedef unsigned long  CAPIFlag
typedef unsigned long CAPIFlag
```

CAPIFlag Constants

CAPI_FLAG_FETCH_EXCLUDE_APPOINTMENTS

Used with CAPI_FetchEvent* calls to exclude regular meetings (appointments).

CAPI_FLAG_FETCH_EXCLUDE_DAILYNOTES

Used with CAPI_FetchEvent* calls to exclude daily notes.

CAPI_FLAG_FETCH_EXCLUDE_DAYEVENTS

Used with CAPI_FetchEvent* calls to exclude day events.

CAPI_FLAG_FETCH_EXCLUDE_HOLIDAYS

Used with CAPI_FetchEvent* calls to exclude holidays.

CAPI_FLAG_NONE

Used to select the default behaviour.

CAPI_FLAG_STORE_DELPROPS

Used with CAPI_Store* functions to specify that the supplied properties are to be cleared or deleted on the server.

CAPI_FLAG_STORE_MODPROPS

Used with CAPI_Store* functions to specify that the supplied properties are to be modified on the server without changing other properties (where possible).

Capabilities

The following capabilities can be requested via CAPI_GetCapabilities.

Typedefs

```
typedef long  CAPICapabilityID
```

CAPI_CAPAB_ABOUT_BOX

Returns information about CAPI.

CAPI_CAPAB_AUTH

Returns the authentication mechanisms supported by the server (e.g. "cs-standard,gssapi:kerberos5,sasl:KERBEROS_V4"). A server connection must exist to read this capability.

CAPI_CAPAB_CAPI_VERSION

Returns the CAPI version as a string. (e.g. "2.5.0")

CAPI_CAPAB_COMP

Returns the compression mechanisms supported by the server (e.g. "cs-simple,none"). A server connection must exist to read this capability.

CAPI_CAPAB_ENCR

Returns the encryption mechanisms supported by the server (e.g. "cs-light,none"). A server connection must exist to read this capability.

CAPI_CAPAB_MAXDATE

Returns the largest date which CAPI can handle ("20371129").

CAPI_CAPAB_SERVER_VERSION

Returns the server version as a string. (e.g. "5.5"). A server connection must exist to read this capability.

CAPI_CAPAB_UNSUPPORTED_ICAL_COMP

Returns a comma delimited list of iCal components which CAPI does not process. ("VJOURNAL,VFREEBUSY")

CAPI_CAPAB_UNSUPPORTED_ICAL_PROP

Returns a comma delimited list of iCal properties which CAPI does not process. ("GEO,COMMENT"). A server connection must exist to read this capability.

CAPI_CAPAB_VERSION

Same as CAPI_CAPAB_CAPI_VERSION.

5

Status codes

This chapter documents all CAPI_Status values that may be returned by CAPI functions, in alphabetical order. The functions CAPI_GetStatusString and CAPI_GetStatusLevels may be useful when interpreting CAPI_Status values.

CAPI_STAT_API

API class status.

CAPI_STAT_API_BADPARAM

A bad parameter was passed.

CAPI_STAT_API_CALLBACK

There was a problem with a callback.

CAPI_STAT_API_CALLBACK_ERROR

The callback returned an error, which is returned in bit field 5.

CAPI_STAT_API_FLAGS

Bad flags were passed.

CAPI_STAT_API_HANDLE

There was a problem with a handle.

CAPI_STAT_API_HANDLE_BAD

The passed handle was corrupt.

CAPI_STAT_API_HANDLE_NOTNULL
The passed handle was not null.

CAPI_STAT_API_HANDLE_NULL
The passed handle was null.

CAPI_STAT_API_NULL
A null pointer was passed.

CAPI_STAT_API_SESSION
There was a problem with a session.

CAPI_STAT_API_SESSION_BAD
The passed session was corrupt.

CAPI_STAT_API_SESSION_NOTNULL
The passed session was not null.

CAPI_STAT_API_SESSION_NULL
The passed session was null.

CAPI_STAT_API_STREAM
There was a problem with a stream.

CAPI_STAT_API_STREAM_BAD
The passed stream was corrupt.

CAPI_STAT_API_STREAM_NOTNULL
The passed stream was not null.

CAPI_STAT_API_STREAM_NULL
The passed stream was null.

CAPI_STAT_DATA
Data class status.

CAPI_STAT_DATA_COOKIE
Information about the supplied cookie.

CAPI_STAT_DATA_DATE
Information about a date.

CAPI_STAT_DATA_DATE_FORMAT
The format of the date data is incorrect.

CAPI_STAT_DATA_DATE_INVALID
A specified date is invalid (e.g. Feb 30th)

CAPI_STAT_DATA_DATE_OUTOFRANGE
A specified date is out of the range supported by this implementation.

CAPI_STAT_DATA_DATE_RANGE
The date range is incorrect.

CAPI_STAT_DATA_EMAIL
Information about email.

CAPI_STAT_DATA_EMAIL_NOTSET
No email address is set on the server for 1 or more users / resources.

CAPI_STAT_DATA_ENCODING
Information about the encoding of supplied data.

CAPI_STAT_DATA_HOSTNAME
Information about a hostname.

CAPI_STAT_DATA_HOSTNAME_FORMAT
The format of the hostname string was wrong.

CAPI_STAT_DATA_HOSTNAME_HOST
The hostname string could not be resolved to a host.

CAPI_STAT_DATA_HOSTNAME_SERVER

No server could be found on the specified host and port.

CAPI_STAT_DATA_ICAL

Information about iCalendar data.

CAPI_STAT_DATA_ICAL_COMPEXTRA

An extra component was encountered. Either multiple specifications of a component which should only appear once, or a component which should not appear.

CAPI_STAT_DATA_ICAL_COMPMISSING

An expected or required component was missing.

CAPI_STAT_DATA_ICAL_COMPNAME

There was a problem with a component name.

CAPI_STAT_DATA_ICAL_COMPVALUE

There was a problem with what a component contained.

CAPI_STAT_DATA_ICAL_FOLDING

There was a problem in the line folding.

CAPI_STAT_DATA_ICAL_IMPLEMENT

A problem with this particular iCalendar implementation.

CAPI_STAT_DATA_ICAL_LINEOVERFLOW

One of the iCal data lines was too long, breaching the iCalendar spec (RFC 2445).

CAPI_STAT_DATA_ICAL_NONE

The provided data was not iCalendar data.

CAPI_STAT_DATA_ICAL_OVERFLOW

There was an overflow when parsing the iCalendar data. This is caused by an internal limitation of the iCalendar library, and not not by a breach of the spec.

CAPI_STAT_DATA_ICAL_PARAMEXTRA

An extra parameter was encountered. Either multiple specifications of a parameter which should only appear once, or a parameter which should not appear.

CAPI_STAT_DATA_ICAL_PARAMMISSING

An expected or required parameter was missing.

CAPI_STAT_DATA_ICAL_PARAMNAME

There was a problem with a parameter name.

CAPI_STAT_DATA_ICAL_PARAMVALUE

There was a problem with a parameter value.

CAPI_STAT_DATA_ICAL_PROPEXTRA

An extra property was encountered. Either multiple specifications of a property which should only appear once, or a property which should not appear.

CAPI_STAT_DATA_ICAL_PROPMISSING

An expected or required property was missing.

CAPI_STAT_DATA_ICAL_PROPNAME

There was a problem with a property name.

CAPI_STAT_DATA_ICAL_PROPVALUE

There was a problem with a property value.

CAPI_STAT_DATA_ICAL_RECURMODE

There was a problem with the recurrence specification. The rules laid out in the description of CAPI_StoreEvent were breached.

CAPI_STAT_DATA_MIME

Information about MIME data.

CAPI_STAT_DATA_MIME_CHARSET

There was a problem with a parameter name.

CAPI_STAT_DATA_MIME_COMMENT
A comment could not be parsed.

CAPI_STAT_DATA_MIME_ENCODING
The encoding specified in the MIME object is not supported.

CAPI_STAT_DATA_MIME_FOLDING
There was a problem with a parameter name.

CAPI_STAT_DATA_MIME_HEADER
A header could not be parsed.

CAPI_STAT_DATA_MIME_IMPLEMENT
A restriction specific to this MIME implementation was breached.

CAPI_STAT_DATA_MIME_IMPLEMENT_NESTING
The MIME object was nested too deeply.

CAPI_STAT_DATA_MIME_LENGTH
One of the header lines was too long.

CAPI_STAT_DATA_MIME_NOICAL
No MIME parts were found whose headers indicated that they contain iCalendar data.

CAPI_STAT_DATA_MIME_NONE
No MIME data was found.

CAPI_STAT_DATA_MIME_OVERFLOW
There was a problem with a parameter name.

CAPI_STAT_DATA_UID
Information about a UID.

CAPI_STAT_DATA_UID_FORMAT
The format of the UID string was wrong.

CAPI_STAT_DATA_UID_NOTFOUND

An event with the supplied UID could not be found.

CAPI_STAT_DATA_UID_RECURRENCE

The specified recurrence could not be found.

CAPI_STAT_DATA_USERID

Information about a userid.

CAPI_STAT_DATA_USERID_EXT

There was a problem with the Extended part of the UserId string.

CAPI_STAT_DATA_USERID_EXT_CONFLICT

Either userid AND X.400 were specified, or both a node and a calendar domain were specified.

CAPI_STAT_DATA_USERID_EXT_FORMAT

The format of the extended string was bad.

CAPI_STAT_DATA_USERID_EXT_INIFILE

There was a problem with the inifile.

CAPI_STAT_DATA_USERID_EXT_MANY

Multiple users were identified by the string.

CAPI_STAT_DATA_USERID_EXT_NODE

The specified node could not be found.

CAPI_STAT_DATA_USERID_EXT_NONE

No users were identified by the string.

CAPI_STAT_DATA_USERID_FORMAT

The format of the UserId string was wrong.

CAPI_STAT_DATA_USERID_ID

There was a problem with the Id part of the UserId string.

CAPI_STAT_LIBRARY
Library class status.

CAPI_STAT_LIBRARY_IMPLEMENTATION
The feature is not fully implemented.

CAPI_STAT_LIBRARY_INTERNAL
An internal error occured in the library.

CAPI_STAT_LIBRARY_INTERNAL_COSMICRAY
Something completely unexpected happened internally.

CAPI_STAT_LIBRARY_INTERNAL_DATA
There was a corruption of data in the library.

CAPI_STAT_LIBRARY_INTERNAL_EXPIRY
The function has expired in this library.

CAPI_STAT_LIBRARY_INTERNAL_FUNCTION
The library miscalled a function.

CAPI_STAT_LIBRARY_INTERNAL_OVERFLOW
Some internal maximum was exceeded.

CAPI_STAT_LIBRARY_INTERNAL_PROTOCOL
The library abused a protocol.

CAPI_STAT_LIBRARY_INTERNAL_UNKNOWN_EXCEPTION
CAPI received an unknown C++ exception.

CAPI_STAT_LIBRARY_INTERNAL_UNKNOWN_LIBRARY_ERRCODE
Failed to map an error code from a dependant library.

CAPI_STAT_LIBRARY_SERVER
A limitation of or occurence on the server.

CAPI_STAT_LIBRARY_SERVER_BUSY

The server cannot service the request right now because it is busy.

CAPI_STAT_LIBRARY_SERVER_SUPPORT

The server does not provide support.

CAPI_STAT_LIBRARY_SERVER_SUPPORT_CHARSET

There is no support for the required character set.

CAPI_STAT_LIBRARY_SERVER_SUPPORT_STANDARDS

There is no support for CAPI on this server.

CAPI_STAT_LIBRARY_SERVER_SUPPORT_UID

There is no support for storing UIDs.

CAPI_STAT_LIBRARY_SERVER_USERDATA

There is some problem with user data on the server.

CAPI_STAT_OK

Operation completed successfully. Value 0.

CAPI_STAT_SECUR

Security class status.

CAPI_STAT_SECUR_LOGON

There was a security error on logon.

CAPI_STAT_SECUR_LOGON_AUTH

Logon authentication failed.

CAPI_STAT_SECUR_LOGON_LOCKED

The specified account is locked.

CAPI_STAT_SECUR_LOGON_LOCKED_RESOURCE

Logon is locked for resources.

CAPI_STAT_SECUR_LOGON_LOCKED_SYSOP
Logon is locked for Sysops.

CAPI_STAT_SECUR_READ
There was a security error on read.

CAPI_STAT_SECUR_READ_ALARM
There was a security error reading alarm data.

CAPI_STAT_SECUR_READ_PROPS
There was a security error reading properties.

CAPI_STAT_SECUR_SERVER
There was a security error in the server.

CAPI_STAT_SECUR_SERVER_LICENSE
There was a licensing error on the server.

CAPI_STAT_SECUR_SERVER_SET_IDENTITY_SYSOP
The server requires a SetIdentity call on the sysop logon to perform the operation.

CAPI_STAT_SECUR_WRITE
There was a security error on write.

CAPI_STAT_SECUR_WRITE_AGENDA
There was a security error writing to an agenda.

CAPI_STAT_SECUR_WRITE_EVENT
There was a security error writing to an event.

CAPI_STAT_SERVICE
Service class status.

CAPI_STAT_SERVICE_ACE
There was a problem caused by one of the ACE plugins.

CAPI_STAT_SERVICE_ACE_LOAD
Required ACE plugin could not be loaded.

CAPI_STAT_SERVICE_ACE_SUPPORT
Requested ACE option not supported.

CAPI_STAT_SERVICE_FILE
There was a problem with system file services.

CAPI_STAT_SERVICE_FILE_CLOSE
There was a problem closing a file.

CAPI_STAT_SERVICE_FILE_DELETE
There was a problem deleting a file.

CAPI_STAT_SERVICE_FILE_MODE
There was a problem with the read or write mode for a file.

CAPI_STAT_SERVICE_FILE_OPEN
There was a problem opening a file.

CAPI_STAT_SERVICE_FILE_READ
There was a problem reading from a file.

CAPI_STAT_SERVICE_FILE_TEMP
There was a problem allocating a temporary file.

CAPI_STAT_SERVICE_FILE_WRITE
There was a problem writing to a file.

CAPI_STAT_SERVICE_LIBRARY
There was a problem with the standard library services.

CAPI_STAT_SERVICE_MEM
There was a problem with system memory services.

CAPI_STAT_SERVICE_MEM_ALLOC
Could not allocate memory.

CAPI_STAT_SERVICE_NET
There was a problem with network services.

CAPI_STAT_SERVICE_NET_TIMEOUT
Timeout while waiting for network services.

CAPI_STAT_SERVICE_THREAD
There was a problem with system thread services.

CAPI_STAT_SERVICE_TIME
There was a problem with the standard time services.

CAPI_STAT_SERVICE_TIME_GMTIME
GMTime could not be obtained.

CAPI_STATMODE_FATAL