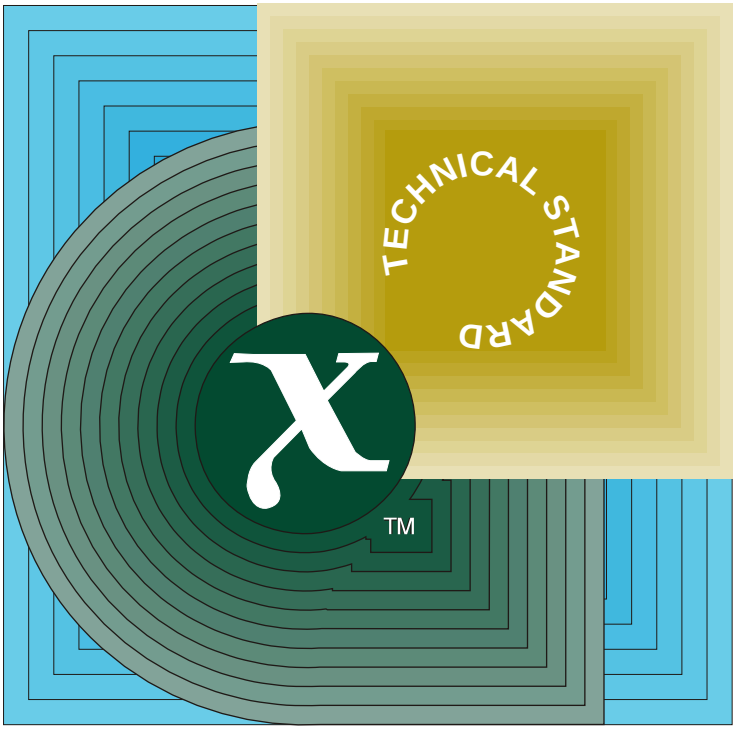
Technical Standard

Calendaring and Scheduling API (XCS)



THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specification

Calendaring and Scheduling API (XCS)

X/Open Company Ltd.

X/Open CAE Specification

Calendaring and Scheduling API (XCS)

ISBN: 1-85912-076-8

X/Open Document Number: C321

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

List of Figures

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

This document is an X/Open CAE Specification. It defines an application programming interface (API) to calendaring and scheduling services. The API enables application programmers to integrate calendaring and scheduling into their applications.

This specification is included in the X/Open Common Desktop Environment (XCDE) family of specifications, which define a common graphical user interface environment based on the X Windows System. Other specifications in the XCDE family are:

- the XCDE specification 2-volume set, comprising the Definitions & Infrastructure (XCDI) and the Services & Applications (XCSA) specifications
- the Motif (1.2) Toolkit API specification
- the X Windows System (X11R5), which comprises a 4-volume set (X Protocols, X Library, X Toolkit, and File Formats & Application Conventions).

The XCS API can also be implemented and used in environments other than XCDE.

Audience

This document is directed toward calendaring and scheduling service developers who might wish to support such an application program interface. This document may also guide application developers in understanding implementation-independent features of the Calendaring and Scheduling API. The application developers must follow manuals provided by the system they are using for calendaring support.

Structure

- **Chapter 1** introduces the scope and purpose of this XCS API
- **Chapter 2** presents the functional architecture
- **Chapter 3** describes the Data Structures
- **Chapter 4** defines the functional interface
- **Chapter 5** presents the C Declaration summary

- **Chapter 6** describes extensions
- **Chapter 7** gives platform-specific information
- **Chapter 8** defines the extended and recurrence rule grammar.

A glossary and index are provided.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — external variables, such as *errno*
  - — functions; these are shown as follows: *name*( ). Names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C construct.
- The notation [ABCD] is used to identify a return value ABCD, including if this is an an error value.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items.

Trade Marks

UNIX® is a registered trade mark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of X/Open Company Limited.

Acknowledgements

X/Open acknowledges the contribution by the XAPI Association of the base document for this Specification. It is published jointly by X/Open and the XAPI Association.

XAPI Association contributors included:

Bill Abbot, Lotus Development
Duane Buss, WordPerfect
Hanford Choy, Novell
David Cornfield, Microsoft
David Curley, DAC Associates
Frank Dawson, IBM
Bill Drummy, CE Software
Anik Ganguly, Campbell Services, Inc.
Steve Karlson, Microsystems Software, Inc.
Brian Kress, LinkAge Office Information Solutions
Alison Noble, Attachmate
Ravi Pandya, Arabesque Software
Roger Wiens, Lotus
Yvonne Tso, SunSoft

Referenced Documents

The following documents are referenced in this specification:

ANSI C
: American National Standard for Information Systems: standard X3.159-1989, Programming Language C.

ISO 3166
: ISO 3166: 1988, Codes for the Representation of Names of Countries, Bilingual edition.

ISO 639
: ISO 639: 1988, Codes for the Representation of Names of Languages, Bilingual edition.

ISO 8601
: ISO 8601: 1988, Data Elements and Interchange Formats — Information Interchange — Representation of Dates and Times.

ISO 9070
: ISO 9070: 1990-02-01, Information processing - SGML support facilities - Registration procedures for public text owner identifiers.

Chapter 1

Introduction

This chapter introduces the X/Open Calendaring and Scheduling (XCS) Application Program Interface (API) and its specifications. It indicates the purpose of the interface, provides an overview of it, provides document references, explains the level of abstraction of the interface, defines C naming conventions, and specifies conformance requirements.

1.1 Purpose

The purpose of this document is to specify a high-level Calendaring and Scheduling (CS) Application Program Interface (API) that can be supported by calendaring and scheduling services. The API is intended to enable application programmers to easily integrate calendaring and scheduling into their applications, facilitating the creation of calendar-enabled applications. This document is directed toward calendaring and scheduling service developers who might wish to support such an application program interface. This document may also guide application developers in understanding implementation-independent features of the Calendaring and Scheduling API. The application developers must follow manuals provided by the system they are using for calendaring support.

1.2 Overview

Groupware is a term that has been used to describe the area of information technology that provides for computer-based interpersonal collaboration. One category of groupware is the set of services that provides calendaring and scheduling capabilities. The Calendaring and Scheduling Application Program Interface (XCS API) is intended to define a set of high-level functions for calendar-enabled applications to access the varied features of such a calendaring and scheduling service. This interface is designed to be independent of the actual calendaring and scheduling implementation. The interface will support the functions for adding, deleting, modifying, and reading a calendar and the entries within a calendar. In addition, the interface will support the search for free time intervals within a calendar. This is achieved through generic definition of capabilities for calendaring and scheduling, plus a mechanism for defining extensions, which can be used to invoke implementation-specific features of a function. The interface is also designed to be independent of the operating system and underlying hardware used by the calendaring service. Another important consideration in the design of this API is to minimise the number of function calls needed. There are a single set of functions for managing multiple types of calendar entries, rather then a different set for each type of calendar entry. The XCS API is designed to be complementary to other API's, such as the Common Messaging Call (CMC) API. The XCS interface is designed to allow a common interface to a calendaring and scheduling service. For each XCS implementation, the view/capabilities presented by XCS must be mapped to the view/capabilities of the underlying calendaring service.

1.3 Functional Views

This document defines the XCS API in terms of two functional views. It defines an abstract, functional interface that is independent of any particular programming language. In addition, it defines an interface binding of that abstract functional view to the American National Standard for the C Programming Language. The abstract functional interface is included to guide the development of other language- specific specifications, for example C++. For readability, the specifications of the functional and C interfaces are combined. In Section 3, the XCS data structures are described generically, but include a C declaration. In Section 4, the XCS functions are specified generically, but include a synopsis written in C. For clarity, constants and error codes throughout this specification are written in the C syntax described below.

1.4 Conformance

In order for an implementation of the Calendaring and Scheduling API to conform to this specification it must meet the following criteria:

- All functions and data structures must be implemented as defined. Statements elsewhere in the specification which describe features as optional or with exceptions take precedence over this criterion.
- The implementation must be able to add and delete calendar entries.
- All extensions are optional. Vendors are encouraged to support the XCS-defined standard extension set specified in this document. It is further encouraged that standard extension sets are developed for any proprietary or non-proprietary calendaring services for which a XCS interface is provided, to accommodate features specific to that calendaring.
- Minimum conformance for an extension set is defined by the creator of the extension set.

1.5 C Naming Conventions

The identifier for an element of the C interface is derived from the generic name of the element and its associated data type, as specified in the following table. The generic name is prefixed with the character string in the second column of the table, alphabetic characters are converted to the case in the third column.

Element Type	Prefix	Case
Data type	CSA_	Lower
Data value	CSA_	Upper
Function	csa_	Lower
Function argument	none	Lower
Function result	none	Lower
Constant	CSA_	Upper
Error	CSA_E_	Upper
Macro	CSA_	Upper
Reserved for extension sets	CSA_XS_	any
Reserved for extensions	CSA_X_	any
Reserved for use by implementors	CSAP	any
Reserved for vendor function extensions	csa_x_	Lower
Structure Tag	CSA_TAG_	Upper

Table 1-1 Derivation of C Naming Conventions

Elements with the prefix ‘‘CSAP’’ (any case) are reserved for internal proprietary use by implementors of the XCS service. They are not intended for direct use by programs written using the XCS interface. The prefixes ‘‘CSA_XS_’’, ‘‘CSA_X_’’ (in either upper or lower case), and ^Scsa_x_^T are reserved for extensions of the interface by vendors or groups. These extensions may be extensions to the base set of functions defined by this specification. For constant data values, there is usually an additional string appended to ‘‘CSA_’’ to indicate the data structure or function to which the constant data value pertains.

Chapter 2

Functional Architecture

This chapter describes the functional architecture of services supporting the XCS API. It provides an abstract implementation model, an abstract data model, and a functional overview.

2.1 Implementation Model

The following abstract implementation model is provided as a reference aid to facilitate understanding the scope of the application programming interface defined by this specification. The XCS interface is defined between a calendar-enabled application and a calendaring service. All functions in this interface are designed to be independent of the calendaring service. However, the API does allow protocol-specific extensions to the common functions to be invoked through the use of extensions (see Section 2.4 on page 11). The XCS interface is depicted in Figure 2-1.

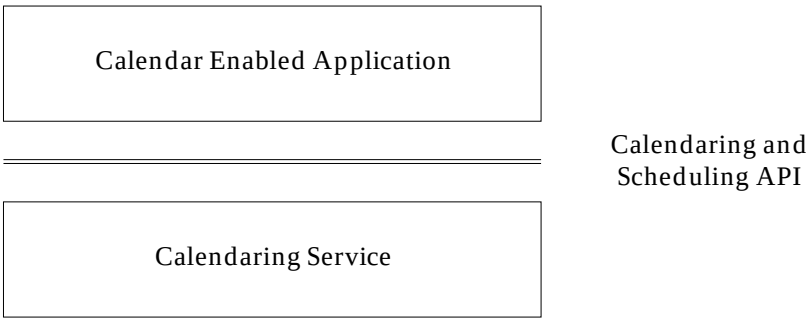


Figure 2-1 Positioning of the Calendaring and Scheduling API

The model of the XCS interface can be divided into three components:

- administration
- calendar management
- entry management.

These components are shown in Figure 2-2.

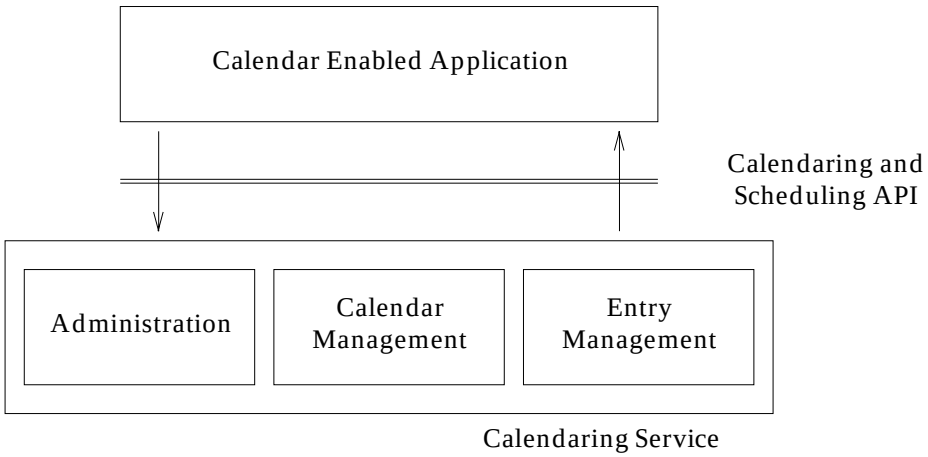


Figure 2-2 Components of the Calendaring and Scheduling API

Access to the calendaring service is established by way of a calendaring session. The session provides for a valid connection to the calendaring service and assists in ensuring the integrity of the calendaring information maintained by the service. A calendar-enabled application logs on to an individual calendar within the calendaring service to establish a valid session or connection. The session is terminated by the calendar-enabled application by logging off from the calendar.

The calendaring service maintains one or more calendars. The calendar service provides some level of administration support for these calendars. A calendar-enabled application might access a list of the calendars maintained by a particular calendar service. In addition, the calendar service might provide support for the archive and restore of calendar information into some implementation specific, persistent format. Where a calendar service provides support for maintaining more than one calendar, there are support functions defined for creating and deleting additional calendars. In addition, there are functions provided to support administering the characteristics of the calendar.

The majority of the functions within the XCS interface deal with the management of individual calendar entries. Calendar entries may be either *events*, *to-dos* or *memos*. Entries can be added, deleted, updated, and read from a particular calendar. In addition, a calendar-enabled application can search for free time intervals on a particular set of calendars. A calendar-enabled application can also add reminders to calendar entries.

2.2 Data Model

The XCS interface is an access method to a conceptual back-end store of calendaring information maintained by a calendaring service. A common data model is helpful in visualising the components of the calendaring information maintained by a calendaring service.

The data model is based on the concept of a *calendar entity*. The calendar is represented by a named collection of administrative *calendar attributes* and *calendar entries*. A calendar is owned by an individual user. The user could represent a person, a group of people, or a resource.

The calendar attributes are a set of named values that may represent common, implementation specific, or application specific administrative characteristics about the calendar. For example, the country, language, time zone, name, owner, and access rights to the calendar can be specified in individual calendar attributes.

The calendar entries are the primary components of a calendar. There are three classes of calendar entries. These include *events*, *to-dos* and *memos*. Calendar entries are represented by a uniquely named collection of *entry attributes*. The entry attributes are a set of named values that represent common, implementation specific, or application specific characteristics of the calendar entry. For example, an event might contain a start and end date and time, a list of attendees, a description, and a subtype. A to-do might contain a date it was created, a due date, a priority, and a status. A memo might contain a date it was created and a text content or description.

The calendar atttributes and entry attributes consist of a name, type, and value tuple. The attributes can be singular or compound values. Some attributes might consist of a list of attributes.

There is also the provision for extension of the common attributes defined by this specification. Implementations may define implementation specific attributes. In addition, some implementations may provide the capability for applications to define application specific attributes.

The accessibility of a calendar to an individual user can be controlled by the *access rights* given that user. Access rights are paired with a calendar user. A calendar user can be an individual, group or resource. The access rights are maintained in an *access list*. The access list is a particular calendar attribute. The access rights are individually controlled and can be accumulated to define a range of accessibility of a user to a calendar and its entries. The access rights can also be specified in terms of one of three access roles. These include the role of the *owner* of a calendar, the *organiser* of a particular entry within the calendar, and the *sponsor* of a particular entry within the calendar. The owner role permits the user to do anything to the calendar or calendar entries that the owner of the calendar can do. This includes deleting the calendar, viewing, inserting, and changing calendar attributes, adding and deleting calendar entries, and viewing, inserting, and changing entry attributes. The organiser role permits the user to delete the entry or view and change entry attributes for those calendar entries which the user is specified as the organiser. The organiser defaults to the calendar user that created the entry. The sponsor role permits the user to delete the entry or view and change entry attributes for those calendar entries which the user is specified as the sponsor. The sponsor is the calendar user that effectively owns the calendar event or to-do. In addition to these roles, an access right can be set to limit access to free time searches, view, insert or change calendar attributes, or view, insert or change entries; dependent on their classification as **public**, **confidential private**. The entry classification acts as a secondary filter on accessibility.

2.3 Functional Overview

The XCS interface supports three principle types of tasks: administration, calendar management, and entry management.

Some implementations may provide support for the user interface (UI) common extension. This extension provides the capability for supporting user interface dialogs from within individual XCS functions to resolve parameter values for functions and logon to the calendar service.

2.3.1 Administration

With minor exception, the XCS function calls occur within the context of a calendar session. The calendar session is a logical connection between the calendar-enabled application and a particular calendar maintained by the calendaring service. A session is established with a call to the XCS *Logon*( ) function and terminated with a call to the XCS *Logoff*( ) function. The context of the session is represented by a session handle. This provides a token in each of the XCS functions to distinguish one calendar session from another. The XCS *Logon*( ) function also authenticates the user to the calendaring service and sets session attributes. Session attributes include character set to be used for the session and the caller's XCS product version number. Currently, there is no support for sharing calendar sessions among applications.

The XCS *LookUp*( ) function is used to resolve a calendar user information needed to logon to a calendar service. The function can be called with the user name of a calendar user; with the user type and the calendar address of the calendar user being returned.

To look up calendar user names in the XCS implementation, the calendar-enabled application establishes a session through the XCS *Logon*( ) function. The application then uses XCS *LookUp*( ) to translate a user-friendly name into a calendar address. Memory allocated by the system for structures is released by passing the associated memory pointer to the XCS *Logoff*( ) function.

The XCS *ListCalendars*( ) function is used to list the names of the calendars managed by a particular calendar service. If the user specifies a NULL calendar service name, then the calendars manged by the default calendar service will be listed.

The XCS *QueryConfiguration*( ) function is used to list information about the current calendar service configuration. This information can include the character set, line terminator characters for text strings, default service name, default authorization user identifier for the specified calendar service, an indicator of whether a password is needed to authenticate the user identifier, an indicator of whether the common extensions for user interface dialogs is supported, the version of the XCS implementation and the XCS specification supported by the implementation.

The XCS *Save*( ) and *Restore*( ) functions are used to archive and retrieve calendar information into and from an implementation specific file format. This capability is useful to management of large amounts of calendar information over time.

The XCS implementation provides support for managing the memory for calendar objects and attributes that are returned by the service. The XCS *Free*( ) function is used to free up this memory after it is no longer needed. It is the responsibility of the calendar user to free up the memory allocated and managed by the calendar service.

2.3.2 Calendar Management

There are several calendar management functions that are provided by the XCS interface. For those implementation that support multiple calendars per calendar service, the calendar-enabled application can add or delete calendars. The XCS *DeleteCalendar*( ) function is used to delete calendars. The XCS *AddCalendar*( ) function is used to add calendars to the service.

The application can also list, read and update calendar attributes. The XCS *ListCalendarAttributes*( ), *ReadCalendarAttributes*( ) and *UpdateCalendarAttributes*( ) functions are used to perform these respective functions. For calendar services supporting asynchronous notification of calendar activity, the application can register callback functions for receiving notification of a calendar logon, calendar deletion, update of a calendar attribute, addition of a new calendar entry, deletion of a calendar entry, and update of a calendar entry. The callback function is only registered for the duration of the calendar session. In any case, this information may be invaluable for some calendar administration applications.

2.3.3 Entry Management

The XCS interface has a robust set of functions for managing calendar entries. The context of a calendar entry and a calendar session is maintained by the entry handle. This provides a token in the XCS functions to distinguish one calendar entry from another. The entry handle is returned by the *AddEntry*( ) and *ListEntries*( ) functions. The entry handle is valid for the duration of the calendar session or until the entry is deleted or updated. The entry handle also becomes invalid when it is freed by a call to *Free*( ).

The XCS *AddEntry*( ) function is used to add new entries to a calendar. The XCS *DeleteEntry*( ) function is used to delete an entry in a calendar. The XCS *ListEntry*( ) function is used to enumerate the calendar entries that match a particular set of entry attribute criteria. The XCS *ReadEntry*( ) function is used to get either all or a set of entry attribute values associated with a particular calendar entry.

To add an entry to a calendar, a calendar-enabled application must first establish a session with the calendaring service through the XCS *Logon*( ) function. Then the application must invoke the XCS *AddEntry*( ) function to specify the new entry. The calendar-enabled application is responsible for *Logoff*( ) function.

The XCS *FreeTimeSearch*( ) function is used to search for blocks of available time on one or more calendars. The function returns a list of free time intervals. Free time is an interval of time that is not currently scheduled on the specified calendars.

The entry attributes in an individual calendar entry can be enumerated with the XCS *ListEntryAttributes*( ) function. The values of one or more attributes can be read with the XCS *ReadEntryAttributes*( ) function. Individual entry attributes can be modified with the XCS *UpdateEntryAttributes*( ) function. Updating the value of some attributes may have the side-effect of being propagated to other users calendars. For example, updating the status field of the attendee structure for an entry for which the calendar user is specified as an attendee will cause the status to be propagated to the corresponding calendar entry on the calendar of the sponsor of the meeting.

To reply to an event in a calendar, a calendar-enabled application must establish a session through the XCS *Logon*( ) function. The application can then enumerate calendar information through the XCS *ListEntries*( ) function. Individual calendar entries can be retrieved through the XCS *Read*( ) function. The XCS *UpdateEntryAttributes*( ) function allows the user to act on a calendar entry and propagate a reply to the sponsor of the entry. For example, reply to a meeting request. Memory allocated by the XCS implementation for retrieved calendar information is released by passing the associated memory pointers to the XCS *Free*( ) function.

The session is terminated with the XCS *Logoff*( ) function.

Some calendar entries are associated with a recurring activity. The XCS *ListEntrySequence*( ) function can be used to enumerate the other recurring calendar entries. This function will return a list of the entry handles for the recurring entries.

Many calendar services provide support for alarms or reminders to be associated with event and to-do calendar entries. Reminders can take the form of audio reminders from the terminal speaker, flashing reminders presented on the terminal screen, mail reminders sent to the calendar user, or pop-up reminders presented on the terminal screen. The calendar service manages the reminders, but it is the responsibility of the calendar application to retrieve the reminder information and act on it. The XCS *ReadNextReminder*( ) function is used to read the information about the next scheduled type of reminder.

2.4 Extensions

The major data structures and functions defined in this specification can be extended methodically through the use of extensions. Extensions are used to add additional fields to data structures and additional parameters to a function call. A standard generic data structure has been defined for these extensions. It consists of an item code, identifying the extension; an item data, holding the length of extension data or the data itself; an item reference, pointing to where the extension value is stored or NULL if there is no related item storage; and flags for the extension.

Extensions that are additional parameters to a function call may be input or output. That is, the extension may be passed as input parameters from the application to the XCS service or passed as output parameters from XCS service to the application. If an extension is an input parameter, the application allocates memory for the extension structure and any other structures associated with the extension. If an extension is an output parameter, the XCS service allocates the storage for the extension result, if necessary. In this case, the application must free the allocated storage with a XCS *Free*( ) call.

Extensions play a dual role in this specification. First, they provide a mechanism whereby features not common across all calendaring services can be accommodated. Second, they provide a mechanism to extend the specification in the future, minimising any backward-compatibility issues.

In the latter role, implementations that plan on extending the function set within the API are encouraged to identify these additional functions with an extension prefix to the function name. The sentinel character format is similar to that used for extension parameter nomenclature. For example, a vendor abc might have added a function *FindNext*( ) to its implementation. It would be identified as a function named *csa_x_abc_find_next*( ).

Use of extensions for the first reason, while very important, should be employed with caution. Reliance on features specific to particular calendaring-services limits application portability across calendaring services; also, such features may not survive a journey through a gateway in a enterprise network.

To minimise portability issues, implementors are encouraged to specify extensions as generically as possible, and to contribute these extensions as proposed additions to the XCS-defined extension set. Through this process, the CS API set will evolve in a positive direction in a manner which continues to maximise portability.

Implementations should return CSA_E_UNSUPPORTED_FUNCTION_EXT if they do not support a requested function extension, even if CSA_EXT_REQUIRED extension flag is cleared in the extension.

For more information on extension registration and the extensions defined in this document, see the section on Extensions.

Chapter 3

Data Structures

3.1 Basic Data Types

Some of the abstract data types are defined in terms of intermediate or basic data types whose precise definitions in C are system-defined. The basic data types used in the XCS interface specification include:

sint16	The positive and negative integers representable in 16 bits.
sint32	The positive and negative integers representable in 32 bits.
uint8	The non-negative integers representable in 8 bits.
uint16	The non-negative integers representable in 16 bits.
uint32	The non-negative integers representable in 32 bits.

C DECLARATION

```
typedef system-defined, e.g., int                 CSA_sint16;
typedef system-defined, e.g., long int            CSA_sint32;
typedef system-defined, e.g., unsigned char       CSA_uint8;
typedef system-defined, e.g., unsigned int        CSA_uint16;
typedef system-defined, e.g., unsigned long int   CSA_uint32;
```

3.2 Abstract Data Types

This section defines, and Table 3-1 lists, the data structures used in the XCS interface specification.

Data Type Name	Description
Access List	List of access rights structures for calendar users
Attendee List	List of attendee structures
Attribute	Attribute structure
Attribute Reference	Attribute reference structure
Boolean	A value that indicates logical true or false
Buffer	Pointer to a data item
Calendar User	Calendar user structure
Callback Function Prototype and Data Structures	Callback function prototype and data structures
Date and Time	Date and time designation
Date and Time List	List of date and time values
Date and Time Range	Range of date and time
Entry Handle	Handle for the calendar entry
Enumerated	Data type containing a value from an enumeration
Extension	Extension structure
Flags	Container for bit masks
Free Time	Free time structure
Opaque Data	Opaque data
Reminder	Reminder structure
Reminder Reference	Reminder reference structure
Return Code	Return value indicating either that a function succeeded or why it failed
Service Reference	Service reference structure
Session Handle	Handle for the calendar session
String	Character string pointer
Time Duration	Time duration

Table 3-1 XCS Data Structures

The abstract data types used in the XCS interface specification are specified in the man page descriptions in this section.

NAME

Access List - type definition for a list of access rights data value.

SYNOPSIS

```
typedef struct CSA_TAG_ACCESS_RIGHTS {
    CSA_calendar_user               *user;
    CSA_flags                        rights;
    struct CSA_TAG_ACCESS_RIGHTS    *next;
} CSA_access_rights, *CSA_access_list;
```

DESCRIPTION

A data value of this data type is used to define the access rights for a list of calendar users. An access list is a list of access rights data structures. The access rights structure consists of the following components:

user

Identifies the calendar user associated with this access entry.

rights

Specifies the access rights for this user to this calendar. The defined access rights include:

CSA_FREE_TIME_SEARCH
CSA_VIEW_PUBLIC_ENTRIES
CSA_VIEW_CONFIDENTIAL_ENTRIES
CSA_VIEW_PRIVATE_ENTRIES
CSA_INSERT_PUBLIC_ENTRIES
CSA_INSERT_CONFIDENTIAL_ENTRIES
CSA_INSERT_PRIVATE_ENTRIES
CSA_CHANGE_PUBLIC_ENTRIES
CSA_CHANGE_CONFIDENTIAL_ENTRIES
CSA_CHANGE_PRIVATE_ENTRIES
CSA_VIEW_CALENDAR_ATTRIBUTES
CSA_INSERT_CALENDAR_ATTRIBUTES
CSA_CHANGE_CALENDAR_ATTRIBUTES
CSA_ORGANIZER_RIGHTS
CSA_SPONSOR_RIGHTS
CSA_OWNER_RIGHTS

CSA_FREE_TIME_SEARCH

When set, implies that the user will be able to access the calendar entries for free time search. No access is provided to the calendar attributes.

CSA_VIEW_PUBLIC_ENTRIES

CSA_VIEW_PRIVATE_ENTRIES

CSA_VIEW_CONFIDENTIAL_ENTRIES

When set, gives the user access to list and read the calendar entries with an access class of public, private, or confidential, respectively. No access is provided to the calendar attributes.

CSA_INSERT_PUBLIC_ENTRIES

CSA_INSERT_PRIVATE_ENTRIES

CSA_INSERT_CONFIDENTIAL_ENTRIES

When set, gives the user access to add the calendar entries with an access class of public, private, or confidential, respectively. No access is provided to the calendar

attributes.

CSA_CHANGE_PUBLIC_ENTRIES

CSA_CHANGE_PRIVATE_ENTRIES

CSA_CHANGE_CONFIDENTIAL_ENTRIES
When set, gives the user access to update, and delete the calendar entries with an access class of public, private, or confidential, respectively. No access is provided to the calendar attributes.

CSA_VIEW_CALENDAR_ATTRIBUTES
When set, gives the user access to list and read the calendar attributes.

CSA_INSERT_CALENDAR_ATTRIBUTES
When set, gives the user access to add calendar attributes.

CSA_CHANGE_CALENDAR_ATTRIBUTES
When set, gives the user access to update, and delete the calendar attributes.

CSA_ORGANIZER_RIGHTS
When set gives the user access rights to read, update, or delete the entry attributes for any calendar entry for which the user is specified as the organiser.

CSA_SPONSOR_RIGHTS
When set gives the user the same access to read, update, or delete the entry attributes for any calendar entry for which the user is specified as the sponsor.

CSA_OWNER_RIGHTS
When set gives the user the same access rights as the owner of the calendar. These rights permit the user to read, update, or add either the calendar attributes or the calendar entries, regardless of their access class. The default access rights are implementation specific.

next
Points to the access entry for the next calendar user in this access list.

NAME

Attendee List - type definition for a list of attendee data values.

SYNOPSIS

```
typedef struct CSA_TAG_ATTENDEE {
    CSA_calendar_user           attendee;
    CSA_enum                    priority;
    CSA_enum                    status;
    CSA_boolean                 rsvp_requested;
    struct CSA_TAG_ATTENDEE    *next;
} CSA_attendee, *CSA_attendee_list;
```

DESCRIPTION

A data value of this data type is used to define a list of attendees. An attendee list is a list of attendee data structures. The attendee structure consists of the following components:

attendee

The name and address of the attendee.

priority

An indication of the originator's expectation of the attendee's attendance. Valid values include:

CSA_FOR_YOUR_INFORMATION
CSA_ATTENDANCE_REQUESTED
CSA_ATTENDANCE_REQUIRED
CSA_IMMEDIATE_RESPONSE

Support for all of these values is optional for conformance to this specification.

status

The current status of the attendees participation. Valid values include:

CSA_STATUS_ACCEPTED
CSA_STATUS_NEEDS_ACTION
CSA_STATUS_SENT
CSA_STATUS_TENTATIVE
CSA_STATUS_CONFIRMED
CSA_STATUS_REJECTED
CSA_STATUS_COMPLETED
CSA_STATUS_DELEGATED

Support for all of these values is optional for conformance to this specification.

rsvp_requested

Whether (CSA_TRUE) or not (CSA_FALSE) the favor of a reply is requested.

next

Points to the next attendee in the attendee list.

NAME

Attribute - type definition for a XCS calendaring attribute structure.

SYNOPSIS

```
typedef struct CSA_TAG_ATTRIBUTE_VALUE{
    CSA_enum   type;
    union {
        CSA_boolean            boolean_value;
        CSA_enumerated         enumerated_value;
        CSA_flags              flags_value;
        CSA_sint32             sint32_value;
        CSA_uint32             uint32_value;
        CSA_calendar_user      calendar_user_value;
        CSA_date_time          date_time_value;
        CSA_date_time_range    date_time_range_value;
        CSA_time_duration      time_duration_value;
        CSA_string             string_value;
        CSA_attendee_list      attendee_list_value;
        CSA_date_time_list     date_time_list_value;
        CSA_attendee_list      attendee_list_value;
        CSA_access_list        access_list_value;
        CSA_reminder          *reminder_value;
        CSA_opaque            *opaque_data_value;
    } item;
} CSA_attribute_value;
typedef struct CSA_TAG_ATTRIBUTE{
    CSA_string                 name;
    CSA_attribute_value       *value;
    CSA_extension             *attribute_extensions;
} CSA_attribute;
```

DESCRIPTION

A data value of this type is an attribute. The attributes are the primary components of the calendar and calendar entry conceptual data elements. An attribute has the following components:

name

The name of the attribute. It identifies the attribute. There are a set of defined attribute types for calendar attributes and entry attributes. Implementations may provide extensions to these types.

value

The attribute value. The attribute value is defined by the attribute_value structure. This structure is defined as a type-value pair. The value.type can include the following types:

CSA_VALUE_BOOLEAN
CSA_VALUE_ENUMERATED
CSA_VALUE_FLAGSCSA_VALUE_SINT32
CSA_VALUE_UINT32
CSA_VALUE_STRING
CSA_VALUE_CALENDAR_USER
CSA_VALUE_DATE_TIME
CSA_VALUE_DATE_TIME_RANGE
CSA_VALUE_TIME_DURATION
CSA_VALUE_ACCESS_LIST
CSA_VALUE_ATTENDEE_LIST
CSA_VALUE_DATE_TIME_LIST
CSA_VALUE_ACCESS_LIST
CSA_VALUE_REMINDER
CSA_VALUE_OPAQUE_DATA

NAME

Attribute Reference - type definition for an attribute reference data value.

SYNOPSIS

```
typedef CSA_string    CSA_attribute_reference;
```

DESCRIPTION

A data value of this data type is an attribute reference. It specifies the name of an attribute.

NAME

Boolean - type definition for a Boolean data value.

SYNOPSIS

```
typedef  CSA_uint32    CSA_boolean;
```

DESCRIPTION

A data value of this data type is a Boolean, i.e. either false or true. In the C interface, false is denoted by zero (CSA_FALSE), and true is denoted by any other integer, although the symbolic constant (CSA_TRUE) refers to the integer one specifically.

NAME

Buffer - type definition for storage space in memory of an undefined type.

SYNOPSIS

```
typedef  void *    CSA_buffer;
```

DESCRIPTION

A data value of this data type is a pointer to a storage location in memory of an undefined type. The size of a void * is specific to the platform.

NAME

Calendar User - type definition for a calendar user data value.

SYNOPSIS

```
typedef struct CSA_TAG_CALENDAR_USER {
    CSA_string        user_name;
    CSA_enum          user_type;
    CSA_string        calendar_address;
    CSA_extension    *calendar_user_extensions;
} CSA_calendar_user;
```

DESCRIPTION

A data value of this data type is a calendar user. This structure has the following components:

user_name

The friendly name or display name of the calendar user.

user_type

The type of the user. The user can either be an individual, CSA_TYPE_INDIVIDUAL, or a group, CSA_TYPE_GROUP, or a resource, CSA_TYPE_RESOURCE. Support for user types of GROUP and RESOURCE are optional for conformance to this specification.

calendar_address

The address that is acceptable to the underlying calendaring service for the calendar associated with the user. The format of the address string is not defined by this specification. It is intended to support any string notation supported by a given implementation.

NAME

Callback Data Structures - type definitions for a callback function data values.

SYNOPSIS

```
typedef struct CSA_TAG_LOGON_CB_DATA {
    CSA_calendar_user        *user;
} CSA_logon_callback_data;

typedef struct CSA_TAG_CALENDAR_DELETED_CB_DATA {
    CSA_calendar_user        *user;
} CSA_calendar_deleted_callback_data;

typedef struct CSA_TAG_CALENDAR_ATTR_UPDATE_CB_DATA {
    CSA_calendar_user        *user;
    CSA_uint32                number_attributes;
    CSA_attribute_reference  *attribute_names;
} CSA_calendar_attr_update_callback_data;

typedef struct CSA_TAG_ADD_ENTRY_CB_DATA {
    CSA_calendar_user        *user;
    CSA_opaque_data           added_entry_id;
} CSA_add_entry_callback_data;

typedef struct CSA_TAG_DELETE_ENTRY_CB_DATA {
    CSA_calendar_user        *user;
    CSA_opaque_data           deleted_entry_id;
    CSA_enum                  scope;
    CSA_date_time             date_and_time;
} CSA_delete_entry_callback_data;

typedef struct CSA_TAG_UPDATE_ENTRY_CB_DATA {
    CSA_calendar_user        *user;
    CSA_opaque_data           old_entry_id;
    CSA_opaque_data           new_entry_id;
    CSA_enum                  scope;
    CSA_date_time             date_and_time;
} CSA_update_entry_callback_data;

typedef void (*CSA callback) (
    CSA_session_handle        session,
    CSA_flags                 reason,
    CSA_buffer                call_data,
    CSA_buffer                client_data,
    CSA_extension            *callback_extensions
);
```

DESCRIPTION

Callback procedures allow the service to inform applications that something of interest has happened. Typical callbacks indicate that another session has been opened with the calendar, or that another session has changed the data in the calendar. All callback procedures are of type **csa_callback**.

Programmers writing callback procedures should consider the platform specific method that the callback is performed and of the performance impact of callback functions. Callbacks are

invoked in an implementation specific sequence by the service when either the specified callback activity occurs or the function *csa_call_callbacks*( ) is called. Effectively, the XCS application running at the time of the callback invocation will be blocked until the callback returns. Responsiveness of the XCS application will be impacted if the callback function does not return quickly.

The callback function prototype components include the following:

session
The session handle of the calendar in which the activity occured.

reason
A bitmask of flags. Exactly one bit will be set which indicates the activity performed and how to interpret the *call_data* argument. The following flags are defined:

CSA_CB_CALENDAR_LOGON
CSA_CB_CALENDAR_DELETED
CSA_CB_CALENDAR_ATTRIBUTE_UPDATED
CSA_CB_ENTRY_ADDED
CSA_CB_ENTRY_DELETED
CSA_CB_ENTRY_UPDATED

CSA_CB_CALENDAR_LOGON
If set, the callback will be invoked when the calendar is logged on by a user.

CSA_CB_CALENDAR_DELETED
If set, the callback will be invoked when a user requests that the calendar be deleted.

CSA_CB_CALENDAR_ATTRIBUTE_UPDATED
If set, the callback will be invoked whenever any of the calendars attributes are updated.

CSA_CB_ENTRY_ADDED
If set, the callback will be invoked whenever an entry is added to the calendar.

CSA_CB_ENTRY_DELETED
If set, the callback will be invoked whenever an entry is deleted from the calendar.

CSA_CB_ENTRY_UPDATED
If set, the callback will be invoked whenever any of the calendars entries are updated.

call_data
A pointer to the callback data structure specific to the activity performed. The data is dependent on the reason argument.

client_data
A pointer to the data passed to *csa_register_callback*( ) when this callback function is registered.

callback_extensions
A pointer to an array of CSA_extension structures for this callback function.

Each callback function returns a pointer to one of the callback data structures in its *call_data* argument. The structure that is returned depends on the context of the callback and is determined by the value of the reason argument, as described below.

The callback data structure is the mechanism that the calendar service uses to provide update operation-specific information to the application. Application can have additional context passed to their callback functions through the use of the *client_data* argument.

When a new entry is added, the CSA_CB_ENTRY_ADDED callback list is processed. Registered functions are called with CSA_CB_ENTRY_ADDED in the *reason* argument and a pointer to the CSA_add_entry_callback_data structure in the *call_data* argument.

When an entry is deleted, the CSA_CB_ENTRY_DELETED callback list is processed. Registered functions are called with CSA_CB_ENTRY_DELETED in the *reason* argument and a pointer to the CSA_delete_entry_callback_data structure in the *call_data* argument.

When an entry is updated, the CSA_CB_ENTRY_UPDATED callback list is processed. Registered functions are called with CSA_CB_ENTRY_UPDATED in the *reason* argument and a pointer to the CSA_update_entry_callback_data structure in the *call_data* argument.

When a calendar user logs on to a calendar, the CSA_CB_CALENDAR_LOGON callback list is processed. Registered functions are called with CSA_CB_CALENDAR_LOGON in the *reason* argument and a pointer to the CSA_logon_callback_data structure in the *call_data* argument.

When a calendar is deleted, the CSA_CB_CALENDAR_DELETED callback list is processed. Registered functions are called with CSA_CB_CALENDAR_DELETED in the *reason* argument and a pointer to the CSA_calendar_deleted_callback_data structure in the *call_data* argument.

When a calendar attribute is updated, the CSA_CB_CALENDAR_ATTRIBUTE_UPDATED callback list is processed. Registered functions are called with CSA_CB_CALENDAR_ATTRIBUTE_UPDATED in the reason argument and a pointer to the CSA_calendar_attr_update_callback_data structure in the *call_data*

In all cases, the order in which the callback functions are invoked by the service is implementation specific.

The callback data structure elements include the following:

user
The calendar user that initiated the corresponding calendar service operation.

number_attributes
The number of attribute names in the *attribute_names* argument.

attribute_names
The names of the calendar attributes that were updated.

added_entry_id
The reference identifier of the entry that was added to the calendar.

deleted_entry_id
The reference identifier of the entry that was deleted from the calendar.

old_entry_id
The reference identifier of the entry to be updated.

new_entry_id
The reference identifier of the updated entry.

scope
The scope of the delete or the update entry operation. The values are the same as defined in the *csa_delete_entry*( ) and *csa_updated_entry_attributes*( ) functions.

date_and_time
The date and time that the callback activity occurred.

NAME

Date and Time - type definition for a date and time data value.

SYNOPSIS

```
typedef CSA_string    CSA_date_time;
```

DESCRIPTION

A data value of this data type is an UTC (Coordinated Universal Time) based date and time value consistent with the combined date and time of the day representation of ISO 8601. The data value is a concatenation of the **date** and **time** representations. The character [T] is used as time designator to indicate the start of the representation of time of day in the combined date and time of day string expression. For example, ccyymmddThhmmssZ, where [cc] is the century string, [yy] is the year string, [mm] is the month string, [dd] is the day of the month string, [hh] is the hour string in a 24-hour format, [mm] is the minutes past the hour string, [ss] is the seconds past the minute string, and the character [Z] is the UTC time designator..

date

The calendar date, expressed as the complete representation, basic format, as defined in ISO 8601, clause 5.2.1.1. For example, April 14, 1985 would be represented by the string

```
19850412.
```

time

The time of the day, expressed as the equivalent Coordinated Universal Time (UTC), complete representation, basic format, as defined in ISO 8601, clause 5.3.3. For example, UTC time 20 minutes and 30 seconds past 23 hours would be represented by the string 232030Z.

NAME

Date and Time List - type definition for a list of date and time data values.

SYNOPSIS

```
typedef struct CSA_TAG_DATE_TIME_ITEM{
    CSA_date_time                    date_time;
    struct CSA_TAG_DATE_TIME_ITEM    *next;
} CSA_date_time_entry, *CSA_date_time_list;
```

DESCRIPTION

A data value of this data type is a list of UTC (Coordinated Universal Time) based date and time values. The date and time values are represented as defined by the CSA_date_time data type.

date_time

A date and time value, expressed as the complete representation, basic format, as defined in ISO 8601, clause 5.2.1.1 for the date and UTC time of day. For example, April 14, 1985 and UTC time 20 minutes and 30 seconds past 23 hours would be represented by the string:

```
I19850412T232030Z
```

next

The pointer to the next entry in the list of date and time data values.

NAME

Date and Time Range - type definition for a range of date and time data value.

SYNOPSIS

```
typedef CSA_string        CSA_date_time_range;
```

DESCRIPTION

A data value of this data type is a period of time expressed as a duration of time delimited by a specific start and a specific end. The period of time is represented as an alphanumeric string consistent with the periods of time complete representation of clause 5.5.3.1 of ISO 8601. For example, a period beginning at 30 minutes and 0 seconds past 8 hours UTC on 12 April 1985 and ending at 30 minutes and 0 seconds past 10 hours UTC on April 12, 1985 is represented by the string:

```
19850412T083000Z/19850412T103000Z
```

The start and end date and time are delimited by the solidus [/] separator character.

NAME

Entry handle - type definition for a calendar entry handle data value.

SYNOPSIS

```
typedef     system-defined, for example, uint32    CSA_entry_handle;
```

DESCRIPTION

A data value of this data type is an opaque calendar entry handle. The calendar entry handles are unique to the calendar service. The handle is created by XCS *ListEntries*( ), *AddEntry*( ), *ListEntrySequence*( ), and *UpdateEntryAttributes*( ) functions.

NAME

Enumerated - type definition for an enumerated data value.

SYNOPSIS

```
typedef     CSA_sint32        CSA_enum;
```

DESCRIPTION

A data value of this data type contains a value selected from an enumerated list.

NAME

Extension - type definition for a XCS extension structure.

SYNOPSIS

```
typedef struct CSA_TAG_EXTENSION {
    CSA_uint32    item_code;
    CSA_uint32    item_data;
    CSA_buffer    item_reference;
    CSA_flags     extension_flags;
} CSA_extension;
```

DESCRIPTION

A data value of this type is an extension. The same extension structure is used to specify and receive extension information related to XCS function calls and XCS data structures.

In general, function calls and data structures may allow input and output extensions, with the direction implied by the extension item code. Input extensions may refer to storage allocated by the application and output extensions may refer to storage allocated by the XCS service.

For XCS extension arrays that may contain output extension storage allocated by the XCS service, callers must use *csa_free*( ) to free the pointer returned in the item_reference field. These structures are identified by the output flag CSA_EXT_OUTPUT set and a non-NULL item_reference value. Callers explicitly request output function extensions from function calls by setting the appropriate extension *item_code*. All substructures contained in the allocated memory will be freed when the base structure pointer is freed.

Data extensions do not need to be freed explicitly since they are freed with the structure they are contained in.

An extension has the following components:

item_code

A code that uniquely identifies this extension. The lower 16 bits are reserved for definition by the XCS specification. Any unused bits of these must be clear. The upper 16 bits of flags are reserved for definition by the extension.

item_data

Depending on the item_code, item_data may hold the length of the item value, the item value itself or other information about the item. The specification of the extension describes how this field should be interpreted.

item_reference

Depending on the item_code, item_reference may hold a pointer to where the item value is stored or NULL if there is no related item storage. The specification of the extension describes how this field should be interpreted.

extension flags

Bits for boolean attributes. The lower16 bits are reserved for definition by the XCS specification. Any unused bits of these must be clear. The upper 16 bits of flags are reserved for definition by the extension.

- CSA_EXT_REQUIRED

  Set: Return an error if this extension cannot be supported.

  Clear: Allow "best effort" support, including no support, of this extension.

- CSA_EXT_OUTPUT

  Set: Indicates on output extensions that this extension contains a pointer to memory allocated by the XCS implementation which must be freed with *csa_free*( ).

  Clear: The implementation did not allocate memory for the extension that the application needs to free. This flag is always clear on data extensions as described above.

- CSA_EXT_LAST_ELEMENT

  Set: Identifies the last structure in an array of such structures. This must be at the end of the extension array.

  Clear: This is not the last array element.

NAME

Flags - type definition for an XCS flag.

SYNOPSIS

```
typedef    CSA_uint32    CSA_flags;
```

DESCRIPTION

A data value of this type contains 32 flag bits. The meaning of the bits depends on the context in which the flags data value is used. Undocumented flags are reserved. Flags set to zero are referred to as "clear." Flags set non-zero are referred to as "set." Unspecified flags should always be clear.

NAME

Free Time - type definition for a XCS free time structure.

SYNOPSIS

```
typedef struct CSA_TAG_FREE_TIME {
    CSA_uint32               number_free_time_data;
    CSA_date_time_range     *free_time_data;
} CSA_free_time;
```

DESCRIPTION

A data value of this type is a free time structure. It has the following components:

number_free_time_data

Specifies the number of free time data elements returned.

free_time_data

A pointer to an array of date and time ranges. This array consists of free time intervals defined in terms of start and stop times.

NAME

Opaque Data - type definition for a opaque data value.

SYNOPSIS

```
typedef     struct CSA_TAG_OPAQUE_DATA {
    CSA_uint32        size;
    CSA_uint8        *data;
} CSA_opaque_data;
```

DESCRIPTION

A data value of this data type is an opaque data value. Opaque data structure consists of the following components.

size

Specifies the number of 8-bit, bytes of opaque data pointed to by data.

data

A pointer to an array of 8 bit values. There is no explicit semantics to this data.

NAME

Reminder - type definition for a reminder data value.

SYNOPSIS

```
typedef     struct CSA_TAG_REMINDER {
    CSA_time_duration        lead_time;
    CSA_time_duration        snooze_time;
    CSA_uint32               repeat_count;
    CSA_opaque_data         *reminder_data;
} CSA_reminder;
```

DESCRIPTION

A data value of this data type is a reminder. This data type is used to define a reminder attribute in the *csa_add_entry*( ), *csa_update_entry_attributes*( ) or *csa_read_entry_attributes*( ) functions.

lead_time

The lead time of the reminder. This date and time duration is interpreted as the duration before the start time of the entry, as defined by the Start Time attribute. A minus sign (i.e., ‘‘-’’) as the first character of the lead_time string represents a duration after the start of the entry.

snooze_time

The interval of time that the reminder is to be dormant if the reminder is to be repeated (i.e., *repeat_count* > 1).

repeat_count

The number of times that the reminder is to be repeated. Repeat counts greater than 1 need not be supported by implementations conforming to this specification.

reminder_data

Reminder specific information.

NAME

Reminder Reference - type definition for a CSA reminder reference structure.

SYNOPSIS

```
typedef    struct CSA_TAG_REMINDER_REFERENCE {
    CSA_entry_handle            entry;
    CSA_date_time               run_time;
    CSA_time_duration           snooze_time;
    CSA_uint32                  repeat_count;
    CSA_attribute_reference     attribute_name;
} CSA_reminder_reference;
```

DESCRIPTION

A data value of this type is reminder reference structure. This data type is used to represent a reminder for the *CSA_next_reminder*( ) function. The components of a reminder reference structure include:

entry
: The identifier of the calendar entry associated with this reminder.

run_time
: The date and time that this reminder is to be run.

snooze_time
: The interval after *run_time* that the reminder is to be dormant if the reminder is to be repeated (i.e., *repeat_count* > 1).

repeat_count
: The number of times that the reminder is to be repeated.

attribute_name
: The name of the reminder.

NAME

Return Code - type definition for a value returned from all XCS functions.

SYNOPSIS

```
typedef    CSA_uint32    CSA_return_code;
```

DESCRIPTION

A return code is defined as a 32 bit value. A non-zero value indicates an error with the error code being indicated by the value returned. A return value of zero indicates success. Values contained within the low order 16 bits are reserved for error codes defined in this specification. Values contained within the high order 16 bits are reserved for implementation defined error codes while the low order 16 bits should be set to an appropriate XCS error. Implementations specific return codes must contain a valid XCS return code portion in the lower 16 bits.

NAME

Service Reference - type definition for a calendar service reference data value.

SYNOPSIS

```
typedef     CSA_string        CSA_service_reference;
```

DESCRIPTION

A data value of this data type is a calendar service reference, i.e. the formal name for identifying a calendar service. This structure is used in the *CSA_list_calendars*( ) and the *CSA_logon*( ) functions.

NAME

Session Handle - type definition for a XCS Session Handle.

SYNOPSIS

```
typedef     system-defined, for example, uint32    CSA_session_handle;
```

DESCRIPTION

Opaque Session Handle. The session handle represents a logical connection to one calendar in the calendar service. The context identified by the Session Handle includes -session specific information. The CSA_session_handle is created by the XCS Logon function and destroyed by the XCS Logoff function.

NAME

String - type definition for a XCS character string.

SYNOPSIS

```
typedef    char *    CSA_string;
```

DESCRIPTION

A data value of this type is a string. The char array pointed to is interpreted as a null-terminated array of char by default. All implementations must support null terminated strings. The width of a character and the corresponding null terminating character are determined by the character set chosen.

To determine the character set of characters in the string, the XCS implementation looks at the session context. If there is no session context created before the call, the string will be interpreted using the implementations default character set. The implementation should always attempt to map all strings passed to the client application to the character set for the session.

NAME

Time Duration - type definition for a time duration value.

SYNOPSIS

```
typedef     CSA_string        CSA_time_duration;
```

DESCRIPTION

A data value of this type is a time duration value. The data type defines an interval of time. The value is defined in accordance with the basic format of ISO 8601, clause 5.5.3.2 for representing duration of time. For example,a duration of 10 hours, 30 minutes, and 20 seconds would be represented by the string:

```
PT10H30M20S
```

While ISO 8601 supports the representation of durations of time in terms of years, months, and days in addition to time, this specification restricts the duration to be the elements defining time.

3.3 Calendar Attributes

This section describes the calendar attributes defined by the XCS interface. The list of calendar attributes is extensible through the extended naming convention defined in Section 1.5 on page 2.

Attribute Name	Description
Access List	Access list attribute
Calendar Name	Calendar name attribute
Calendar Owner	Calendar owner attribute
Calendar Size	Calendar size attribute (computed)
Character Set	Character set attribute
Country	Country name attribute
Date Created	Calendar creation date attribute
Language	Language attribute
Number Entries	Number of entries attribute (computed)
Product Identifier	Product identifier or name attribute
Time Zone	Time zone attribute
Version	Product version attribute
Work Schedule	Preferred Work Schedule

Table 3-2 XCS Calendar Attributes

NAME

Access List - definition of the access list attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_ACCESS_LIST                \
    "-//XAPIA/CSA/CALATTR//NONSGML Access List//EN"
```

DESCRIPTION

The calendar access list. This defines the calendar users that have access to the calendar and the type of access that they have. The attribute default is an empty access list, implying that only the owner has access. The attribute is not deletable. It can be modified by a call to *csa_update_calendar_attributes*( ) if the user has CSA_OWNER_RIGHTS rights or CSA_CHANGE_CALENDAR_ATTRIBUTES rights.

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_ACCESS_LIST type of attribute. It contains a list of access rights structures as defined by **CSA_access_rights**.

NAME

Calendar Name - definition of the calendar name attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_CALENDAR_NAME                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Calendar Name//EN"
```

DESCRIPTION

The name of the calendar. This attribute is not defaultable. The name must be specified when the calendar is created. It is implementation specific whether this attribute can be modified or is read-only. If it can be modified, the user does so by a call to *csa_update_calendar_attributes*( ) if the user has CSA_OWNER_RIGHTS rights or CSA_CHANGE_CALENDAR_ATTRIBUTES rights.

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Calendar Owner - definition of the calendar owner attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_CALENDAR_OWNER                \
    "-//XAPIA/CSA/CALATTR//NONSGML Calendar Owner//EN"
```

DESCRIPTION

The owner of the calendar. This attribute is not defaultable. The name of the owner must be specified when the calendar is created. It is implementation specific whether this attribute can be modified or is read-only. If it can be modified, the user does so by a call to *csa_update_calendar_attributes*( ) if the user has CSA_OWNER_RIGHTS rights or CSA_CHANGE_CALENDAR_ATTRIBUTES rights.

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_CALENDAR_USER type of attribute.

NAME

Calendar Size - definition of the calendar size attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_CALENDAR_SIZE                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Calendar Size//EN"
```

DESCRIPTION

The size of the calendar in bytes. This is a computed attribute. It is computed when the attribute is read. This is a read-only attribute. It can not be modified by a call to

csa_update_calendar_attributes

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

NAME

Character Set - definition for a character set attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_CHARACTER_SET                \
    "-//XAPIA/CSA/CALATTR//NONSGML Character Set//EN"
```

DESCRIPTION

The character set for the calendar. This attribute default is implemenetation specific. The attribute values are a string representing the formal public identifier for the character set. The formal public identifer can be one of the following:

```
#define CSA_CHARSET_437        "-//XAPIA//CHARSET IBM 437//EN"
#define CSA_CHARSET_850        "-//XAPIA//CHARSET IBM 850//EN"
#define CSA_CHARSET_1252       "-//XAPIA//CHARSET Microsoft 1252//EN"
#define CSA_CHARSET_ISTRING    "-//XAPIA//CHARSET Apple ISTRING//EN"
#define CSA_CHARSET_UNICODE    "-//XAPIA//CHARSET UNICODE//EN"
#define CSA_CHARSET_T61        "-//XAPIA//CHARSET TSS T61//EN"
#define CSA_CHARSET_IA5        "-//XAPIA//CHARSET TSS IA5//EN"
#define CSA_CHARSET_ISO_10646  "-//XAPIA//CHARSET ISO 10646//EN"
#define CSA_CHARSET_ISO_646    "-//XAPIA//CHARSET ISO 646//EN"
#define CSA_CHARSET_ISO_8859_1 "-//XAPIA//CHARSET ISO 8859-1//EN"
```

Implementations may provide support for other, additional character sets. Support for this attribute is optional for implementations conforming to this specification. This attribute is defaultable to an implementation specific value. Once created, the attribute is read-only. It can not be modified by a call to *csa_update_calendar_attributes*( ).

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Country - definition for the country name attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_COUNTRY                          \
    "-//XAPIA/CSA/CALATTR//NONSGML Country//EN"
```

DESCRIPTION

The country name or nation. This attribute is not defaultable. The country name must be specified when the calendar is created. Once created, the attribute is read-only. It can not be modified by a call to *csa_update_calendar_attributes*( ). The valid values are the two character country code designations of ISO 3166. National language (i.e., US English) can be specified by the attribute tuple of CSA_CAL_ATTR_COUNTRY and CSA_CAL_ATTR_LANGUAGE. Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Date Created - definition of the date created attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_DATE_CREATED                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Date Created//EN"
```

DESCRIPTION

The date and time the calendar was created. The date and time value is the UTC time expressed in the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string 19850412T101530Z. This is a read-only attribute. The date is set by the calendar service. It can not be modified by a call to *csa_update_calendar_attributes*( ). The creation date is the date that the calendar was added to the calendar service.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

Language - definition of the language attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_LANGUAGE                          \
    "-//XAPIA/CSA/CALATTR//NONSGML Language//EN"
```

DESCRIPTION

The language that the calendar is encoded in. o This attribute is not defaultable. The language of the calendar must be specified when the calendar is created. Once created, the attribute is read-only. It can not be modified by a call to *csa_update_calendar_attributes*( ). The valid values are the two character language code designations of ISO 639. National language (i.e., US English) can be specified by the attribute tuple of CSA_CAL_ATTR_COUNTRY and CSA_CAL_ATTR_LANGUAGE.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Number Entries - definition of the number of entries attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_NUMBER_ENTRIES                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Number Entries//EN"
```

DESCRIPTION

The number of entries in the calendar. This is a computed attribute. It is computed when the attribute is read. This is a read-only attribute. It can not be modified by a call to *csa_update_calendar_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

NAME

Product Identifier - definition of the product identifier attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_PRODUCT_IDENTIFIER                \
    "-//XAPIA/CSA/CALATTR//NONSGML Product Identifier//EN"
```

DESCRIPTION

The calendar service product's unique identifier. The value is set by the calendar service. The vendor of the implementation must assure that this is a unique ISO 9070 formal public identifier value. This is a read-only attribute. It can not be modified by a call to *csa_update_calendar_attributes*( ).

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Time Zone - definition of the time zone attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_TIME_ZONE                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Time Zone//EN"
```

DESCRIPTION

The local time zone of the calendar. If not set at the time the calendar is created, this attribute defaults to the time zone of the service. The value is the difference between local time and Coordinate Universal Time. The format for the value is the basic format as specified in ISO 8601, clause 5.3.3.1.The value is positive if the local time is ahead of UTC and is negative if the local time is behind UTC. For example, the time zone in New York is normally 5 hours behind UTC, is represented by the string -0500. The time zone in Geneva is normally 1 hour ahead of UTC and is represented by the string +0100. This format supports the odd time zones such as those based on the half hour. This is a read-only attribute. It can not be modified by a call to *csa_update_calendar_attributes*( ).

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Version - definition of the version attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_VERSION                          \
    "-//XAPIA/CSA/CALATTR//NONSGML Version//EN"
```

DESCRIPTION

The version of the XCS Specification supported by the calendar service. This is a read-only attribute. It can not be modified by a call to *csa_update_calendar_attributes.*( ) The formal public identifier for the XCS Specification version associated with this specification must be the string:

```
"-//XAPIA/CSA/VERSION1/NONSGML CSA Version 1//EN"
```

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Work Schedule - definition of the work schedule attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_WORK_SCHEDULE                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Work Schedule//EN"
```

DESCRIPTION

The preferred work schedule of the calendar owner. This attribute is not defaultable. It can be modified by a call to *csa_update_calendar_attributes*( ). When present, it identifies the preferred time ranges that should be searched when searching for free time on this calendar. The schedule may be defined as some number of sets of non-cyclic work times or as a cycle of sets of work times.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_OPAQUE_DATA type of attribute. The data field of the opaque data that defines the work schedule attribute value is a pointer to the following structure:

```
typedef struct CSA_TAG_WORK_SCHEDULE{
    CSA_date_time          schedule_begin_time;
    CSA_boolean            cyclic_definition_flag;
    CSA_date_time          cycle_end_time;
    CSA_date_time_list    *work_cycle
} CSA_work_schedule;
```

The components of the work schedule structure are described as follows.

schedule_begin_time

The date and time that defines the beginning of the definition of the work schedule cycle.

cyclic_definition_flag

A boolean flag indicating whether the specified work schedule is non-cyclic (CSA_FALSE) or cyclic (CSA_TRUE) in nature.

cycle_end_time

The time that defines the end of the cyclic sets of work times. If *cycle_definition_flag* is **CSA_FALSE**, then this is NULL.

work_cycle

A list of work start and work end date and times. The specification of the date and time is in UTC as defined by the **CSA_date_time** data value.

3.4 Entry Attributes

This section describes the entry attributes defined by the XCS interface. The list of entry attributes is extensible through the extended naming convention defined in Section 1.5 on page 2.

Attribute Name	Description
Attendee List	Attendee list attribute
Audio Reminder	Audio reminder attribute
Classification	Classification attribute
Date Completed	Date completed attribute
Date Created	Entry creation date attribute
Description	Description attribute
Due Date	Due date attribute
End Date	End date attribute
Exception Dates	Exception dates attribute
Exception Rule	Exception rule attribute
Flashing Reminder	Flashing reminder attribute
Last Update	Last update attribute
Mail Reminder	Mail reminder attribute
Number Recurrences	Number recurrences attribute
Organizer	Organizer attribute
Popup Reminder	Popup reminder attribute
Priority	Priority attribute
Recurrence Rule	Recurrence rule attribute
Recurring Dates	Recurring dates attribute
Reference Identifier	Reference identifier attribute
Sequence Number	Sequence number attribute
Sponsor	Sponsor attribute
Start Date	Start date attribute
Status Status	attribute
Subtype	Subtype attribute
Summary	Summary attribute
Transparency	Time transparency or blocking attribute
Type	Type attribute

Table 3-3 XCS Entry Attributes

NAME

Attendee List - definition of the attendee list attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_ATTENDEE_LIST                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Attendee List//EN"
```

DESCRIPTION

The attendee list for an event, to-do, or memo entry type. This attribute is not defaultable. The attendee list must be specified when the entry is of type CSA_EVENT_MEETING. The attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_ATTENDEE_LIST type of attribute. It contains a list of attendee structures as defined by **CSA_attendee**.

NAME

Audio Reminder - definition of the audio reminder attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_AUDIO_REMINDER                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Audio Reminder//EN"
```

DESCRIPTION

An audio reminder.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_REMINDER type of attribute.

NAME

Classification - definition of the classification attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_CLASSIFICATION                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Classification//EN"
```

DESCRIPTION

The access classification for the calendar entry. The default for this attribute is CSA_CLASS_PUBLIC. The valid values include:

CSA_CLASS_PUBLIC
CSA_CLASS_PRIVATE
CSA_CLASS_CONFIDENTIAL

The attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

NAME

Date Completed - definition of the date completed attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_DATE_COMPLETED                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Date Completed//EN"
```

DESCRIPTION

The date and time the to-do was completed. The date and time value is the UTC time expressed in the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

Date Created - definition of the date created attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_DATE_CREATED                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Date Created//EN"
```

DESCRIPTION

The date and time the entry was created. The date and time value is the UTC time expressed in the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This is a read-only attribute. It can not be modified by a call to *csa_update_entry_attributes*( ). The creation date is the date that the entry was added to the calendar. The attribute value is set by the calendar service.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

Description - definition of the description attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_DESCRIPTION                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Description//EN"
```

DESCRIPTION

The entry description. The default value is a NULL string. This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Due Date - definition of the due date attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_DUE_DATE                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Due Date//EN"
```

DESCRIPTION

The date and time the to-do is due to be completed. The UTC (Coordinate Universal Time) based date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

End Date - definition of the end date attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_END_DATE                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML End Date//EN"
```

DESCRIPTION

The date and time the event will end. The UTC (Coordinate Universal Time) based date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute can be modified by a call to *csa_update_entry_attributes*( ). Events can have a start date but no end date. In this case, the event does not take up any time.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

Exception Dates - definition of the exception dates attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_EXCEPTION_DATES                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Exception Dates//EN"
```

DESCRIPTION

The list of exception dates to a recurring entry. The values for the attribute are a list of date and time values. It can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME_LIST type of attribute. The contents of this attribute is a list of string values representing UTC date and times in the basic, complete representation of ISO 8601.

NAME

Exception Rule - definition of the exception rule attribute.

SYNOPSIS

```
#define    CSA_CAL_ATTR_EXCEPTION_RULE                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Exception Rule//EN"
```

DESCRIPTION

The rule or repeating pattern for the exceptions to a recurring entry. The value for the attribute is a pattern specification for the exceptions. It can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute. The contents of this attribute is a formatted string that specifies the repeating pattern for the exception rule.

Basic Exception Rule Grammar

An exception rule is a string or clear-text encoding of a exception specification. An exception rule is composed of several components. A rule begins with a **frequency** which describes the type of repeating event (for example, daily, weekly, etc.). This is followed by an **interval** which indicates how often the frequency repeats (daily, every third day, etc). This can be followed by optional **frequency modifier** information and either an **end date** or a **duration**.

Below is the form of a typical rule. This example causes events to be generated every other week on Tuesday and Thursday:

```
W2 TU TH #4
```

where, W is the Frequency, 2 is the Duration, TU and TH are the optional Frequency Modifiers, and #4 is the Interval.

The basic exception rule grammar supports six types of repetition. The six types follow the same form with only the frequency name and optional modifier information changing from one type of frequency to the next.

Daily Rule

The daily rule is used for specifying repeating events based on an interval of a day or more. These can range from every day to every 200th day and beyond. The daily rule begins with the letter D followed by an interval (representing days) and an optional duration or end date.

Some examples follow:

Daily for 10 events:	`D1 #10`
Daily until 12/24/94:	`D1 19941224T000000Z`
Every other day - forever:	`D2 #0`
Every 10 days, 5 events:	`D10 #5`

Weekly Rule

The weekly rule is used for specifying repeating events based on an interval of a week or more. The basic weekly rule has the same form as the daily rule except that the rule begins with a **W** and can contain an optional list of weekdays the events are generated on. For weekly rules, the interval represents weeks.

Some examples follow:

Weekly for 10 events:	`W1 #10`
Weekly until 12/24/94:	`W1 19941224T000000Z`
Every other week - forever:	`W2 #0`
Weekly on Tuesday and Thursday for 5 weeks:	`W1 TU TH #5`
Every other week on Monday Wednesday and Friday until 12/24/94:	`W2 MO WE FR 19941224T000000Z`

Monthly Rule

The monthly rule is used for specifying repeating events base on an interval of a month or more. There are two types of monthly exception rules: one for **by-position** and one for **by-day**. The by-position rule allows weekdays in the month to be specified in relation to their occurrence in the month. An example would be to specify the third Sunday of the month or the last Friday of the month. A by-day rule allows actual day numbers to be specified such as the 12th day or 29th day. An occurrence specifier may be used in monthly by-position rules. The occurrence specifiers control which occurrence of a weekday in a month an event occurs on:

1+, 2+, ... 5+ for the first occurrence, second, ...fifth occurrence of the month.

1-, 2-, ... 5- for the last occurrence, second to last occurrence, etc.

A `2+ FR SA` would indicate the second occurrence of Friday and Saturday in the month. A `1- MO` would indicate the first occurrence of Monday working from the end of the month backwards (for example, the last occurrence). A `2- MO` would be the second to the last Monday of the month. The by-position rule begins with a `MP` and the by-day rule begins with a `MD`. The interval in monthly rules represents months. Some examples follow:

Monthly on the 1st Friday for ten events:	`MP1 1+ FR #10`
Monthly on the 1st Friday until 12/24/94:	`MP1 1+ FR 19941224T000000Z`
Every other month on the 1st and last Sunday of the month:	`MP2 1+ SU 1- SU #10`
Every six months on the 2nd Monday through Friday:	`MP6 2+ MO TU WE TH FR #10`
Monthly on the 2nd and 15th of the month:	`MD1 2 15 #10`
Monthly on the 1st and last day of the month:	`MD1 1 LD #10`
Every 18 months on the 10th through 15th of the month:	`MD18 10 11 12 13 14 15 #10`

Yearly Rule

The yearly rule is used for specifying repeating events base on an interval of a year or more. There are two types of yearly exception rules. One for **by-month** and one for **by-day**. The by-month rule allows specific months out of the year to be specified. The by-day allows specific days to be specified. In the by-month rule, the day in the month the rule is to occur on is determined from the initial appointment.

The by-month rule begins with a `YM` and the by-day rule begins with a `YD`. The interval in yearly rules represents year.

Some examples follow:

Yearly in June and July:	`YM1 6 7 #10`
Every other year on January, Feb, and March:	`YM2 1 2 3 #10`
Every 3rd year on the 1st, 100th and 200th day:	`YD3 1 100 200 #10`

Grammar

The following grammar defines the extended recurrence rule syntax.

{}	0 or more
[]	0 or 1
start	<minuteop> [<enddate>] | <daily> [<enddate>] | <weekly> [<enddate>] | <monthlybypos> [<enddate>] | <monthlybyday> [<enddate>] | <yearlybymonth> [<enddate>] | <yearlybyday> [<enddate>]
endmarker	$
enddate	ISO 8601 (clause 5.4.1) string(e.g. 19940712T101530Z)
interval	<digits>
duration<digits>	
lastday	LD
plus	+
minus	-
daynumber	<1-31>[<endmarker>] | <lastday>
daynumberlist	daynumber {<daynumber>}
month	<1-12>[<endmarker>]
monthlist	<month> {<monthlist>}
day	<1-366>[<endmarker>]
daylist	<day> {<daylist>}
occurrence	<1-5><plus>[<endmarker>] | <1-5><minus>[<endmarker>]
occurrencelist	<occurrence> {<occurrencelist>}
weekday	<SU | MO | TU | WE | TH | FR | SA>[<endmarker>]
weekdaylist	<weekday> {<weekdaylist>}
time	<hhmm>[<endmarker>]
timelist	<time> {<timelist>}

daytime	<weekday> {<timelist>} {<daytime>}
weekdaytime	<occurrencelist> <weekdaylist> {<timelist>} {<weekdaytime>}
minuteop	M<interval> [<duration>]
daily	D<interval> [<timelist>] [<duration>] [<minuteop>]
weekly	W<interval> <daytime> [<duration>] [<minuteop>]
monthlybypos	MP<interval> [<weekdaytime>] [<duration>][<weekly> | <daily> | <minuteop>]
monthlybyday	MD<interval> [<daynumberlist>] [<duration>] [<weekly> | <daily> | <minuteop>]
yearlybymonth	YM<interval> [<monthlist>] [<duration>] [<monthlybyday> | <monthlybypos> | <weekly> | <daily> | <minuteop>]
yearlybyday	YD<interval> [<daylist>] [<duration>] [<monthlybyday> | <monthlybypos> | <weekly> | <daily> | <minuteop>]

Glossary

enddate Controls when a repeating event terminates. The enddate is the last time an event can occur.

interval Defines the frequency in which a rule repeats.

duration Controls the number of events a rule generates.

lastday Can be used as a replacement to daynumber to indicate the last day of the month.

daynumber A number representing a day of the month.

month A number representing a month of the year.

day A number representing a day of the year.

occurrence Controls which week of the month a particular weekday event occurs.

weekday A symbol representing a day of the week.

minuteop Defines a rule that repeats on a particular minute interval.

daily Defines a rule that repeats on a daily basis.

weekly Defines a rule that repeats on a weekly basis.

monthlybypos Defines a rule that repeats on a monthly basis on a relative day and week.

monthlybyday Defines a rule that repeats on a monthly basis on an absolute day.

yearlybymonth Defines a rule that repeats on specific months of the year.

yearlybyday Defines a rule that repeats on specific days of the year.

Policies

1. The duration portion of a rule defines the total number of events the rule generates, including the first event.
2. Information, not contained in the rule, necessary to determine the next event time and date is derived from the Start Time entry attribute.
3. If an end date and a duration is specified in the rule, the recurring event ceases when the end date is reached or the number of events indicated in the duration occur; whichever comes first.

4. If the duration or and end date is not established in the rule (for example, D4) the event occurs twice. That is, D4 is equivalent to D4 #2.
5. If a rule has an ambiguity with respect to whether it repeats on a specific day (12th of the month) versus on a relative day (2nd Friday of the month), the specific day takes precedence. The only exception to this policy is the <*monthlybypos*> rule.
6. A duration of #0 means repeat this event forever.
7. Using the occurrence specifier 5+ (for example 5th Friday) or 5- (for example 5th from last Friday) in a month that does not contain 5 weeks does not generate an event and thus does not count against the duration. The same applies to providing a day of the month that does not occur in the month: 31st.
8. The start time and date of an entry must be synchronised with one of the repeating events defined by its exception rule. The following is not allowed:

   Initial Appt Date: `7/1/94 (Friday)`
   Exception Rule: `W1 MO TH #5`

   The following is acceptable:

   Initial Appt Date: `7/1/94 (Friday)`
   Exception Rule: `W1 MO FR #5 or W1 #5`

9. If the optional <*occurrencelist*> and <*weekdaylist*> information is missing from a <*monthlybypos*> frequency the information is derived from the entry attributes. The <*occurrence*> used in the recurring event is a count from the beginning of the month to the entry date and the <*weekday*> used is the day of the week the entry is scheduled to occur on.

   If the <*monthlybypos*> frequency does not list a week day (for example, SU) in the rule, the week day is established from the entry attribute information. As an example the rule `MP1 #3` used in an entry with a start date of 7/20/94 (which is the third Wednesday of the month) repeats on 8/17/94 which is the third Wednesday of the month.

NAME

Flashing Reminder - definition of the flashing reminder attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_FLASHING_REMINDER                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Flashing Reminder//EN"
```

DESCRIPTION

A flashing reminder. This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_REMINDER type of attribute.

NAME

Last Update - definition of the last update attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_LAST_UPDATE                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Last Update//EN"
```

DESCRIPTION

The UTC (Coordinated Universal Time) based date and time the entry was last updated. This is a read-only attribute. It can not be modified by a call to *csa_update_entry_attributes*( ). The value is set by the calendar server when the entry is added to the calendar and each time that it is updated. The date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

Mail Reminder - definition of the mail reminder attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_MAIL_REMINDER                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Mail Reminder//EN"
```

DESCRIPTION

An mail reminder. This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_REMINDER type of attribute.

NAME

Number Recurrences - definition of the number recurrences attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_NUMBER_RECURRENCES            \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Number Recurrences//EN"
```

DESCRIPTION

The number of times the entry repeats. This is a computed attribute. It is computed when the attribute is read. The value is equal to the number of recurrences that are specified by the union of the Recurrence Dates, Recurrence Rule, Exception Dates, and Exception Rule attribute values. This is a read-only attribute. It can not be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

NAME

Organizer - definition of the organizer attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_ORGANIZER                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Organizer//EN"
```

DESCRIPTION

The organiser of the calendar entry. The value of this attribute is set by the calendar service and is set to the calendar user who added the entry. At the time the calendar entry is created, it can also be set to a different calendar user. It is implementation specific whether this attribute can be modified or is read-only. If it can be modified, it is done so by a call to *csa_update_entry_attributes*( ).

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_CALENDAR_USER type of attribute.

NAME

Popup Reminder - definition of the pop-up reminder attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_POPUP_REMINDER                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Popup Reminder//EN"
```

DESCRIPTION

A pop-up reminder. This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_REMINDER type of attribute.

NAME

Priority - definition of the priority attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_PRIORITY                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Priority//EN"
```

DESCRIPTION

The priority of the entry. This attribute can be modified by a call to *csa_update_entry_attributes*( ). A value of zero specifies an undefined priority. A value of one is the highest priority. A value of two is the second highest priority. Subsequent numbers specify a decreasing ordinal priority.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

NAME

Recurring Dates - definition of the recurring dates attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_RECURRING_DATES                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Recurring Dates//EN"
```

DESCRIPTION

The list of recurring dates for the entry. The values for the attribute are a list of date and time values. It can be modified by a call to *csa_update_entry_attributes*( ). This attribute is only one of the attributes that CSA_ENTRY_ATTR_NUMBER_RECURRENCES is computed from.

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME_LIST type of attribute. The contents of this attribute is a list of string values representing UTC date and times in the basic, complete representation of ISO 8601.

NAME

Recurrence Rule - definition of the recurrence rule attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_RECURRENCE_RULE                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Recurrence Rule//EN"
```

DESCRIPTION

The rule or repeating pattern for a recurring entry. The value for the attribute is a pattern specification for the recurrence. It can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute. The contents of this attribute is a formatted string that specifies the repeating pattern for the recurrence rule.

SEE ALSO

For specification of Basic Recurrence Rule Grammar, and definitions for Daily Rule, Weekly Rule, Monthly Rule, Yearly Rule, the Grammar used, a Glossary of the associated terminology, and Policies, see the man-page definition for **Exception Rule**.

NAME

Reference Identifier - definition of the reference identifier attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_REFERENCE_IDENTIFIER           \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Reference Identifier//EN"
```

DESCRIPTION

A persistent, unique identifier that can be used to reference the entry within a calendar service. This attribute is persistent. The attribute is set by the calendar service when the entry is added to the calendar. This is a read-only attribute. It can not be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_OPAQUE_DATA type of attribute. The format for the data in this attribute is implementation specific.

NAME

Sequence Number - definition of the sequence number attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_SEQUENCE_NUMBER                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Sequence Number//EN"
```

DESCRIPTION

The sequence number of the entry. The value is set to zero by the calendaring service when the entry is added to the calendar. It is incremented when the entry is updated. This is a read-only attribute. It can not be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

NAME

Sponsor - definition of the sponsor attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_SPONSOR                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Sponsor//EN"
```

DESCRIPTION

The sponsor of the calendar entry. This attribute defaults to the owner of the calendar. The value of this attribute can also be set by the creator of the calendar entry to be a value of a different calendar user. The sponsor of the calendar entry is the calendar user that is the effective owner for the event or to-do. It can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_CALENDAR_USER type of attribute.

NAME

Start Date - definition of the start date attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_START_DATE                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Start Date//EN"
```

DESCRIPTION

The UTC (Coordinate Universal Time) based date and time the event will start. The date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute must be specified when the entry is created. Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_DATE_TIME type of attribute.

NAME

Status - definition of the status attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_STATUS                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Status//EN"
```

DESCRIPTION

The status associated with the calendar entry. This attribute is a CSA_VALUE_UINT32 type of attribute. The valid values for this attributes include:

CSA_STATUS_ACCEPTED
CSA_STATUS_NEEDS_ACTION
CSA_STATUS_SENT
CSA_STATUS_TENTATIVE
CSA_STATUS_CONFIRMED
CSA_STATUS_REJECTED
CSA_STATUS_COMPLETED
CSA_STATUS_DELEGATED

The default value for this attribute is CSA_STATUS_NEEDS_ACTION. This attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

NAME

Subtype - definition of the subtype attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_SUBTYPE                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Subtype//EN"
```

DESCRIPTION

The subtype of the calendar entry. The subtype of the calendar entry must be specified, if the attribute is supported, when the calendar entry is created. It can be modified by a call to *csa_update_entry_attributes*( ).

Possible values for the type of calendar entry subtype might include the following:

CSA_SUBTYPE_APPOINTMENT
CSA_SUBTYPE_CLASS
CSA_SUBTYPE_HOLIDAY
CSA_SUBTYPE_MEETING
CSA_SUBTYPE_MISCELLANEOUS
CSA_SUBTYPE_PHONE_CALL
CSA_SUBTYPE_SICK_DAY
CSA_SUBTYPE_SPECIAL_OCCASION
CSA_SUBTYPE_TRAVEL
CSA_SUBTYPE_VACATION

Implementations may define their own set of values for this attribute. Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Summary - definition of the summary attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_SUMMARY                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Summary//EN"
```

DESCRIPTION

The summary or subject of the calendar entry. This attribute is defaultable to a null string. The attribute can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_STRING type of attribute.

NAME

Time Transparency - definition of the time transparency attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_TIME_TRANSPARENCY                \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Time Transparency//EN"
```

DESCRIPTION

The time transparency of the entry to free time searches. The default value for this attribute is zero, the entry is not transparent. It can be modified by a call to *csa_update_entry_attributes*( ).

Support for this attribute is optional for implementations conforming to this specification.

This attribute is a CSA_VALUE_SINT32 type of attribute. A value of zero specifies that the entry blocks time and will be factored into a free time search. A non-zero value has implementation specific meaning on free time search. Some implementations may treat all non-zero values as non-blocking or transparent events. Other implementations may use the numeric value to provide a layering of levels of transparency.

NAME

Type - definition of the entry type attribute.

SYNOPSIS

```
#define    CSA_ENTRY_ATTR_TYPE                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Type//EN"
```

DESCRIPTION

The type of the calendar entry. This attribute must be specified when the calendar entry is created. This attribute is not defaultable. The type must be specified when the calendar is created. This is a read-only attribute. It can not be modified by a call to *csa_update_entry_attributes*( ). The valid types for the calendar entry include:

CSA_TYPE_EVENT
CSA_TYPE_TODO
CSA_TYPE_MEMO

Each type of entry has meaningful set of attributes. Some attributes do not make sense for a given calendar entry type.

Support for this attribute is mandatory for implementations conforming to this specification.

This attribute is a CSA_VALUE_UINT32 type of attribute.

Chapter 4

Functional Interface

This chapter defines the functions of the Calendaring and Scheduling API interface. The functions of both the generic and C interfaces are specified. Those of the C interface are repeated in Chapter 5. Table 4-1 lists the functions of the XCS interface.

Function	Description
Simple Calendaring and Scheduling Interface	
Add Event	String-based function to add an event to a calendar.
Add Todo	String-based function to add a to-do to a calendar.
Add Memo	String-based function to add a memo to a calendar.
Administration	
Free	Free memory allocated by the calendar service.
List Calendars	List calendars supported by a calendar service.
Logoff	Terminate a session with a calendar.
Logon	Establish a session with a calendar.
Look Up	Look up a calendar addressing information.
Query Configuration	Query configuration of a service.
Restore	Restore calendar entries from an archive file.
Save	Save calendar entries to an archive file.
Calendar Management	
Add Calendar	Add a calendar to the calendar service.
Call Callbacks	Force the invocation of the specified callback functions.
Delete Calendar	Delete a calendar from the calendar service.
List Calendar Attributes	List the names of the calendar attributes.
Read Calendar Attributes	Read and return a specified set of calendar attributes.
Register Callback Functions	Register the callback functions for a calendar.
Unregister Callback Functions	Remove the specified callback functions for a calendar.
Update Calendar Attributes	Update a set of calendar attributes.
Entry Management	
Add Entry	Add an entry to a calendar.
Delete Entry	Delete one or more entries on a calendar.
Free Time Search	Search for free time on one or more calendars.
List Entries	List the entry handles for the entries in a calendar
List Entry Attributes	List the names of the entry attributes in a calendar entry.
List Entry Sequence	List the entry handles for a sequence of recurring entries.
Read Entry Attributes	Read the entry attributes associated with a calendar entry.
Read Next Reminders	Read the next set of reminders in a calendar.
Update Entry Attributes	Update a set of entry attributes.

Table 4-1 XCS Interface Functions

The manual pages for these functions are given in the remainder of this chapter.

4.1 Simple Calendaring and Scheduling Interface

The Simple Calendaring and Scheduling Interface functions provide a simple way for non-calendaring and non-scheduling applications to add an entry to a calendar. For example, a product management application adds an entry to a calendar after scheduling a task within a project.

Function parameters are string-based or integer-based to enable bindings with languages which do not support complex data types. For example, these languages might include BASIC and application- embedded macro languages.

If the organiser of a XCS entry has not yet logged into the calendaring service, the Simple Calendaring and Scheduling Interface function will attempt to prompt the user using an implementation specific user interface for logon parameters such as the calendar service, the calendar name, a password, and other session context information. If the implementation does not support a user interface, then the Simple Calendaring and Scheduling Interface functions in this Section 4.1 returns an error and the function does not complete successfully.

The Simple Calendaring and Scheduling Interface functions provide three simple string-based calendar and scheduling functions, one for each of the three types of calendar entries.

NAME

Add Event - a string-based function to add an event type of entry to the specified calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_add_event(
    CSA_service_reference    calendar_service,
    CSA_string               calendar_address,
    CSA_string               logon_user,
    CSA_string               logon_password,
    CSA_string               attendee,
    CSA_enum                 attendee_priority,
    CSA_enum                 attendee_status,
    CSA_boolean              attendee_rsvp_requested,
    CSA_date_time            start_date,
    CSA_date_time            end_date,
    CSA_string               organizer,
    CSA_string               sponsor,
    CSA_string               summary,
    CSA_string               description,
    CSA_string               recurrence_rule,
    CSA_string               exception_rule,
    CSA_string               subtype,
    CSA_enum                 classification,
    CSA_string               delimiters,
    CSA_string               add_event_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification, unless the platform type does not support the Simple Calendaring and Scheduling Interface functions. Platform specific information is included in another section of this specification.

This function adds an event type of entry to the specified calendar. For more than one attendee for an event, this function must be called for each attendee. This function is primarily intended for calling from a scripting language (for example, spreadsheet macro) that cannot handle data structures.

This function will try to establish a calendar session without a logon user interface. If this is not possible and a user interface has been requested with a function extension, it will prompt for logon information to establish the calendar session. The session is always closed on completion. The function will return CSA_E_UNSUPPORTED_FUNCTION_EXT if the user requests a user interface through the extension mechanism and it is not supported by the implementation. If the user interface was not requested and insufficient information was specified to logon to the calendar service, then the error CSA_E_INVALID_USER will be returned. The user interface portion of this function is not required for conformance to this specification.

ARGUMENTS

Calendar Service (Service Reference)

A pointer to a calendar service name specifying the requested calendaring service (for example, the path to a calendar store or a remote service node name). This value may be NULL if the underlying calendaring service does not require a service name or if the

function extension is used to specify a user interface. This may be necessary on some implementations and ignored on others.

Calendar Address (String)
A pointer to the address that is acceptable to the underlying calendaring service for a specific calendar. The format of the address string is not defined by this specification. It is intended to support any string notation supported by a given implementation.

Logon User (String)
A pointer to the friendly name or display name of the calendar user that is logging on to the calendaring service.

Logon Password (String)
A pointer to a string containing the password required for access to the calendaring service. This value may be NULL if the underlying service does not require a password or if the function extension is used to request a user interface.

Attendee (String)
A pointer to the address that is acceptable to the underlying calendaring service for the intended attendee for the calendar entry. The format of the address string is not defined by this specification. It is intended to support any string notation supported by a given implementation.

Attendee Priority (Enum)
An indication of the originators expectation of the attendees attendance. Valid values include:

CSA_FOR_YOUR_INFORMATION
CSA_ATTENDANCE_REQUESTED
CSA_ATTENDANCE_REQUIRED
CSA_IMMEDIATE_RESPONSE

Support for all of these values is optional for conformance to this specification.

Attendee Status (Enum)
The current status of the attendees participation. Valid values include:

CSA_STATUS_ACCEPTED
CSA_STATUS_NEEDS_ACTION
CSA_STATUS_SENT
CSA_STATUS_TENTATIVE
CSA_STATUS_CONFIRMED
CSA_STATUS_REJECTED
CSA_STATUS_COMPLETED
CSA_STATUS_DELEGATED

Support for all of these values is optional for conformance to this specification.

Attendee Rsvp Requested (Boolean)
Whether (CSA_TRUE) or not (CSA_FALSE) the favour of a reply is requested.

Start Date (Date Time)
The UTC (Coordinate Universal Time) based date and time the event will start. The date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1. For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute must be specified when the entry is added to the calendar.

End Date (Date Time)
The date and time the event will end. The UTC (Coordinate Universal Time) based date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1." For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

Events can have a start date but no end date. In this case, the event does not take up any time.

Organizer (String)
The organiser of the calendar entry. If NULL, the value of this attribute will be set by the calendar service to the calendar user who added the entry.

Sponsor (String)
The sponsor of the calendar entry. If NULL, this attribute will default to owner of the calendar to which the entry is added. The value of this attribute can also be set by the creator of the calendar entry to be a value of a different calendar user. The sponsor of the calendar entry is the calendar user that is the effective owner for the event.

Summary (String)
The summary or subject of the event. If NULL, then the attribute defaults to a null string.

Description (String)
The entry description. If NULL, then the attribute defaults to a null string.

Recurrence Rule (String)
The rule or repeating pattern for the repeating events associated with a recurring event. The value for the attribute is a pattern specification for the recurrences to the event. If NULL, then the attribute defaults to a null string to denote that a recurrence rule does not exist and the event has a single occurrence. For more information on specifying a recurrence rule see the section Recurrence Rule.

Exception Rule (Strinag)
The rule or repeating pattern for the exceptions to a recurring event. The value for the attribute is a pattern specification for the exceptions to the event. If NULL, then the attribute defaults to a null string to denote that an exception rule does not exist. For more information on specifying an exception rule see the section Exception Rule.

Subtype (String)
The subtype of the calendar entry. The subtype of the calendar entry must be specified, if the attribute is supported, when the calendar entry is created. Possible values for the type of calendar entry subtype might include the following:

CSA_SUBTYPE_APPOINTMENT
CSA_SUBTYPE_CLASS
CSA_SUBTYPE_HOLIDAY
CSA_SUBTYPE_MEETING
CSA_SUBTYPE_MISCELLANEOUS
CSA_SUBTYPE_PHONE_CALL
CSA_SUBTYPE_SICK_DAY
CSA_SUBTYPE_SPECIAL_OCCASION
CSA_SUBTYPE_TRAVEL
CSA_SUBTYPE_VACATION

Implementations may define their own set of values for this attribute. Support for this attribute is optional for implementations conforming to this specification.

Classification (Enum)
The access classification for the calendar entry. The default for this attribute is CSA_CLASS_PUBLIC. The valid values include:

CSA_CLASS_PUBLIC
CSA_CLASS_PRIVATE
CSA_CLASS_CONFIDENTIAL

Delimiters (String)
A pointer to a character that is used to delimit the argument components of the *add_event_exceptions* string. This character can not be NULL.

Add Event Extensions (String)
A pointer to a delimited string of argument-value pairs. The pairs are separated with the equal sign character (that is, ''='). The delimiter argument specifies the character used to delimit the argument-value pairs. The valid argument values include: *item_code*, *item_data*, *item_reference* and *extension_flags*. The item_code values are the symbolic names of the extension identifier. The error CSA_E_INVALID_FUNCTION_EXT is returned if the function is invalid. The *item_data* and *item_reference* are extension specific string values. The extension_flag values are hexadecimal characters representing the 32-bit boolean bitmask representation of the flags. More than one function extension can be specified in the delimited string. The error CSA_E_UNSUPPORTED_FUNCTION_EXT is returned if the implementation does not support the specified extension.

RESULTS

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INVALID_CALENDAR_SERVICE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_ENUM
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_PASSWORD
CSA_E_INVALID_RULE
CSA_E_INVALID_USER
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_PASSWORD_REQUIRED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TOO_MANY_USERS
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Add Todo - a string-based function to add a to-do type of entry to the specified calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_add_todo(
    CSA_service_reference    calendar_service,
    CSA_string               calendar_address,
    CSA_string               logon_user,
    CSA_string               logon_password,
    CSA_enum                 attendee_priority,
    CSA_enum                 attendee_status,
    CSA_boolean              attendee_rsvp_requested,
    CSA_date_time            start_date,
    CSA_date_time            due_date,
    CSA_uint32               priority,
    CSA_string               summary,
    CSA_string               description,
    CSA_enum                 classification,
    CSA_string               delimiters,
    CSA_string               add_todo_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification, unless the platform type does not support the Simple Calendaring and Scheduling Interface functions. Platform specific information is included in another section of this specification.

This function adds a to-do type of entry to the specified calendar. For a group to-do, this function must be called for each attendee of the to-do. This function is primarily intended for calling from a *scripting language* (for example, spreadsheet macro) that cannot handle data structures.

This function will try to establish a calendar session without a logon user interface. If this is not possible and a user interface has been requested with a function extension, it will prompt for logon information to establish the calendar session. The session is always closed on completion. The function will return CSA_E_UNSUPPORTED_FUNCTION_EXT if the user requests a user interface through the extension mechanism and it is not supported by the implementation. If the user interface was not requested and insufficient information was specified to logon to the calendar service, then the error CSA_E_INVALID_USER will be returned. The user interface portion of this function is not required for conformance to this specification.

ARGUMENTS

Calendar Service (Service Reference)

A pointer to a calendar service name specifying the requested calendaring service (for example, the path to a calendar store or a remote service node name). This value may be NULL if the underlying calendaring service does not require a service name or if the function extension is used to specify a user interface. This may be necessary on some implementations and ignored on others.

Calendar Address (String)
A pointer to the address that is acceptable to the underlying calendaring service for the calendar associated with the specified user. The format of the address string is not defined by this specification. It is intended to support any string notation supported by a given implementation.

Logon User (String)
A pointer to the friendly name or display name of the calendar user that is logging on to the calendaring service.

Logon Password (String)
A pointer to a string containing the password required for access to the calendaring service. This value may be NULL if the underlying service does not require a password or if the function extension is used to request a user interface.

Attendee Priority (Enum)
An indication of the originators expectation of the attendees attendance. Valid values include:

CSA_FOR_YOUR_INFORMATION
CSA_ATTENDANCE_REQUESTED
CSA_ATTENDANCE_REQUIRED
CSA_IMMEDIATE_RESPONSE

Support for all of these values is optional for conformance to this specification.

Attendee Status (Enum)
The current status of the attendees participation. Valid values include:

CSA_STATUS_ACCEPTED
CSA_STATUS_NEEDS_ACTION
CSA_STATUS_SENT
CSA_STATUS_TENTATIVE
CSA_STATUS_CONFIRMED
CSA_STATUS_REJECTED
CSA_STATUS_COMPLETED
CSA_STATUS_DELEGATED

Support for all of these values is optional for conformance to this specification.

Attendee Rsvp Requested (Boolean)
Whether (CSA_TRUE) or not (CSA_FALSE) the favour of a reply is requested.

Start Date (Date Time)
The UTC (Coordinate Universal Time) based date and time the to-do is to be assigned. In some implementations, this will be the date that the to-do will begin to show on a calendar. The date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1." For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute must be specified when the entry is added to the calendar.

Due Date (Date Time)
The date and time the to-do is due to be completed. The UTC (Coordinate Universal Time) based date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1." For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

Events can have a start date but no end date.

Priority (Uint32)
The priority of the to-do. A value of zero specifies an undefined priority. A value of one is the highest priority. A value of two is the second highest priority. Subsequent numbers specify a decreasing ordinal priority.

Summary (String)
The summary or subject of the event. If NULL, then the attribute defaults to a null string.

Description (String)
The entry description. If NULL, then the attribute defaults to a null string.

Classification (Enum)
The access classification for the calendar entry. The default for this attribute is CSA_CLASS_PUBLIC. The valid values include:

CSA_CLASS_PUBLIC
CSA_CLASS_PRIVATE
CSA_CLASS_CONFIDENTIAL

Delimiters (String)
A pointer to a character that is used to delimit the argument components of the add_todo_extensions string. This character can not be NULL.

Add Todo Extensions (String)
A pointer to a delimited string of argument-value pairs. The pairs are separated with the equal sign character (that is, ‘‘=’’). The delimiter argument specifies the character used to delimit the argument-value pairs. The valid argument values include: *item_code*, *item_data*, *item_reference* and *extension_flags*. The item_code values are the symbolic names of the extension identifier. The error CSA_E_INVALID_FUNCTION_EXT is returned if the function is invalid. The *item_data* and *item_reference* are extension specific string values. The extension_flag values are hexadecimal characters representing the 32-bit boolean bitmask representation of the flags. More than one function extension can be specified in the delimited string. The error CSA_E_UNSUPPORTED_FUNCTION_EXT is returned if the implementation does not support the specified extension.

RESULTS

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INVALID_CALENDAR_SERVICE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_ENUM
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_PASSWORD
CSA_E_INVALID_USER
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_PASSWORD_REQUIRED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TOO_MANY_USERS
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Add Memo -a string-based function to add a memo type of entry to the specified calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_add_memo(
    CSA_service_reference    calendar_service,
    CSA_string               calendar_address,
    CSA_string               logon_user,
    CSA_string               logon_password,
    CSA_date_time            start_date,
    CSA_string               summary,
    CSA_string               delimiters,
    CSA_string               add_memo_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification, unless the platform type does not support the Simple Calendaring and Scheduling Interface functions. Platform specific information is included in another section of this specification.

This function adds a memo type of entry to the specified calendar. This function is primarily intended for calling from a scripting language (for example, spreadsheet macro) that cannot handle data structures.

This function will try to establish a calendar session without a logon user interface. If this is not possible and a user interface has been requested with a function extension, it will prompt for logon information to establish the calendar session. The session is always closed on completion. The function will return CSA_E_UNSUPPORTED_FUNCTION_EXT if the user requests a user interface through the extension mechanism and it is not supported by the implementation. If the user interface was not requested and insufficient information was specified to logon to the calendar service, then the error CSA_E_INVALID_USER will be returned. The user interface portion of this function is not required for conformance to this specification.

ARGUMENTS

Calendar Service (Service Reference)

A pointer to a calendar service name specifying the requested calendaring service (for example, the path to a calendar store or a remote service node name). This value may be NULL if the underlying calendaring service does not require a service name or if the function extension is used to specify a user interface. This may be necessary on some implementations and ignored on others.

Calendar Address (String)

A pointer to the address that is acceptable to the underlying calendaring service for the calendar associated with the specified user. The format of the address string is not defined by this specification. It is intended to support any string notation supported by a given implementation.

Logon User (String)

A pointer to the friendly name or display name of the calendar user that is logging on to the calendaring service.

Logon Password (String)
A pointer to a string containing the password required for access to the calendaring service. This value may be NULL if the underlying service does not require a password or if the function extension is used to request a user interface.

Start Date (Date Time)
The UTC (Coordinate Universal Time) based date and time the event will start. The date and time value is the complete representation, basic format as specified in ISO 8601, clause 5.4.1." For example, UTC time of 10 hours, 15 minutes and 30 seconds on April 12, 1985 is specified the string:

```
19850412T101530Z
```

This attribute must be specified when the entry is added to the calendar. Summary (String) The memo text or summary. If NULL, then the attribute defaults to a null string.

Delimiters (String)
A pointer to a character that is used to delimit the argument components of the *add_event_exceptions* string. This character can not be NULL.

Add Memo Extensions (String)
A pointer to a delimited string of argument-value pairs. The pairs are separated with the equal sign character (that is, ''='). The delimiter argument specifies the character used to delimit the argument-value pairs. The valid argument values include: *item_code*, *item_data*, *item_reference* and *extension_flags*. The item_code values are the symbolic names of the extension identifier. The error CSA_E_INVALID_FUNCTION_EXT is returned if the function is invalid. The *item_data* and *item_reference* are extension specific string values. The *extension_flag* values are hexadecimal characters representing the 32-bit boolean bitmask representation of the flags. More than one function extension can be specified in the delimited string. The error CSA_E_UNSUPPORTED_FUNCTION_EXT is returned if the implementation does not support the specified extension.

RESULTS

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INVALID_CALENDAR_SERVICE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_PASSWORD
CSA_E_INVALID_USER
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_PASSWORD_REQUIRED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TOO_MANY_USERS
CSA_E_UNSUPPORTED_FUNCTION_EXT

4.2 Administration

The man-page definitions in this section provide the *administration* functionality of the Calendaring and Scheduling interface.

These comprise the following:

- *Free*( )
- *List Calendars*( )
- *Logoff*( )
- *Logon*( )
- *Look Up*( )
- *Query Configuration*( )
- *Restore*( )
- *Save*( )

NAME

Free - free memory allocated by the calendaring service.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_free(
    CSA_buffer    memory
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification. This function frees memory allocated by the calendaring service. After the call, the pointer **memory** will be invalid and should not be referenced again. When any XCS function allocates and returns a buffer to the application, the application will free that memory with this call when it is finished with the memory.

When a XCS function returns a base pointer to a complex structure containing several levels of pointers, all the application will do to free the entire structure or array of structures is call this routine with the base pointer returned by the XCS function. The XCS functions which return structures of this form are *csa_list_calendars*( ), *csa_look_up*( ), *csa_query_configuration*( ), *csa_list_calendar_attributes*( ), *csa_read_calendar_attributes*( ), *csa_free_time_search*( ), *csa_add_entry*( ), *csa_list_entries*( ), *csa_list_entry_attributes*( ), *csa_list_entry_sequence*( ), *csa_read_entry_attributes*( ), *csa_read_next_reminder*( ) and *csa_update_entry_attributes*( ).

The behaviour of *csa_free*( ) is undefined when called with a pointer to a memory block not allocated by the calendaring service, a pointer to a memory block that has already been freed, or a pointer contained in a structure returned by the XCS implementation.

In some situations, the extensions specified for a function may be a combination of input and output extensions. In this case, the memory returned in the output extensions must be freed one at a time using *csa_free*( ).

ARGUMENTS

Memory (Buffer)

A pointer to memory allocated by the calendaring service. A value of NULL will be ignored, but will return the return code CSA_SUCCESS..

RESULTS

Return Code (Return Code)

Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INVALID_MEMORY

NAME

List Calendars - list the calendars supported by a calendar service.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_list_calendars(
    CSA_service_reference      calendar_service,
    CSA_uint32                *number_names,
    CSA_calendar_user        **calendar_names,
    CSA_extension             *list_calendars_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported. This function lists all the calendars supported by the specified calendar service. The names of the calendars supported are returned in calendar_names, which is an array of **CSA_calendar_users** structures with **number_names** elements. It is implementation specific what authority a calendar user needs to invoke this function.

ARGUMENTS

Calendar Service (Service Reference)

Specifies the calendar service. A NULL pointer will reference the default calendar service name. If the calendar service name is invalid, then the error CSA_E_INVALID_CALENDAR_SERVICE is returned. If the user is not sufficiently authorised to list the calendars on the calendar service, then the error CSA_E_NO_AUTHORITY is returned.

List Calendars Extensions (Extension)

A pointer to an array of CSA_extension structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Names (Uint32)

A pointer to the number of calendar names returned in *calendar_names.* A value of zero indicates that no calendars are known to the calendar service. The error CSA_E_INSUFFICIENT_MEMORY is returned if the service has insufficient memory to store the complete set of requested information.

Calendar Names (Calendar User)

A pointer to the array of calendar user structures, representing the calendars supported by the specified calendar service. This pointer is allocated by the service, and should be freed with a single call to *csa_free*( ).

List Calendars Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extension structure for more information.

Return Code (Return Code)

Indicates whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_CALENDAR_SERVICE
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E-INVALID_PARAMETER
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Logoff - terminate a session with a calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_logoff(
    CSA_session_handle    session,
    CSA_extension        *logoff_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification. This function allows the calling application to terminate a session with a calendar.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service. It becomes invalid as a result of this call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Logoff Extensions (Extension)

A pointer to an array of CSA_extension structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Logoff Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)

Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_SESSION_HANDLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Logon - log on to the calendar service and establish a session with a calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_logon(
    CSA_service_reference     calendar_service,
    CSA_calendar_user        *user,
    CSA_string                password,
    CSA_string                character_set,
    CSA_string                required_csa_version,
    CSA_session_handle       *session,
    CSA_extension            *logon_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification.

This function allows the calling application to logon to the calendar service. If the specified calendar does not exist, then the error CSA_E_CALENDAR_NOT_EXIST is returned.

The function returns a Session Handle which the application will use in subsequent XCS calls.

ARGUMENTS

Calendar Service (Service Reference)

A calendar service name specifying the requested calendaring service, (for example, the path to a calendar store or a remote server node name). This value may be NULL if the underlying calendaring service does not require a service name or if UI is allowed. This may be necessary on some implementations and ignored on others.

User (Calendar User)

A pointer to the calendar user structure that identifies the user and calendar to the calendaring service. This value may be NULL.

Password (String)

A string containing the password required for access to the XCS service. This value may be NULL if the underlying calendaring service does not require a password or if UI is allowed.

Character Set (String)

A XCS formal public identifier string for the character set of strings used by the XCS caller. The client may pass NULL in which case the character set used is determined by the XCS service. The supported values are implementation specific.

Required XCS Version (String)

The formal public identifier for the XCS version number required by the application. For this version of the specification this string must be

```
"-//XAPIA/CSA/VERSION1/NONSGML CSA Version 1//EN".
```

Logon Extensions (Extensions)

A pointer to an array of CSA_extension structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

Through extensions, the application can find out which extensions are available and set which data extensions will be active for the session.

RESULTS

Session (Session Handle)
Opaque session handle that represents a session with the XCS calendar.

Logon Extensions (Extensions)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_CALENDAR_NOT_EXIST
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_CALENDAR_SERVICE
CSA_E_INVALID_CONFIGURATION
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_PASSWORD
CSA_E_INVALID_USER
CSA_E_NO_AUTHORITY
CSA_E_PASSWORD_REQUIRED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TOO_MANY_USERS
CSA_E_UNSUPPORTED_CHARACTER_SET
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_UNSUPPORTED_VERSION

NAME

Look Up - Looks up calendar information.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_look_up(
    CSA_session_handle    session,
    CSA_calendar_user    *users,
    CSA_flags             look_up_flags,
    CSA_uint32           *number_users,
    CSA_calendar_user   **user_list,
    CSA_extension        *look_up_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function looks up calendar addressing information in the directory provided by the XCS calendaring service. It primarily is used to resolves a users friendly name to a calendar address.

Multiple addresses may be returned. An array of calendar user descriptors is allocated and returned containing fully resolved information about each entry.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Users (Calendar User)

A pointer to a calendar user structure containing the user-friendly names whose calendar address is to be looked up. For name resolution, the name field in the structure contains the user name to be resolved. The user type can be set to provide information on desired resolution of the name. See the calendar user structure documentation for possible types.

For displaying calendar user details, the calendar user structure must contain an entry that resolves to only one user. If not, the error CSA_E_AMBIGUOUS_USER will be returned.

For both name resolution and displaying user details, all user structures except the first will be ignored.

Look Up Flags (Flags)

Bit mask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved. Flag settings include:

CSA_LOOKUP_RESOLVE_PREFIX_SEARCH
CSA_LOOKUP_RESOLVE_IDENTITY

CSA_LOOKUP_RESOLVE_PREFIX_SEARCH

If set, the search method should be prefix. Prefix search means that all names matching the prefix string, beginning at the first character of the name, will be matched. If not

set, the search method should be exact match. XCS implementations are required to support simple prefix searching. The availability of wild-card or substring searches is optional.

CSA_LOOKUP_RESOLVE_IDENTITY
If set, the function will return a user record for the identity of the user in the calendar system. If this cannot be uniquely determined, ambiguous name resolution will be carried out. This allows the application to find out the address of the current user.

Look Up Extensions (Extension)
A pointer to an array of CSA_extension structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Users (Uint32)
A pointer to the number of elements in user_list. If no names match the criteria, a value of zero is returned, and the error CSA_E_USER_NOT_FOUND is returned.

User List (Calendar User)
A pointer to an array of one or more calendar user structures allocated by *csa_look_up*( ). The structure may then be used to fill in an attendee list structure. This pointer is allocated by the service, and should be freed with a single call to *csa_free*( ).

Look Up Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_AMBIGUOUS_USER
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_USER_NOT_FOUND

NAME

Query Configuration - Determine information about the installed XCS configuration.

SYNOPSIS

```
#include <xcmc.h>

CSA_return_code
csa_query_configuration(
    CSA_session_handle    session,
    CSA_enum              item,
    CSA_buffer           *reference,
    CSA_extension        *query_configuration_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification.

This function queries the underlying implementation's configuration, and returns the information requested about it, allocating memory when necessary.

The underlying configuration file format is implementation dependent.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

Session may be NULL to indicate that there is no session and that session independent information should be returned. This will provide default logon information.

If this value is set to a valid Session Handle, session dependent configuration information will be returned.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Item (Enum)

This argument indicates which configuration information should be returned. If the specified item is not appropriate for the implementation, the error CSA_E_UNSUPPORTED_ENUM is returned. The possible values include:

CSA_CONFIG_CHARACTER_SET
CSA_CONFIG_LINE_TERM
CSA_CONFIG_DEFAULT_SERVICE
CSA_CONFIG_DEFAULT_USER
CSA_CONFIG_REQ_PASSWORD
CSA_CONFIG_REQ_SERVICE
CSA_CONFIG_REQ_USER
CSA_CONFIG_UI_AVAIL
CSA_CONFIG_VER_IMPLEM
CSA_CONFIG_VER_SPEC

CSA_CONFIG_CHARACTER_SET

The reference argument will be a pointer to the array of character set FPI strings for the implementation. The array will be terminated with a NULL string. The first character set FPI in the array is the default character set used if the caller does not specify one

explicitly. This pointer to the array should be freed using *csa_free*( ). This FPI is used by the caller at logon to specify to the implementation to use a different character set than the default.

CSA_CONFIG_LINE_TERM
: The reference argument will be a pointer to a CSA_enum variable, which will be set to a value of CSA_LINE_TERM_CRLF if the line delimiter is a carriage return followed by a line feed, CSA_LINE_TERM_LF if the line delimiter is a line feed, or CSA_LINE_TERM_CR if the line delimiter is a carriage return.

CSA_CONFIG_DEFAULT_SERVICE
: The returned reference argument will be a pointer to a CSA_string into which the default service name will be returned. A pointer value of NULL will be written if no default service name is available. This pointer should be freed using *csa_free*( ). This string, along with the one returned by CSA_CONFIG_DEFAULT_USER, can be used as defaults in user dialogs when asking for the service name, user name, and password. This will be returned in the implementation default character set.

CSA_CONFIG_DEFAULT_USER
: The reference argument will be a pointer to a CSA_string, into which the default user name will be returned. A pointer value of NULL will be written if no default user name is available. This pointer should be freed using *csa_free*( ). This string, along with the one returned by CSA_CONFIG_DEFAULT_SERVICE, can be used as defaults in user dialogs when asking for the provider name, user name, and password. This will be returned in the implementation default character set.

CSA_CONFIG_REQ_PASSWORD
: The reference argument will be a pointer to a CSA_enum variable, which will be set to a value of CSA_REQUIRED_NO if the password is not required to logon, CSA_REQUIRED_OPT if the password is optional to logon, or CSA_REQUIRED_YES if the password is required to logon.

CSA_CONFIG_REQ_SERVICE
: The reference argument will be a pointer to a CSA_enum variable, which will be set to a value of CSA_REQUIRED_NO if the service name is not required to logon, CSA_REQUIRED_OPT if the service name is optional to logon, or CSA_REQUIRED_YES if the service name is required to logon.

CSA_CONFIG_REQ_USER
: The reference argument will be a pointer to a CSA_enum variable, which will be set to a value of CSA_REQUIRED_NO if the user name is not required to logon, CSA_REQUIRED_OPT if the user name is optional to logon, or CSA_REQUIRED_YES if the user name is required to logon.

CSA_CONFIG_UI_AVAIL
: The reference argument will be a pointer to a CSA_boolean variable, which will be set to a true value if there is UI provided by the CSA implementation.

CSA_CONFIG_VER_IMPLEM
: The reference argument will be a pointer to a CSA_string variable, which will be set to the XCS formal public identifier for the version number for the implementation. This pointer should be freed using *csa_free*( ).

CSA_CONFIG_VER_SPEC
: The reference argument will be a pointer to a CSA_string variable, which will be set to the XCS formal public identifier for the XCS specification version number supported by this implementation. This pointer should be freed using *csa_free*( ).

The error CSA_E_UNSUPPORTED_ENUM is returned if the specified value is not supported by the implementation.

Query Configuration Extensions (Extension)
A pointer to an array of CSA_extension structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

Through extensions, the application can find out which extensions are available. The extension to do this is CSA_X_COM_SUPPORT_EXT. Any XCS implementation that supports extensions must support this extension. For more information on this extension, see the common extensions section of the extensions appendix in this document

RESULTS

Reference (Buffer)
This argument points to the buffer in which to receive the configuration information. The type of the variable or buffer depends on the item argument.

Query Configuration Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ENUM
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_ENUM
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Restore- restores calendar entries from an archive file.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_restore(
    CSA_session_handle   session,
    CSA_string           archive_name,
    CSA_uint32           number_attributes,
    CSA_attribute       *attributes,
    CSA_enum            *operators,
    CSA_extension       *restore_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function restores entries from the specified archive file into the calendar that is currently logged on to. Only calendar entries that match all the given criteria are restored. It is implementation specific whether new entry handles will be assigned to the entries restored. If *attributes* is NULL, all the entries in the archive file will be restored. If *operators* is NULL, this specifies a short hand for an array of operators of CSA_MATCH_EQUAL_TO. Only the owner of the calendar or users with CSA_OWNER_RIGHTS can restore an archive file.

The matching criteria are specified by the array of attributes and the array of operators. Each operator in operators specifies how the corresponding attribute value is to be matched. The following operators are supported:

CSA_MATCH_ANY
CSA_MATCH_EQUAL_TO
CSA_MATCH_NOT_EQUAL_TO
CSA_MATCH_GREATER_THAN
CSA_MATCH_LESS_THAN
CSA_MATCH_GREATER_THAN_OR_EQUAL_TO
CSA_MATCH_LESS_THAN_OR_EQUAL_TO
CSA_MATCH_CONTAIN

CSA_MATCH_ANY matches an entry that contains the corresponding attribute regardless of the value.

CSA_MATCH_EQUAL_TO matches an entry that contains an attribute with a value equal to the corresponding value.

CSA_MATCH_NOT_EQUAL_TO matches an entry that contains an attribute with a value not equal to the corresponding value.

CSA_MATCH_GREATER_THAN matches an entry that contains an attribute with a value greater than the corresponding value.

CSA_MATCH_LESS_THAN matches an entry that contains an attribute with a value less than the corresponding value.

CSA_MATCH_GREATER_THAN_OR_EQUAL_TO matches an entry that contains an attribute with a value greater than or equal to the corresponding value.

CSA_MATCH_LESS_THAN_OR_EQUAL_TO matches an entry that contains an attribute with a value less than or equal to the corresponding value.

CSA_MATCH_CONTAIN applies to character string values only. It matches an entry that contains the corresponding substring value. The only operators supported for reminder type attributes are CSA_MATCH_ANY and CSA_MATCH_EQUAL_TO. Searching of attributes with opaque data type is not supported.

ARGUMENTS

Session (Session Handle)
Opaque session handle which represents a session with the calendar service.

Session handles are created by a logon function call and invalidate with a logoff function call.

If the session handle is invalid, the error CSA_E_INVALID_SESSION_HANDLE is returned.

Archive Name (String)
A string containing the name of the archive file on the local system.

Number Attributes (Uint32)
Specifies the size of the arrays pointed to by *attributes* and *operators*. If the *attributes* argument is NULL, then this must be zero.

Attributes (Attribute)
A pointer to an array of attribute structures specifying the matching criteria. If the *number_attributes* argument is zero, then this must be NULL.

Operators (Enum)
A pointer to an array of matching operator flags for the corresponding attribute in *attributes*.

Restore Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Restore Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_FILE_NOT_EXIST
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_ENUM
CSA_E_INVALID_FLAG
CSA_E_INVALID_FILE_NAME

CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_RULE
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TEXT_TOO_LARGE
CSA_E_UNABLE_TO_OPEN_FILE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_CHARACTER_SET
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_UNSUPPORTED_VERSION

NAME

Save - saves calendar entries into an archive file.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_save(
    CSA_session_handle     session,
    CSA_string             archive_name,
    CSA_uint32             number_attributes
    CSA_attribute         *attributes,
    CSA_enum              *operators,
    CSA_boolean            delete_entry,
    CSA_extension         *save_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function saves entries from the calendar that is currently logged on to into the specified archive file. Only calendar entries that match all the given criteria are saved. The entry handles for the entries are not guaranteed to be maintained. If *attributes* is NULL, all the entries in the calendar will be saved. If *operators* is NULL, this specifies a short hand for an array of operators of CSA_MATCH_EQUAL_TO. Only the owner of the calendar or users with CSA_OWNER_RIGHTS can save entries into an archive file.

The matching criteria are specified by the array of attributes and the array of operators. Each operator in operators specifies how the corresponding attribute value is to be matched. The following operators are supported:

CSA_MATCH_ANY
CSA_MATCH_EQUAL_TO
CSA_MATCH_NOT_EQUAL_TO
CSA_MATCH_GREATER_THAN
CSA_MATCH_LESS_THAN
CSA_MATCH_GREATER_THAN_OR_EQUAL_TO
CSA_MATCH_LESS_THAN_OR_EQUAL_TO
CSA_MATCH_CONTAIN

CSA_MATCH_ANY matches an entry that contains the corresponding attribute regardless of the value.

CSA_MATCH_EQUAL_TO matches an entry that contains an attribute with a value equal to the corresponding value.

CSA_MATCH_NOT_EQUAL_TO matches an entry that contains an attribute with a value not equal to the corresponding value.

CSA_MATCH_GREATER_THAN matches an entry that contains an attribute with a value greater than the corresponding value.

CSA_MATCH_LESS_THAN matches an entry that contains an attribute with a value less than the corresponding value.

CSA_MATCH_GREATER_THAN_OR_EQUAL_TO matches an entry that contains an attribute with a value greater than or equal to the corresponding value.

CSA_MATCH_LESS_THAN_OR_EQUAL_TO matches an entry that contains an attribute with a value less than or equal to the corresponding value.

CSA_MATCH_CONTAIN applies to character string values only. It matches an entry that contains the corresponding substring value. The only operators supported for reminder type attributes are CSA_MATCH_ANY and CSA_MATCH_EQUAL_TO. Searching of attributes with opaque data type is not supported.

ARGUMENTS

Session (Session Handle)
Opaque session handle which represents a session with the calendar service.

Session handles are created by a logon function call and invalidate with a logoff function call.

If the session handle is invalid then, the error CSA_E_INVALID_SESSION_HANDLE is returned.

Archive Name (String)
A string containing the name of the archive file on the local system.

Number Attributes (Uint32)
Specifies the size of the arrays pointed to by *attributes* and *operators*. If *attribute* argument is NULL, then this must be zero.

Attributes (Attribute)
A pointer to an array of attribute structures specifying the matching criteria. If *number_attributes* argument is zero, then this must be NULL.

Operators (Enum)
A pointer to an array of matching operator flags for the corresponding attribute in attributes.

Delete Entry (Boolean)
Specifies whether the associated entries are to be deleted from the calendar after being archived into the archive file.

Save Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Save Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_FILE_EXIST
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE

CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_ENUM
CSA_E_INVALID_FLAG
CSA_E_INVALID_FILE_NAME
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TEXT_TOO_LARGE
CSA_E_UNABLE_TO_OPEN_FILE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

4.3 Calendar Management

The man-page definitions in this section provide the *Calendar management* functionality of the Calendaring and Scheduling interface.

These comprise the following functions:

- *Add Calendar*( )
- *Call Callbacks*( )
- *Delete Calendar*( )
- *List Calendar Attributes*( )
- *Read Calendar Attributes*( )
- *Register Callback Functions*( )
- *Unregister Callback Functions*( )
- *Update Calendar Attributes*( )

NAME

Add Calendar-add a calendar to the calendar service.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_add_calendar(
    CSA_session_handle    session,
    CSA_calendar_user    *user,
    CSA_uint32            number_attributes,
    CSA_attribute        *calendar_attributes,
    CSA_extension        *add_calendar_extensions
);
```

DESCRIPTION

This function creates a calendar on the calendar service.

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported. If the user does not have sufficient authority to add a calendar to the service, the error CSA_E_NO_AUTHORITY is returned. If the calendar already exists the error CSA_E_CALENDAR_EXISTS is returned.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service. Session handles are created by a logon function call and invalidated with a logoff function call. If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned. The session handle may be NULL if the calendar service does not require a session to be established in order to add a calendar.

User (Calendar User)

A pointer to a calendar user structure. This specifies the user and name of the calendar to be added to the calendar service.

Number Attributes (Uint32)

The number of elements in *calendar_attributes* argument.

Calendar Attributes (Attributes)

A pointer to an array of attribute structures that specify the calendar attributes for the new calendar.

Add Calendar Extensions (Extension)

A pointer to an array of CSA_extension structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Add Calendar Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_CALENDAR_EXISTS
CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_INVALID_USER
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_READONLY
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TEXT_TOO_LARGE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_CHARACTER_SET
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_ENUM
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Call Callbacks - force the invocation of the callback functions associated with the specified callback list(s).

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_call_callbacks(
    CSA_session_handle     session,
    CSA_flags              reason,
    CSA_extension         *call_callbacks_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function causes the service to call the registered callback functions associationed with the specified callback list(s). The service will process each specified callback list and call the registered callback functions if there have been changes that would trigger the callbacks of that type. The order in which callbacks are invoked is implementation specific.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service. If a valid session handle is specified, the callback functions registered with that session are invoked.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Reason (Flags)

A bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved. Each flag corresponds to a callback activity. Flag settings include:

CSA_CB_CALENDAR_LOGON
CSA_CB_CALENDAR_DELETED
CSA_CB_CALENDAR_ATTRIBUTE_UPDATED
CSA_CB_ENTRY_ADDED
CSA_CB_ENTRY_DELETED
CSA_CB_ENTRY_UPDATED.

CSA_CB_CALENDAR_LOGON - If set, the new callback will be invoked when a the calendar is accessed by a user.

CSA_CB_CALENDAR_DELETED - If set, the new callback will be invoked when a user requests that the calendar be deleted.

CSA_CB_CALENDAR_ATTRIBUTE_UPDATED - If set, the new callback will be invoked whenever a calendar attribute is updated.

CSA_CB_ENTRY_ADDED - If set, the new callback will be invoked whenever an entry is added to the calendar.

CSA_CB_ENTRY_DELETED - If set, the new callback will be invoked whenever an entry is deleted from the calendar.

CSA_CB_ENTRY_UPDATED - If set, the new callback will be invoked whenever an entry is updated on the calendar.

It is implementation specific if the callback function is invoked before or after the specified update type occurs.

Call Callbacks Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Call Callbacks Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Delete Calendar-delete a calendar from the calendar service.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_delete_calendar(
    CSA_session_handle      session,
    CSA_extension          *delete_calendar_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function deletes a calendar on the calendar service. Only the owner of the calendar, or a user with the owner authority, can remove the calendar from the calendar service.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Delete Calendar Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Delete Calendar Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)

Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_FLAG
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

List Calendar Attributes - list the names of the calendar attributes associated with a calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_list_calendar_attributes(
    CSA_session_handle          session,
    CSA_uint32                 *number_names,
    CSA_attribute_reference    **calendar_attributes_names,
    CSA_extension               *list_calendar_attributes_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function lists the names of the calendar attributes associated with a calendar. Using the returned calendar attribute name(s), the attribute value(s) may be read using the *csa_read_calendar_attributes*( ) function. Only the owner of the calendar, users with CSA_OWNER_RIGHTS, or users with CSA_VIEW_CALENDAR_ATTRIBUTES access rights can list the calendar attributes.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service. Session handles are created by a logon function call and invalidated with a logoff function call. If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

List Calendar Attributes Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Names (Uint32)

A pointer to the number of calendar attribute names returned in *calendar_attribute_names*.

Calendar Attribute Names (Attribute Reference)

A pointer to an array of attribute names containing the names of the calendar attributes in the calendar. This array is allocated by the service, and should be freed with a single call to *csa_free*( ).

List Calendar Attributes Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)

Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
XCS_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_FLAG
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Read Calendar Attributes - read and return the calendar attributes values for a calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_read_calendar_attributes(
    CSA_session_handle          session,
    CSA_uint32                  number_names,
    CSA_attribute_reference    *attribute_names,
    CSA_uint32                 *number_attributes,
    CSA_attribute             **calendar_attributes
    CSA_extension              *read_calendar_attributes_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function returns an array of attributes structures containing the values of the calendar attributes of the specified calendar. The function will return all of the attributes if *number_names* argument is zero and *attribute_names* argument is NULL. Only the owner of the calendar, users with CSA_OWNER_RIGHTS, or users with CSA_VIEW_CALENDAR_ATTRIBUTES access rights can read the calendar attributes.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Number Names (Uint32)

The number of calendar attribute names in attribute_names. If zero, then all of the attributes will be returned. If the attribute_names argument is NULL, then this must be zero.

Attribute Names (Attribute Names)

A pointer to an array of attribute names that are to be read. If NULL, then all of the attributes will be returned. If the number_names argument is zero, then this must be NULL.

Read Calendar Attributes Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Attributes (Uint32)
A pointer to the number of attributes returned in calendar_attributes. The error CSA_E_INSUFFICIENT_MEMORY is returned if there is the service has insufficient memory to store the complete set of requested information.

Calendar Attributes (Attribute)
A pointer to an array of attribute structures containing the values for the calendar attributes for the specified calendar. This array is allocated by the service, and should be freed with a single call to *csa_free*( ).

Read Calendar Attributes Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA-E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Register Callback Functions - register the callback functions to be invoked when the specified type of update occurs in the calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_register_callback(
    CSA_session_handle      session,
    CSA_flags               reason,
    CSA_callback            callback,
    CSA_buffer              client_data,
    CSA_extension          *register_callback_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function adds the callback procedure to the specified callback list for the session. The service maintains a set of callback lists, logically one list per callback activity type per session. The service reports changes to the state of the service or its calendars by invoking the appropriate callbacks in sequence when the client calls the csa_call_callbacks function or when the specified callback activity occurs.

The callback is only active for the duration of the calendar session.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Reason (Flags)

A bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved. Each flag corresponds to a callback activity and the specified procedure is added to the callback list for that activity type. Flag settings include:

CSA_CB_CALENDAR_LOGON
CSA_CB_CALENDAR_DELETED
CSA_CB_CALENDAR_ATTRIBUTE_UPDATED
CSA_CB_ENTRY_ADDED
CSA_CB_ENTRY_DELETED
CSA_CB_ENTRY_UPDATED.

CSA_CB_CALENDAR_LOGON - If set, the new callback will be invoked when a the calendar is accessed by a user.

CSA_CB_CALENDAR_DELETED - If set, the new callback will be invoked when a user requests that the calendar be deleted.

CSA_CB_CALENDAR_ATTRIBUTE_UPDATED - If set, the new callback will be invoked whenever a calendar attribute is updated.

CSA_CB_ENTRY_ADDED - If set, the new callback will be invoked whenever an entry is added to the calendar.

CSA_CB_ENTRY_DELETED - If set, the new callback will be invoked whenever an entry is deleted from the calendar.

CSA_CB_ENTRY_UPDATED - If set, the new callback will be invoked whenever an entry is updated on the calendar.

It is implementation specific if the callback function is invoked before or after the specified update type occurs.

Callback (Callback)
Specifies the client procedure that should be called by the service to handle the callback activity.

Client Data (Opaque Data)
A pointer to a data structure that will be passed to the callback function in its *client_data* argument. This can be used to pass client-specific data structure that will be needed by the callback function to perform the task.

Register Callback Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Register Callback Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Unregister Callback Functions - unregister the specified callback functions.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_unregister_callback(
    CSA_session_handle      session,
    CSA_flags               reason,
    CSA_callback            callback,
    CSA_buffer              client_data,
    CSA_extension          *unregister_callback_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function removes the specified callback procedure to the specified callback list. Both the procedure and the client data must match for this function to remove the procedure.

Specifying a value of NULL for both callback and client data will cause all callbacks on the specified callback list(s) to be removed.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Reason (Flags)

A bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved. Each flag corresponds to a callback list. Flag settings include:

CSA_CB_CALENDAR_LOGON
CSA_CB_CALENDAR_DELETED
CSA_CB_CALENDAR_ATTRIBUTE_UPDATED
CSA_CB_ENTRY_ADDED
CSA_CB_ENTRY_DELETED
CSA_CB_ENTRY_UPDATED.

CSA_CB_CALENDAR_LOGON - If set, the new callback will be invoked when a the calendar is accessed by a user.

CSA_CB_CALENDAR_DELETED - If set, the new callback will be invoked when a user requests that the calendar be deleted.

CSA_CB_CALENDAR_ATTRIBUTE_UPDATED - If set, the new callback will be invoked whenever a calendar attribute is updated.

CSA_CB_ENTRY_ADDED - If set, the new callback will be invoked whenever an entry is added to the calendar.

CSA_CB_ENTRY_DELETED - If set, the new callback will be invoked whenever an entry is deleted from the calendar.

CSA_CB_ENTRY_UPDATED - If set, the new callback will be invoked whenever an entry is updated on the calendar.

It is implementation specific if the callback function is invoked before or after the specified update type occurs.

Callback (Callback)
Specifies the client procedure that should be removed from the specified callback list(s).

Client Data (Opaque Data)
A pointer to an application specific data structure that was specified when the callback procedure was registered This must match for the function to succeed. This constraint allows the application to register the same function on more than one callback list with different client_data. The instances of the callback function on the different callback lists will be treated independently by the calendar service.

Unregister Callback Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Unregister Callback Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_CALLBACK_NOT_REGISTERED
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Update Calendar Attributes - update the calendar attributes values for a calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_update_calendar_attributes(
    CSA_session_handle     session,
    CSA_uint32             number_attributes,
    CSA_attribute         *calendar_attributes,
    CSA_extension         *update_calendar_attributes_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function update the values of the calendar attributes of the specified calendar. The existing value of each specified attribute will be replaced by the new value specified in *calendar_attributes*. New attributes can be added using this function and existing attributes can be effectively deleted by setting the value part of the **CSA_attributes** structure to NULL. If a read-only attribute is specified, the error CSA_E_READ_ONLY is returned by the function. If the function returns an error, none of the specified attributes will be updated. Only the owner of the calendar, users with CSA_OWNER_RIGHTS, users with CSA_INSERT_CALENDAR_ATTRIBUTES access rights, or users with CSA_UPDATE_CALENDAR_ATTRIBUTES, can update the calendar attributes.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Number Attributes (Uint32)

The number of calendar attributes in *calendar_attributes*.

Calendar Attributes (Attribute)

A pointer to an array of attribute structures containing the new values for the calendar attributes.

Update Calendar Attributes Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Update Calendar Attributes Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_READ_ONLY
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TEXT_TOO_LARGE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

4.4 Entry Management

The man-page definitions in this section provide the *entry management* functionality of the Calendaring and Scheduling interface.

These comprise the following functions:

- *Add Entry*( )
- *Delete Entry*( )
- *Free Time Search*( )
- *List Entries*( )
- *List Entry Attributes*( )
- *List Entry Sequence*( )
- *Read Entry Attributes*( )
- *Read Next Reminders*( )
- *Update Entry Attributes*( )

NAME

Add Entry - add an entry to the specified calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_add_entry(
    CSA_session_handle      session,
    CSA_uint32              number_attributes,
    CSA_attribute          *entry_attributes,
    CSA_entry_handle       *entry,
    CSA_extension          *add_entry_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification.

This function adds a new entry to a calendar. The handle for the new entry is returned. Only the owner of the calendar, users with CSA_OWNER_RIGHTS, users with CSA_INSERT_PUBLIC_ENTRIES, user with CSA_INSERT_CONFIDENTIAL_ENTRIES, or users with CSA_INSERT_PRIVATE_ENTRIES access rights can add entries to the calendar.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Number Attributes (Uint32)

Specifies the number of attributes in the *entry_attributes* argument.

Entry Attributes (Attribute)

A pointer to an array of entry attributes that will be used to define the new entry.

Add Entry Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Entry (Entry Handle)

A pointer to the handle for the new entry. This handle is allocated by the service, and should be freed with a single call to *csa_free*( ).

Add Entry Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)

Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_RULE
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_READ_ONLY
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TEXT_TOO_LARGE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Delete Entry - delete an entry from a calendar.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_delete_entry(
    CSA_session_handle      session,
    CSA_entry_handle        entry,
    CSA_enum                delete_scope,
    CSA_extension          *delete_entry_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification.

This function will delete an entry in a calendar. The *delete_scope* indicates the scope of the deletion if the entry has any associated recurring entries. All of the recurring entries can be deleted, only the specified entry can be deleted, or only the recurring entries that follow the specified entry can be deleted. Only the calendar owner, users with CSA_OWNER_RIGHTS, users with CSA_UPDATE_PUBLIC_ENTRIES, users with CSA_UPDATE_CONFIDENTIAL_ENTRIES, or users with CSA_UPDATE_PRIVATE_ENTRIES access rights can delete entries on the calendar.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Entry (Entry Handle)

The handle of the calendar entry to be deleted.

If the entry handle is invalid, then the error CSA_E_INVALID_ENTRY_HANDLE is returned.

Delete Scope (Enum)

The scope for the delete operation. Operator settings include:

CSA_SCOPE_ALL
CSA_SCOPE_ONE
CSA_SCOPE_FORWARD

CSA_SCOPE_ALL - If set and the entry is associated with recurring entries, it specifies that all of the recurring entries are to be deleted.

CSA_SCOPE_ONE - If set and the entry is associated with recurring entries, it specifies that only the specified entry is to be deleted.

CSA_SCOPE_FORWARD - If set and the entry is associated with recurring entries, it specifies that only the recurring entries following the specified entry are to be deleted. The scope also includes the specified entry.

Delete Entry Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Delete Entry Extensions (Extension)

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)

Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INVALID_ENTRY_HANDLE
CSA_E_INVALID_ENUM
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_ENUM
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Free Time Search - searches one or more calendars for available free time.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_free_time_search(
    CSA_session_handle        session,
    CSA_date_time_range       date_time_range,
    CSA_time_duration         time_duration,
    CSA_uint32                number_users,
    CSA_calendar_user        *calendar_users,
    CSA_free_time           **free_time,
    CSA_extension           *free_time_search_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function searches for available free time on one or more calendars and returns a list of free time intervals found. Free time is an interval of time that is not currently scheduled on one or more calendars. The free time search is based on a date and time range and the minimum time duration required of the free time interval. Only the owner of the calendar, users with CSA_FREE_TIME_SEARCH, users with CSA_VIEW_PUBLIC_ENTRIES, users with CSA_VIEW_CONFIDENTIAL_ENTRIES, or users with CSA_VIEW_PRIVATE_ENTRIES can search the calendar for free time.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service. Session handles are created by a logon function call and invalidated with a logoff function call. If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Date Time Range (Date Time Range)

Specifies the start and stop date and time range for the free time search. If the *date_time_range* is invalid, then the error CSA_E_INVALID_DATE_TIME is returned.

Time Duration (Time Duration)

Specifies the minimum time range for the free time intervals to be sought. If the time duration is invalid, then the error CSA_E_INVALID_DATE_TIME is returned.

Number Users (Uint32)

The number of elements in the *calendar_users* argument.

Calendar Users (Calendar User)

A pointer to an array of calendar users. This specifies the calendars to be searched for available free time. If the user does not have authority to access any of the attendee's calendars, then the error CSA_E_NO_AUTHORITY is returned. If an invalid attendee is specified, then the error CSA_E_INVALID_USER is returned. It is implementation specific whether the search function will proceed in any of these cases.

Free Time Search Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Free Time (Free Time)
A pointer to a free time structure that specifies a set of free time intervals. If more free time intervals are found than can be represented in the available memory, the CSA_E_INSUFFICIENT_MEMORY error message will be returned. If no available free time is found, then a NULL pointer will be returned. This structure is allocated by the service, and should be freed with a single call to *csa_free*( ).

Free Time Search Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_INVALID_USER
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

List Entries - list the calendar entries that match all the attribute search criteria.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_list_entries(
    CSA_session_handle      session,
    CSA_uint32              number_attributes,
    CSA_attribute          *entry_attributes
    CSA_enum               *list_operators,
    CSA_uint32             *number_entries,
    CSA_entry_handle      **entries,
    CSA_extension          *list_entries_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification.

This function lists the entry handles for the calendar entries that match all the attribute search criteria. Using the returned entry handles, the entries can have their attributes listed and read, or the entries can updated or deleted. Only the calendar owner, users with CSA_OWNER_RIGHTS, users with CSA_VIEW_PUBLIC_ENTRIES, user with CSA_VIEW_CONFIDENTIAL_ENTRIES, or users with CSA_VIEW_PRIVATE_ENTRIES access rights can list entries in the calendar.

If *list_operators* is NULL, this specifies a short hand for an array of operators of CSA_MATCH_EQUAL_TO. The criteria are specified by the array of attributes and the array of operators. Each operator in operators specifies how the corresponding attribute value is to be matched. The following operators are supported:

CSA_MATCH_ANY
CSA_MATCH_EQUAL_TO
CSA_MATCH_NOT_EQUAL_TO
CSA_MATCH_GREATER_THAN
CSA_MATCH_LESS_THAN
CSA_MATCH_GREATER_THAN_OR_EQUAL_TO
CSA_MATCH_LESS_THAN_OR_EQUAL_TO
CSA_MATCH_CONTAIN

CSA_MATCH_ANY matches an entry that contains the corresponding attribute regardless of the value.

CSA_MATCH_EQUAL_TO matches an entry that contains an attribute with a value equal to the corresponding value.

CSA_MATCH_NOT_EQUAL_TO matches an entry that contains an attribute with a value not equal to the corresponding value.

CSA_MATCH_GREATER_THAN matches an entry that contains an attribute with a value greater than the corresponding value.

CSA_MATCH_LESS_THAN matches an entry that contains an attribute with a value less than the corresponding value.

CSA_MATCH_GREATER_THAN_OR_EQUAL_TO matches an entry that contains an attribute with a value greater than or equal to the corresponding value.

CSA_MATCH_LESS_THAN_OR_EQUAL_TO matches an entry that contains an attribute with a value less than or equal to the corresponding value.

CSA_MATCH_CONTAIN applies to character string values only. It matches an entry that contains the corresponding substring value. The only operators supported for reminder type attributes are CSA_MATCH_ANY and CSA_MATCH_EQUAL_TO. Searching of attributes with opaque data type is not supported.

It is implementation specific in what order the array of entries will be returned. If either *number_attributes* is zero or *entry_attributes* is NULL, then all of the entries will be returned.

ARGUMENTS

Session (Session Handle)
Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Number Attributes (Uint32)
Specifies the size of the arrays pointed to by *entry_attributes* and *list_operators*. If *entry_attributes* is NULL, then this must be zero.

Entry Attributes (Attribute)
A pointer to an array of attribute structures specifying the matching criteria. If *number_attributes* is zero, then this must be NULL.

List Operators (Enum)
A pointer to an array of matching operators.

List Entries Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Entries (Uint32)
The number of entry handles in entries. If this value is NULL, then no entries were found to match the search criteria. If more entries were found than can be represented in the available memory, then the error CSA_E_INSUFFICIENT_MEMORY will be returned.

Entries (Entry Handle)
A pointer to an array of entry handles that match all the search criteria. This array is allocated by the service and should be freed with a single call to *csa_free*( ).

List Entries Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_ENUM
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_ENUM
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

List Entry Attributes - list the names of the entry attributes associated with the specified entry.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_list_entry_attributes(
    CSA_session_handle           session,
    CSA_entry_handle             entry,
    CSA_uint32                  *number_names,
    CSA_attribute_reference    **entry_attributes_names
    CSA_extension               *list_entry_attributes_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification. This function lists the names of the entry attributes associated with a calendar entry. Using the returned entry attributes name(s), the attributes value(s) may be read using the *csa_read_entry_attributes*( ) function.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Entry (Entry Handle)

Specifies the handle of the calendar entry that will have its attributes listed.

If the entry handle is invalid, then the error CSA_E_INVALID_ENTRY_HANDLE is returned.

List Entry Attributes Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Names (Uint32)

A pointer to the number of attribute names returned in *entry_attribute_names*.

Entry Attribute Names (Attribute Reference)

A pointer to an array of attribute references containing the names of the entry attributes in the calendar entry. This array is allocated by the service, and should be freed with a single call to *csa_free*( ).

List Entry Attributes Extensions (Extension)'

If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ENTRY_HANDLE
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

List Entry Sequence - lists the recurring calendar entries that are associated with a calendar entry.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_list_entry_sequence(
    CSA_session_handle       session,
    CSA_entry_handle         entry,
    CSA_date_time_range      time_range,
    CSA_uint32              *number_entries,
    CSA_entry_handle       **entry_list,
    CSA_extension           *list_entry_sequence_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function returns an array of the entry handles for the recurring entries associated with a specific calendar entry.

The entry handles for the recurring calendar entries are returned in entry_list. A NULL is returned if no recurring entries are associated with this calendar entry.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Entry (Entry Handle)

Specifies the handle of the calendar entry to be accessed for a list of associated recurring entries."

If the entry handle is invalid, then the error CSA_E_INVALID_ENTRY_HANDLE is returned.

Time Range (Date Time Range)

The date and time range to be used to filter or restrict the listing of the entry sequence. The date and time range specifies a begin and end date and time. If NULL, then all entries in the sequence will be returned. If the sequence is of an indefinite repetition, then the function will return implementation specific results.

List Entry Sequence Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Entries (Uint32)
The number of calendar entry handles actually returned.

Entry List (Entry Handle)
A pointer to an array of calendar entry handles that represent the recurring entries associated with the specified calendar entry. This array is allocated by the service, and the entire array should be freed with a single call to *csa_free*( ).

List Entry Sequence Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_ENTRY_HANDLE
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Read Entry Attributes - read and return the calendar entry attribute values for a specified calendar entry.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_read_entry_attributes(
    CSA_session_handle          session,
    CSA_entry_handle            entry,
    CSA_uint32                  number_names,
    CSA_attribute_reference    *attribute_names,
    CSA_uint32                 *number_attributes,
    CSA_attribute              **entry_attributes,
    CSA_extension              *read_entry_attributes_extensions
);
```

DESCRIPTION

Support for this function is mandatory for conformation to the XCS interface specification.

This function returns an array of attribute structures containing the values of the attributes of the specified calendar entry. The function will return all of the attributes if *number_names* argument is zero and *attribute_names* argument is NULL.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Entry (Entry Handle)

The handle of the calendar entry to be read. If the entry handle is invalid, then the error CSA_E_INVALID_ENTRY_HANDLE is returned.

Number Names (Uint32)

The number of names in *attribute_names*. If attribute_names argument is NULL, then this must be zero.

Attribute Names (Attribute Reference)

A pointer to an array of attribute names that reference the attributes that are to be read. If *number_names* argument is zero, then this must be NULL.

Read EntryAttribute Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Attributes (Uint32)

The number of attributes returned in entry_attributes. If none of the specified attributes have values, a value of zero is returned.

Entry Attributes (Attribute)
A pointer to an array of attributes. This pointer is allocated by the service and should be freed with a single call to *csa_free*( ).

Read Entry Attribute Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ENTRY_HANDLE
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Read Next Reminder - reads the next reminder of the given type in the specified calendar relative to a given time.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_read_next_reminder(
    CSA_session_handle          session,
    CSA_uint32                  number_names,
    CSA_attribute_reference    *reminder_names,
    CSA_date_time               given_time,
    CSA_uint32                 *number_reminders
    CSA_reminder_reference    **reminder_references,
    CSA_extension               *read_next_reminder_extensions
);
```

DESCRIPTION

Support for this function is optional for conformance to the XCS interface specification. The error CSA_E_NOT_SUPPORTED is returned if the function is not supported.

This function reads the next reminder of the specified type in the specified calendar relative to a given time. More than one type of reminder may be specified. For each reminder type specified, the next reminder of that type after the given time will be returned. The owner of the calendar or users with CSA_OWNER_RIGHTS access right can read the next reminder for a calendar.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Number Names (Uint32)

Specifies the size of *reminder_names*.

Reminder Names (Attribute Reference)

A pointer to an array of attribute references. This is an array of reminder attribute names. The names are used as search criteria to return the next reminder of each type. If NULL, then the first reminder after the given time will be returned, no matter what type it is. If an invalid reminder type is specified, then the error CSA_E_INVALID_ATTRIBUTE is returned.

Given Time (Date Time)

The given date and time after which the search for the next reminder is to begin. If the date and time value is incorrect, then the error CSA_E_INVALID_DATE_TIME will be returned.

Read Next Reminder Extensions (Extension)

A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

Number Reminders (Uint32)
The number of reminder reference structures returned in *reminder_references*. If no reminders were found a value of zero is returned.

Reminder References (Reminder Reference)
A pointer to an array of reminder reference structures. This array is allocated by the service, and should be freed with a single call to *csa_free*( ).

Read Next Reminder Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_DATE_TIME
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_NOT_SUPPORTED
CSA_E_SERVICE_UNAVAILABLE
CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT

NAME

Update Entry Attributes - update the calendar entry attributes.

SYNOPSIS

```
#include <xcsa.h>

CSA_return_code
csa_update_entry_attributes(
    CSA_session_handle       session,
    CSA_entry_handle         entry,
    CSA_enum                 update_scope,
    CSA_boolean              update_propagation,
    CSA_uint32               number_attributes,
    CSA_attribute           *entry_attributes,
    CSA_entry_handle        *new_entry,
    CSA_extension           *update_entry_attributes_extensions
);
```

DESCRIPTION

Support for this function is mandatory for implementations conforming to this specification.

This function updates the values of the entry attributes of the specified calendar entry. The existing value of each specified attribute will be replaced by the new value specified in *entry_attributes*. New attributes can be added using this function and existing attributes can be effectively deleted by setting the value part of the *entry_attributes* structure to NULL. If a read-only attribute is specified, the error CSA_E_READONLY will be returned. If the function returns an error, none of the specified attributes will be updated. Only the owner of the calendar, or users with CSA_OWNER_RIGHTS, users with CSA_CHANGE_PUBLIC_ENTRIES, users with CSA_CHANGE_CONFIDENTIAL_ENTRIES, or users with CSA_CHANGE_PRIVATE_ENTRIES can update the entry attributes.

ARGUMENTS

Session (Session Handle)

Opaque session handle which represents a session with the calendaring service.

Session handles are created by a logon function call and invalidated with a logoff function call.

If the session handle is invalid, then the error CSA_E_INVALID_SESSION_HANDLE is returned.

Entry (Entry Handle)

The handle of the calendar entry to be updated. If the entry handle is invalid, then the error CSA_E_INVALID_ENTRY_HANDLE is returned.

Update Scope (Enum)

Specifies the scope of the entry update. The value is one of:

CSA_SCOPE_ALL
CSA_SCOPE_ONE
CSA_SCOPE_FORWARD

CSA_SCOPE_ALL - Specifies that the scope of the update is for all recurring entries associated with the specified entry:

CSA_SCOPE_ONE - Specifies that the scope of the update is for just the specified entry.

CSA_SCOPE_FORWARD - Specifies that the scope of the update is for those recurring entries subsequent to the specified entry. The scope also includes the specified entry.

Update Propagation (Boolean)
The update propagation flag. A value of **FALSE** indicates that the calendar service is not to propagate updates to the entry to the attendees' calendars. A value of **TRUE** indicates that the calendar service is to attempt to propagate updates to the entry to the attendees' calendars. Propagation of updates is an implementation specific feature. Implementations that do not support update propagation will return the error CSA_E_UNSUPPORTED_PARAMETER if a value other than **FALSE** is specified.

Number Attributes (Uint32)
The number of attributes that will be used to define the entry updates.

Entry Attributes (Attribute)
A pointer to an array of attributes that will be used to define the entry updates.

Update Entry Attributes Extensions (Extension)
A pointer to an array of **CSA_extension** structures for this function. The array may contain both input extensions for providing additional information to the function and output extensions for receiving information from the function. A value of NULL indicates that the caller is not using any extensions. See the extensions structure for more information.

RESULTS

New Entry (Entry Handle)
A pointer to the handle of the updated calendar entry.

If this value is NULL, then the implementation did not need to create a new entry handle for the updated entry. This handle is allocated by the service and should be freed with a single call to *csa_free*( ).

Update Entry Attributes Extensions (Extension)
If output extensions were passed to the function in the extensions list, the results from the service will be available in the extension. See the extensions structure for more information.

Return Code (Return Code)
Whether the function succeeded or not, and, if not, why. It may be success or one of the values listed under ERRORS below.

ERRORS

CSA_E_DISK_FULL
CSA_E_FAILURE
CSA_E_INSUFFICIENT_MEMORY
CSA_E_INVALID_ATTRIBUTE
CSA_E_INVALID_ATTRIBUTE_VALUE
CSA_E_INVALID_DATA_EXT
CSA_E_INVALID_ENTRY_HANDLE
CSA_E_INVALID_ENUM
CSA_E_INVALID_FLAG
CSA_E_INVALID_FUNCTION_EXT
CSA_E_INVALID_PARAMETER
CSA_E_INVALID_SESSION_HANDLE
CSA_E_NO_AUTHORITY
CSA_E_READ_ONLY
CSA_E_SERVICE_UNAVAILABLE
CSA_E_TEXT_TOO_LARGE

CSA_E_UNSUPPORTED_ATTRIBUTE
CSA_E_UNSUPPORTED_DATA_EXT
CSA_E_UNSUPPORTED_ENUM
CSA_E_UNSUPPORTED_FLAG
CSA_E_UNSUPPORTED_FUNCTION_EXT
CSA_E_UNSUPPORTED_PARAMETER

4.5 Return Codes

This section defines the return codes of the XCS interface. The return codes of the generic interface are specified here; the return codes of the C interface are specified in Chapter 5.

The XCS implementation should only return the values that pertain to a specific function if possible. If necessary the implementation may return other errors in the error list that are not specifically assigned to a function. It is not recommended that errors not in the list below be returned.

The generic return codes and the Simple Calendaring and Scheduling Interface function to which the return codes pertain are defined as follows:

Return Code	Add Event	Add Todo	Add Memo
CSA_E_AMBIGUOUS_USER			
CSA_E_CALENDAR_EXISTS			
CSA_E_CALENDAR_NOT_EXIST	X	X	X
CSA_E_CALLBACK_NOT_REGISTERED			
CSA_E_DISK_FULL	X	X	X
CSA_E_FAILURE	X	X	X
CSA_E_FILE_EXIST			
CSA_E_FILE_NOT_EXIST			
CSA_E_INSUFFICIENT_MEMORY			
CSA_E_INVALID_ATTRIBUTE			
CSA_E_INVALID_ATTRIBUTE_VALUE			
CSA_E_INVALID_CALENDAR_SERVICE	X	X	X
CSA_E_INVALID_CONFIGURATION			
CSA_E_INVALID_DATA_EXT			
CSA_E_INVALID_DATE_TIME	X	X	X
CSA_E_INVALID_ENTRY_HANDLE			
CSA_E_INVALID_ENUM	X	X	
CSA_E_INVALID_FILE_NAME			
CSA_E_INVALID_FLAG			
CSA_E_INVALID_FUNCTION _EXT	X	X	X
CSA_E_INVALID_MEMORY			
CSA_E_INVALID_PARAMETER	X	X	X
CSA_E_INVALID_PASSWORD	X	X	X
CSA_E_INVALID_RULE	X		
CSA_E_INVALID_SESSION_HANDLE			
CSA_E_INVALID_USER	X	X	X
CSA_E_NO_AUTHORITY			
CSA_E_NOT_SUPPORTED	X	X	X
CSA_E_PASSWORD_REQUIRED	X	X	X
CSA_E_READONLY			
CSA_E_SERVICE_UNAVAILABLE	X	X	X
CSA_E_TEXT_TOO_LARGE			
CSA_E_TOO_MANY_USERS	X	X	X
CSA_E_UNABLE_TO_OPEN_FILE			
CSA_E_UNSUPPORTED_ATTRIBUTE			
CSA_E_UNSUPPORTED_CHARACTER_SET			
CSA_E_UNSUPPORTED_DATA_EXT			
CSA_E_UNSUPPORTED_ENUM			
CSA_E_UNSUPPORTED_FLAG			
CSA_E_UNSUPPORTED_FUNCTION_EXT	X	X	X
CSA_E_UNSUPPORTED_PARAMETER			
CSA_E_UNSUPPORTED_VERSION			
CSA_E_USER_NOT_FOUND			

Table 4-2 Simple XCS Interface Function Return Codes

The generic return codes and the administrative function to which the return codes pertain are defined as follows:

Return Code	Free	List Calen-dars	Logoff	Logon	Look Up uration	Query Config-	Restore	Save
CSA_E_AMBIGUOUS_USER					X			
CSA_E_CALENDAR_EXISTS								
CSA_E_CALENDAR_NOT_EXIST				X				
CSA_E_CALLBACK_NOT_REGISTERED								
CSA_E_DISK_FULL							X	X
CSA_E_FAILURE	X	X	X	X	X	X	X	X
CSA_E_FILE_EXIST								X
CSA_E_FILE_NOT_EXIST							X	
CSA_E_INSUFFICIENT_MEMORY		X	X	X	X	X	X	X
CSA_E_INVALID_ATTRIBUTE							X	X
CSA_E_INVALID_ATTRIBUTE_VALUE							X	X
CSA_E_INVALID_CALENDAR_SERVICE		X		X				
CSA_E_INVALID_CONFIGURATION			X					
CSA_E_INVALID_DATA_EXT			X	X		X	X	
CSA_E_INVALID_DATE_TIME								
CSA_E_INVALID_ENTRY_HANDLE								
CSA_E_INVALID_ENUM						X	X	X
CSA_E_INVALID_FILE_NAME							X	X
CSA_E_INVALID_FLAG		X	X	X	X	X	X	X
CSA_E_INVALID_FUNCTION_EXT		X	X	X	X	X	X	X
CSA_E_INVALID_MEMORY	X							
CSA_E_INVALID_PARAMETER		X		X	X	X	X	X
CSA_E_INVALID_PASSWORD				X				
CSA_E_INVALID_RULE							X	
CSA_E_INVALID_SESSION_HANDLE			X		X	X	X	X
CSA_E_INVALID_USER				X				
CSA_E_NO_AUTHORITY				X			X	X
CSA_E_NOT_SUPPORTED		X			X		X	X
CSA_E_PASSWORD_REQUIRED				X				
CSA_E_READONLY								
CSA_E_SERVICE_UNAVAILABLE		X		X	X	X		X
CSA_E_TEXT_TOO_LARGE							X	X
CSA_E_TOO_MANY_USERS				X				
CSA_E_UNABLE_TO_OPEN_FILE							X	X
CSA_E_UNSUPPORTED_ATTRIBUTE							X	X
CSA_E_UNSUPPORTED_CHARACTER_SET				X			X	
CSA_E_UNSUPPORTED_DATA_EXT				X	X		X	X
CSA_E_UNSUPPORTED_ENUM							X	X
CSA_E_UNSUPPORTED_FLAG			X	X	X	X	X	X
CSA_E_UNSUPPORTED_FUNCTION_EXT		X	X	X	X	X	X	X
CSA_E_UNSUPPORTED_PARAMETER								
CSA_E_UNSUPPORTED_VERSION				X			X	
CSA_E_USER_NOT_FOUND					X			

Table 4-3 XCS Administrative Function Return Codes

The generic return codes and the calendar management function to which the return codes pertain are defined as follows:

Return Codes	Add Calendar	Call Callbacks	Delete Calendar	List Calendar Attributes	Read Calendar Attributes	Register Callback Functions	Unregister Call-Callback Functions	Update endar Attributes
CSA_E_AMBIGUOUS_USER								
CSA_E_CALENDAR_EXISTS								
CSA_E_CALENDAR_NOT_EXIST	X							
CSA_E_CALLBACK_NOT_REGISTERED							X	
CSA_E_DISK_FULL	X		X					X
CSA_E_FAILURE	X	X	X	X	X	X	X	X
CSA_E_FILE_EXIST								
CSA_E_FILE_NOT_EXIST								
CSA_E_INSUFFICIENT_MEMORY	X		X	X	X	X	X	X
CSA_E_INVALID_ATTRIBUTE	X				X			
CSA_E_INVALID_ATTRIBUTE_VALUE	X						X	
CSA_E_INVALID_CALENDAR_SERVICE								
CSA_E_INVALID_CONFIGURATION								
CSA_E_INVALID_DATA_EXT	X							X
CSA_E_INVALID_DATE_TIME								
CSA_E_INVALID_ENTRY_HANDLE								
CSA_E_INVALID_ENUM								
CSA_E_INVALID_FILE_NAME								
CSA_E_INVALID_FLAG	X	X	X	X	X	X	X	X
CSA_E_INVALID_FUNCTION_EXT	X	X	X	X	X	X	X	X
CSA_E_INVALID_MEMORY								
CSA_E_INVALID_PARAMETER	X	X	X	X	X	X	X	X
CSA_E_INVALID_PASSWORD								
CSA_E_INVALID_RULE								
CSA_E_INVALID_SESSION_HANDLE	X	X	X	X	X	X	X	X
CSA_E_INVALID_USER								
CSA_E_NO_AUTHORITY	X		X	X	X	X	X	X
CSA_E_NOT_SUPPORTED	X	X	X	X	X	X	X	X
CSA_E_PASSWORD_REQUIRED								
CSA_E_READONLY	X							X
CSA_E_SERVICE_UNAVAILABLE	X	X	X	X	X	X	X	X
CSA_E_TEXT_TOO_LARGE	X							X
CSA_E_TOO_MANY_USERS								
CSA_E_UNABLE_TO_OPEN_FILE								
CSA_E_UNSUPPORTED_ATTRIBUTE	X					X		X
CSA_E_UNSUPPORTED_CHARACTER_SET	X							
CSA_E_UNSUPPORTED_DATA_EXT	X							X
CSA_E_UNSUPPORTED_ENUM	X							
CSA_E_UNSUPPORTED_FLAG	X	X	X	X	X	X	X	X
CSA_E_UNSUPPORTED_FUNCTION_EXT	X	X	X	X	X	X	X	X
CSA_E_UNSUPPORTED_PARAMETER								
CSA_E_UNSUPPORTED_VERSION								
CSA_E_USER_NOT_FOUND								

Table 4-4 XCS Calendar Management Function Return Codes

The generic return codes and the entry management function to which the return codes pertain are defined as follows:

Return Code	Add Entry	Delete Entry	Free Time Search	List Entries	List Entry Attri-butes	List Entry Seq-uence	Read Entry Attri-butes	Read Next Remin-ders	Update Entry Attri-butes
CSA_E_AMBIGUOUS_USER									
CSA_E_CALENDAR_EXISTS									
CSA_E_CALENDAR_NOT_EXIST									
CSA_E_CALLBACK_NOT_REGISTERED									
CSA_E_DISK_FULL	X	X							X
CSA_E_FAILURE	X	X	X	X	X	X	X	X	X
CSA_E_FILE_EXIST									
CSA_E_FILE_NOT_EXIST									
CSA_E_INSUFFICIENT_MEMORY	X		X	X	X	X	X	X	X
CSA_E_INVALID_ATTRIBUTE	X			X			X	X	X
CSA_E_INVALID_ATTRIBUTE_VALUE	X			X					X
CSA_E_INVALID_CALENDAR_SERVICE									
CSA_E_INVALID_CONFIGURATION									
CSA_E_INVALID_DATA_EXT	X		X	X					X
CSA_E_INVALID_DATE_TIME			X			X		X	
CSA_E_INVALID_ENTRY_HANDLE		X			X	X	X		X
CSA_E_INVALID_ENUM		X		X					X
CSA_E_INVALID_FILE_NAME									
CSA_E_INVALID_FLAG	X	X	X	X	X	X	X	X	X
CSA_E_INVALID_FUNCTION_EXT	X	X	X	X	X	X	X	X	X
CSA_E_INVALID_MEMORY									
CSA_E_INVALID_PARAMETER	X	X	X	X	X	X	X	X	X
CSA_E_INVALID_PASSWORD									
CSA_E_INVALID_RULE	X								X
CSA_E_INVALID_SESSION_HANDLE	X	X	X	X	X	X	X	X	X
CSA_E_INVALID_USER									
CSA_E_NO_AUTHORITY	X	X	X	X				X	X
CSA_E_NOT_SUPPORTED			X			X		X	
CSA_E_PASSWORD_REQUIRED									
CSA_E_READONLY	X								X
CSA_E_SERVICE_UNAVAILABLE	X	X	X	X	X	X	X	X	X
CSA_E_TEXT_TOO_LARGE	X								X
CSA_E_TOO_MANY_USERS									
CSA_E_UNABLE_TO_OPEN_FILE									
CSA_E_UNSUPPORTED_ATTRIBUTE	X			X			X	X	X
CSA_E_UNSUPPORTED_CHARACTER_SET									
CSA_E_UNSUPPORTED_DATA_EXT	X		X	X					X
CSA_E_UNSUPPORTED_ENUM		X		X					X
CSA_E_UNSUPPORTED_FLAG	X	X	X	X	X	X	X	X	X
CSA_E_UNSUPPORTED_FUNCTION_EXT	X	X	X	X	X	X	X	X	X
CSA_E_UNSUPPORTED_PARAMETER									X
CSA_E_UNSUPPORTED_VERSION									
CSA_E_USER_NOT_FOUND									

Table 4-5 XCS Entry Management Function Return Codes

The return codes are defined as follows:

CSA_E_AMBIGUOUS_USER
The calendar user's name is ambiguous; multiple matches have been found.

CSA_E_CALENDAR_EXISTS
The specified calendar already exists.

CSA_E_CALENDAR_NOT_EXIST
The specified calendar does not exist.

CSA_E_CALLBACK_NOT_REGISTERED
The specified callback function was not registered.

CSA_E_DISK_FULL
Insufficient disk space was available to complete the requested operation (this may refer to local or shared disk space).

CSA_E_FAILURE
There was a general failure which does not fit the description of any other error code.

CSA_E_FILE_EXIST
The specified file in csa_save already exists.

CSA_E_FILE_NOT_EXIST
The specified file in csa_restore does not exist.

CSA_E_INSUFFICIENT_MEMORY
Insufficient memory was available to complete the requested operation.

CSA_E_INVALID_ATTRIBUTE
An attribute was specified that was not defined by this specification and the implementation does not support the attribute as an application specific attribute.

CSA_E_INVALID_ATTRIBUTE_VALUE
An invalid attribute value was specified for an attribute.

CSA_E_INVALID_CALENDAR_SERVICE
The underlying calendar service is invalid, so logging on cannot be completed.

CSA_E_INVALID_CONFIGURATION
The specified logon configuration is inconsistent with the selected calendar service.

CSA_E_INVALID_DATA_EXT
The data extension requested is invalid.

CSA_E_INVALID_DATE_TIME
An invalid date and time combination was specified.

CSA_E_INVALID_ENTRY_HANDLE
An invalid calendar entry handle was specified.

CSA_E_INVALID_ENUM
A CSA_enum value is invalid.

CSA_E_INVALID_FILE_NAME
An invalid file name was specified.

CSA_E_INVALID_FLAG
A flag value in the flags parameter was invalid.

CSA_E_INVALID_FUNCTION
The function extension requested is invalid..

CSA_E_INVALID_MEMORY
Memory pointer passed is invalid.

CSA_E_INVALID_PARAMETER
A function parameter was invalid.

CSA_E_INVALID_PASSWORD
The password is incorrect.

CSA_E_INVALID_RULE
Implementation does not understand the exception or recurrence rule.

CSA_E_INVALID_SESSION_HANDLE
The specified Session Handle is invalid or no longer valid (for example, after logging off).

CSA_E_INVALID_USER
The specified calendar user is invalid.

CSA_E_NO_AUTHORITY
The user has insufficient authority for this function.

CSA_E_NOT_SUPPORTED
The operation requested is not supported by this implementation.

CSA_E_PASSWORD_REQUIRED
A password is required on this calendar service.

CSA_E_READONLY
An attempt was made to update a read-only attribute.

CSA_E_SERVICE_UNAVAILABLE
The requested calendar service is unavailable.

CSA_E_TEXT_TOO_LARGE
The size of the text string passed to the implementation is too large.

CSA_E_TOO_MANY_USERS
The implementation cannot support the additional logon of another calendar user at this time.

CSA_E_UNABLE_TO_OPEN_FILE
Unable to open the specified file.

CSA_E_UNSUPPORTED_ATTRIBUTE
An attribute was encountered that is unsupported by the calendar service.

CSA_E_UNSUPPORTED_CHARACTER_SET
The character set requested is not supported.

CSA_E_UNSUPPORTED_DATA_EXT
The data extension requested is not supported.

CSA_E_UNSUPPORTED_ENUM
The specified enumerated value is not valid.

CSA_E_UNSUPPORTED_FLAG
The flag requested is not supported.

CSA_E_UNSUPPORTED_FUNCTION_EXT
: The specified function extension is not supported or CSA_EXT_REQUIRED is set.

CSA_E_UNSUPPORTED_PARAMETER
: One of the parameters is not supported.

CSA_E_UNSUPPORTED_VERSION
: The specification version specified in the call cannot be supported by this XCS implementation.

CSA_E_USER_NOT_FOUND
: One or more of the specified calendar users were not found.

Chapter 5

C Declaration Summary

This section lists the declarations that define the XCS interface for the C Language binding. The declarations assembled here constitute the contents of a header file to be made accessible to application programmers by implementations conforming to this specification. The header file is to be named **<xcsa.h>**. The symbols the declarations define are the only symbols the calendar service makes visible to the application.

```
/*
 * xcsa.h
 */

/* BEGIN XCS INTERFACE */

/* BASIC DATA TYPES */

#ifndef DIFFERENT_PLATFORM
typedef short int            CSA_sint16;
typedef long int             CSA_sint32;
typedef unsigned char        CSA_uint8;
typedef unsigned short int   CSA_uint16;
typedef unsigned long int    CSA_uint32;
typedef void *               CSA_buffer;
typedef CSA_uint32           CSA_entry_handle;
typedef CSA_uint32           CSA_session_handle;
typedef char *               CSA_string;
#endif

typedef CSA_string     CSA_attribute_reference;
typedef CSA_uint32     CSA_boolean;
typedef CSA_string     CSA_date_time;
typedef CSA_string     CSA_date_time_range;
typedef CSA_sint32     CSA_enum;
typedef CSA_uint32     CSA_flags;
typedef CSA_uint32     CSA_return_code;
typedef CSA_string     CSA_service_reference;
typedef CSA_string     CSA_time_duration;

#define CSA_FALSE      ((CSA_boolean)0)
#define CSA_TRUE       ((CSA_boolean)1)

/* DATA STRUCTURES */

/* EXTENSION */
typedef struct CSA_TAG_EXTENSION{
    CSA_uint32    item_code;
    CSA_uint32    item_data;
    CSA_buffer    item_reference;
    CSA_flags     extension_flags;
} CSA_extension;
```

```
/* EXTENSION FLAGS */
#define    CSA_EXT_REQUIRED      ((CSA_flags)0x10000)
#define    CSA_EXT_OUTPUT        ((CSA_flags)0x20000)
#define    CSA_EXT_LAST_ELEMENT  ((CSA_flags)0x40000)

/* CALENDAR USER */
typedef struct CSA_TAG_CALENDAR_USER{
    CSA_string       user_name;
    CSA_enum         user_type;
    CSA_string       calendar_address;
    CSA_extension   *calendar_user_extensions;
} CSA_calendar_user;

/* CALENDAR USER TYPE */
#define CSA_USER_TYPE_INDIVIDUAL  ((CSA_enum)0)
#define CSA_USER_TYPE_GROUP       ((CSA_enum)1)
#define CSA_USER_TYPE_RESOURCE    ((CSA_enum)2)

/* CLASSIFICATION
#define CSA_CLASS_PUBLIC          ((CSA_enum)0)
#define CSA_CLASS_PRIVATE         ((CSA_enum)1)
#define CSA_CLASS_CONFIDENTIAL    ((CSA_enum)2)

/* ACCESS LIST */
typedef struct CSA_TAG_ACCESS_RIGHTS{
    CSA_calendar_user              *user;
    CSA_flags                       rights;
    struct CSA_TAG_ACCESS_RIGHTS   *next;
} CSA_access_rights, *CSA_access_list;

/* ACCESS RIGHTS FLAGS */
#define CSA_FREE_TIME_SEARCH             ((CSA_flags)0x1)
#define CSA_VIEW_PUBLIC_ENTRIES          ((CSA_flags)0x2)
#define CSA_VIEW_CONFIDENTIAL_ENTRIES    ((CSA_flags)0x4)
#define CSA_VIEW_PRIVATE_ENTRIES         ((CSA_flags)0x8)
#define CSA_INSERT_PUBLIC_ENTRIES        ((CSA_flags)0x10)
#define CSA_INSERT_CONFIDENTIAL_ENTRIES  ((CSA_flags)0x20)
#define CSA_INSERT_PRIVATE_ENTRIES       ((CSA_flags)0x40)
#define CSA_CHANGE_PUBLIC_ENTRIES        ((CSA_flags)0x80)
#define CSA_CHANGE_CONFIDENTIAL_ENTRIES  ((CSA_flags)0x100)
#define CSA_CHANGE_PRIVATE_ENTRIES       ((CSA_flags)0x200)
#define CSA_VIEW_CALENDAR_ATTRIBUTES     ((CSA_flags)0x400)
#define CSA_INSERT_CALENDAR_ATTRIBUTES   ((CSA_flags)0x800)
#define CSA_CHANGE_CALENDAR_ATTRIBUTES   ((CSA_flags)0x1000)
#define CSA_ORGANIZER_RIGHTS             ((CSA_flags)0x2000)
#define CSA_SPONSOR_RIGHTS               ((CSA_flags)0x4000)
#define CSA_OWNER_RIGHTS                 ((CSA_flags)0x8000)

/* ATTENDEE LIST */
typedef struct CSA_TAG_ATTENDEE{
    CSA_calendar_user          attendee;
    CSA_enum                   priority;
```

```
    CSA_enum                  status;
    CSA_boolean               rsvp_requested;
    struct CSA_TAG_ATTENDEE   *next;
} CSA_attendee, *CSA_attendee_list;

/* ATTENDEE PRIORITIES */
#define CSA_FOR_YOUR_INFORMATION   ((CSA_enum)0)
#define CSA_ATTENDANCE_REQUESTED   ((CSA_enum)1)
#define CSA_ATTENDANCE_REQUIRED    ((CSA_enum)2)
#define CSA_IMMEDIATE_RESPONSE     ((CSA_enum)3)

/* ATTENDEE STATUS */
#define CSA_STATUS_ACCEPTED       ((CSA_enum)0)
#define CSA_STATUS_NEEDS_ACTION   ((CSA_enum)1)
#define CSA_STATUS_SENT           ((CSA_enum)2)
#define CSA_STATUS_TENTATIVE      ((CSA_enum)3)
#define CSA_STATUS_CONFIRMED      ((CSA_enum)4)
#define CSA_STATUS_REJECTED       ((CSA_enum)5)
#define CSA_STATUS_COMPLETED      ((CSA_enum)6)
#define CSA_STATUS_DELEGATED      ((CSA_enum)7)

/* OPAQUE DATA */
typedef struct CSA_TAG_OPAQUE_DATA{
    CSA_uint32          size;
    CSA_uint8          *data;
} CSA_opaque_data;

/* CALLBACK */
typedef void (*CSA_callback)(
    CSA_session_handle  session,
    CSA_flags           reason,
    CSA_buffer          call_data,
    CSA_buffer          client_data,
    CSA_extension      *callback_extensions
);

/* CALLBACK DATA */
typedef struct CSA_TAG_LOGON_CB_DATA {
    CSA_calendar_user   *user;
} CSA_logon_callback_data;

/* CALENDAR DELETED CALLBACK DATA */
typedef struct CSA_TAG_CALENDAR_DELETED_CB_DATA {
    CSA_calendar_user   *user;
} CSA_calendar_deleted_callback_data;

/* CALENDAR ATTRIBUTE UPDATED CALLBACK DATA */
typedef struct CSA_TAG_CALENDAR_ATTR_UPDATE_CB_DATA {
    CSA_calendar_user         *user;
    CSA_uint32                 number_attributes;
    CSA_attribute_reference   *attribute_names;
} CSA_calendar_attr_update_callback_data;
```

```
/* ADD ENTRY CALLBACK DATA */
typedef struct CSA_TAG_ADD_ENTRY_CB_DATA {
    CSA_calendar_user      *user;
    CSA_opaque_data         added_entry_id;
} CSA_add_entry_callback_data;

/* DELETE ENTRY CALLBACK DATA */
typedef struct CSA_TAG_DELETE_ENTRY_CB_DATA {
    CSA_calendar_user    *user;
    CSA_opaque_data       deleted_entry_id;
    CSA_enum              scope;
    CSA_date_time         date_and_time;
} CSA_delete_entry_callback_data;

/* UPDATE ENTRY CALLBACK DATA */
typedef struct CSA_TAG_UPDATE_ENTRY_CB_DATA {
    CSA_calendar_user    *user;
    CSA_opaque_data       old_entry_id;
    CSA_opaque_data       new_entry_id;
    CSA_enum              scope;
    CSA_date_time         date_and_time;
} CSA_update_entry_callback_data;

/* DATE AND TIME LIST */
typedef struct CSA_TAG_DATE_TIME_ITEM{
 CSA_date_time                    date_time;
 struct CSA_TAG_DATE_TIME_ITEM    *next;
} CSA_date_time_entry, *CSA_date_time_list;

/* FREE TIME */
typedef struct CSA_TAG_FREE_TIME{
    CSA_uint32              number_free_time_data;
    CSA_date_time_range    *free_time_data;
} CSA_free_time;

/* REMINDER */
typedef struct CSA_TAG_REMINDER{
    CSA_time_duration    lead_time;
    CSA_time_duration    snooze_time;
    CSA_uint32           repeat_count;
    CSA_opaque_data      reminder_data;
} CSA_reminder;

/* REMINDER REFERENCE */
typedef struct CSA_TAG_REMINDER_REFERENCE{
    CSA_entry_handle           entry;
    CSA_date_time              run_time;
    CSA_time_duration          snooze_time;
    CSA_uint32                 repeat_count;
    CSA_attribute_reference    attribute_name;
} CSA_reminder_reference;
```

```
/* ATTRIBUTE */
typedef struct CSA_TAG_ATTRIBUTE_ITEM{
        CSA_enum    type;
        union {
            CSA_boolean            boolean_value;
            CSA_enum               enumerated_value;
            CSA_flags              flags_value;
            CSA_sint32             sint32_value;
            CSA_uint32             uint32_value;
            CSA_calendar_user      calendar_user_value;
            CSA_date_time          date_time_value;
            CSA_date_time_range    date_time_range_value;
            CSA_time_duration      time_duration_value;
            CSA_string             string_value;
            CSA_attendee_list      attendee_list_value;
            CSA_date_time_list     date_time_list_value;
            CSA_access_list        access_list_value;
            CSA_reminder          *reminder_value;
            CSA_opaque_data       *opaque_data_value;
        } item;
    } CSA_attribute_value;

typedef struct CSA_TAG_ATTRIBUTE{
    CSA_string              name;
    CSA_attribute_value    *value;
    CSA_extension          *attribute_extensions;
} CSA_attribute;

/* ATTRIBUTE VALUE TYPE */
#define CSA_VALUE_BOOLEAN            ((CSA_enum) 0)
#define CSA_VALUE_ENUMERATED         ((CSA_enum) 1)
#define CSA_VALUE_FLAGS              ((CSA_enum) 2)
#define CSA_VALUE_SINT32             ((CSA_enum) 3)
#define CSA_VALUE_UINT32             ((CSA_enum) 4)
#define CSA_VALUE_STRING             ((CSA_enum) 5)
#define CSA_VALUE_CALENDAR_USER      ((CSA_enum) 6)
#define CSA_VALUE_DATE_TIME          ((CSA_enum) 7)
#define CSA_VALUE_DATE_TIME_RANGE    ((CSA_enum) 8)
#define CSA_VALUE_TIME_DURATION      ((CSA_enum) 9)
#define CSA_VALUE_ACCESS_LIST        ((CSA_enum) 10)
#define CSA_VALUE_ATTENDEE_LIST      ((CSA_enum) 11)
#define CSA_VALUE_DATE_TIME_LIST     ((CSA_enum) 12)
#define CSA_VALUE_REMINDER           ((CSA_enum) 13)
#define CSA_VALUE_OPAQUE_DATA        ((CSA_enum) 14)

/* WORK SCHEDULE */
typedef struct CSA_TAG_WORK_SCHEDULE{
    CSA_date_time         schedule_begin_time;
    CSA_boolean           cyclic_definition_flag;
    CSA_date_time         cycle_end_time;
    CSA_date_time_list   *work_cycle;
} CSA_work_schedule;
```

```
/* XCS FUNCTIONS */

/* CROSS FUNCTION FLAGS */

/* SCOPE */
#define CSA_SCOPE_ALL       ((CSA_enum)0)
#define CSA_SCOPE_ONE       ((CSA_enum)1)
#define CSA_SCOPE_FORWARD   ((CSA_enum)2)

/* OPERATORS */
#define CSA_MATCH_ANY                       ((CSA_enum)0)
#define CSA_MATCH_EQUAL_TO                  ((CSA_enum)1)
#define CSA_MATCH_NOT_EQUAL_TO              ((CSA_enum)2)
#define CSA_MATCH_GREATER_THAN              ((CSA_enum)3)
#define CSA_MATCH_LESS_THAN                 ((CSA_enum)4)
#define CSA_MATCH_GREATER_THAN_OR_EQUAL_TO  ((CSA_enum)5)
#define CSA_MATCH_LESS_THAN_OR_EQUAL_TO     ((CSA_enum)6)
#define CSA_MATCH_CONTAIN                   ((CSA_enum)7)

/* FREE */
CSA_return_code
csa_free(
    CSA_buffer    memory
);

/* LIST CALENDARS */
CSA_return_code
csa_list_calendars(
    CSA_service_reference    calendar_service,
    CSA_uint32              *number_names,
    CSA_calendar_user      **calendar_names,
    CSA_extension           *list_calendars_extensions
);

/* LOGOFF */
CSA_return_code
csa_logoff(
    CSA_session_handle       session,
    CSA_extension           *logoff_extensions
);

/* LOGON */
csa_logon(
    CSA_service_reference calendar_service,
    CSA_calendar_user       *user,
    CSA_string               password,
    CSA_string               character_set,
    CSA_string               required_csa_version,
    CSA_session_handle      *session,
    CSA_extension           *logon_extensions
);
```

```
/* LOOK UP */
CSA_return_code
csa_look_up(
    CSA_session_handle       session,
    CSA_calendar_user       *users,
    CSA_flags                look_up_flags,
    CSA_uint32              *number_users,
    CSA_calendar_user      **user_list,
    CSA_extension           *look_up_extensions
);

#define CSA_LOOKUP_RESOLVE_PREFIX_SEARCH   ((CSA_flags)0x1)
#define CSA_LOOKUP_RESOLVE_IDENTITY        ((CSA_flags)0x2)

/* QUERY CONFIGURATION */
CSA_return_code
csa_query_configuration(
    CSA_session_handle       session,
    CSA_enum                 item,
    CSA_buffer              *reference,
    CSA_extension           *query_configuration_extensions
);

#define CSA_CONFIG_CHARACTER_SET     ((CSA_enum)0)
#define CSA_CONFIG_LINE_TERM         ((CSA_enum)1)
#define CSA_CONFIG_DEFAULT_SERVICE   ((CSA_enum)2)
#define CSA_CONFIG_DEFAULT_USER      ((CSA_enum)3)
#define CSA_CONFIG_REQ_PASSWORD      ((CSA_enum)4)
#define CSA_CONFIG_REQ_SERVICE       ((CSA_enum)5)
#define CSA_CONFIG_REQ_USER          ((CSA_enum)6)
#define CSA_CONFIG_UI_AVALI          ((CSA_enum)7)
#define CSA_CONFIG_VER_IMPLEM        ((CSA_enum)8)
#define CSA_CONFIG_VER_SPEC          ((CSA_enum)9)

/* CHARACTER SET IDENTIFIERS */
#define CSA_CHARSET_437             "//XAPIA/CHARSET IBM 437//"
#define CSA_CHARSET_850             "//XAPIA/CHARSET IBM 850//"
#define CSA_CHARSET_1252     "//XAPIA/CHARSET Microsoft 1252//"
#define CSA_CHARSET_ISTRING   "//XAPIA/CHARSET Apple ISTRING//"
#define CSA_CHARSET_UNICODE         "//XAPIA/CHARSET UNICODE//"
#define CSA_CHARSET_T61             "//XAPIA/CHARSET TSS T61//"
#define CSA_CHARSET_IA5             "//XAPIA/CHARSET TSS IA5//"
#define CSA_CHARSET_ISO_10646     "//XAPIA/CHARSET ISO 10646//"
#define CSA_CHARSET_ISO_646         "//XAPIA/CHARSET ISO 646//"
#define CSA_CHARSET_ISO_8859_1   "//XAPIA/CHARSET ISO 8859-1//"

/* RESTORE */
CSA_return_code
csa_restore(
    CSA_session_handle     session,
    CSA_string             archive_name,
    CSA_uint32             number_attributes,
```

```
    CSA_attribute         *attributes,
    CSA_enum              *operators,
    CSA_extension         *restore_extensions
);

/* SAVE */
CSA_return_code
csa_save(
    CSA_session_handle     session,
    CSA_string             archive_name,
    CSA_uint32             number_attributes,
    CSA_attribute         *attributes,
    CSA_enum              *operators,
    CSA_boolean            delete_entry,
    CSA_extension         *save_extensions
)

/* ADD CALENDAR */
CSA_return_code
csa_add_calendar(
    CSA_session_handle     session,
    CSA_calendar_user      user,
    CSA_uint32             number_attributes,
    CSA_attribute         *calendar_attributes,
    CSA_extension         *delete_calendar_extensions
);

/* DELETE CALENDAR */
CSA_return_code
csa_delete_calendar(
    CSA_session_handle     session,
    CSA_extension         *delete_calendar_extensions
);

/* LIST CALENDAR ATTRIBUTES
CSA_return_code
csa_list_calendar_attributes(
    CSA_session_handle         session,
    CSA_uint32                *number_names,
    CSA_attribute_reference  **calendar_attributes_names,
    CSA_extension              *list_calendar_attributes_extensions
);

/* READ CALENDAR ATTRIBUTES */
CSA_return_code
csa_read_calendar_attributes(
    CSA_session_handle         session,
    CSA_uint32                 number_names,
```

```
    CSA_attribute_reference   *attribute_names,
    CSA_uint32                *number_attributes,
    CSA_attribute            **calendar_attributes,
    CSA_extension             *read_calendar_attributes_extensions
);

/* REGISTER CALLBACK FUNCTION */
CSA_return_code
csa_register_callback(
    CSA_session_handle         session,
    CSA_flags                  reason,
    CSA_callback               callback,
    CSA_buffer                 client_data,
    CSA_extension             *register_callback_extensions
);

#define CSA_CB_CALENDAR_LOGON               ((CSA_flags)0x0
#define CSA_CB_CALENDAR_DELETED             ((CSA_flags)0x1)
#define CSA_CB_CALENDAR_ATTRIBUTE_UPDATED   ((CSA_flags)0x2)
#define CSA_CB_ENTRY_ADDED                  ((CSA_flags)0x4)
#define CSA_CB_ENTRY_DELETED                ((CSA_flags)0x8)
#define CSA_CB_ENTRY_UPDATED                ((CSA_flags)0x10)

/* UNREGISTER CALLBACK
CSA_return_code
csa_unregister_callback(
    CSA_session_handle     session,
    CSA_flags              reason,
    CSA_callback           callback,
    CSA_buffer             client_data,
    CSA_extension         *unregister_callback_extensions
);

/* CALL CALLBACKS
CSA_return_code
csa_call_callbacks(
    CSA_session_handle     session,
    CSA_flags              reason,
    CSA_extension         *call_callbacks_extensions
);

/* UDPATE CALENDAR ATTRIBUTES */
CSA_return_code
csa_update_calendar_attributes(
    CSA_session_handle     session,
    CSA_uint32             number_attributes,
    CSA_attribute         *calendar_attributes,
    CSA_extension         *update_calendar_attributes_extensions
);

/* Entry Types */
#define CSA_TYPE_EVENT    ((CSA_enum)0)
```

```
#define CSA_TYPE_TODO     ((CSA_enum)1)
#define CSA_TYPE_MEMO     ((CSA_enum)2)

/* ENTRY SUBTYPES */
/* These are included in the header as example subtypes and to      */
/* demonstration how the implementation should declare these values */
#define CSA_SUBTYPE_APPOINTMENT \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Appointment//EN"
#define CSA_SUBTYPE_CLASS \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Class//EN"
#define CSA_SUBTYPE_HOLIDAY \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Holiday//EN"
#define CSA_SUBTYPE_MEETING \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Meeting//EN"
#define CSA_SUBTYPE_MISCELLANEOUS \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Miscellaneous//EN"
#define CSA_SUBTYPE_PHONE_CALL \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Phone Call//EN"
#define CSA_SUBTYPE_SICK_DAY \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Sick Day//EN"
#define CSA_SUBTYPE_SPECIAL_OCCASION \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Special Occasion//EN"
#define CSA_SUBTYPE_TRAVEL \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Travel//EN"
#define CSA_SUBTYPE_VACATION \
    "-//XAPIA/CSA/SUBTYPE//NONSGML Subtype Vacation//EN"
*/

/* ADD ENTRY */
CSA_return_code
csa_add_entry(
    CSA_session_handle    session,
    CSA_uint32            number_attributes,
    CSA_attribute        *entry_attributes,
    CSA_entry_handle     *entry,
    CSA_extension        *add_entry_extensions
);

/* DELETE ENTRY */
CSA_return_code
csa_delete_entry(
    CSA_session_handle    session,
    CSA_entry_handle      entry,
    CSA_enum              delete_scope,
    CSA_extension        *delete_entry_extensions
);

/* FREE TIME SEARCH */
CSA_return_code
csa_free_time_search(
    CSA_session_handle     session,
    CSA_date_time_range    date_time_range,
```

```
    CSA_time_duration      time_duration,
    CSA_uint32             number_users,
    CSA_calendar_user     *calendar_users,
    CSA_free_time        **free_time,
    CSA_extension         *free_time_search_extensions
);

/* LIST ENTRIES */
CSA_return_code
csa_list_entries(
    CSA_session_handle     session,
    CSA_uint32             number_attributes,
    CSA_attribute         *entry_attributes,
    CSA_enum              *list_operators,
    CSA_uint32            *number_entries,
    CSA_entry_handle     **entries,
    CSA_extension         *list_entries_extensions
);

/* LIST ENTRY ATTRIBUTES */
CSA_return_code
csa_list_entry_attributes(
    CSA_session_handle          session,
    CSA_entry_handle            entry,
    CSA_uint32                 *number_names,
    CSA_attribute_reference   **entry_attribute_names,
    CSA_extension              *list_entry_attributes_extensions
);

/* LIST ENTRY SEQUENCE */
CSA_return_code
csa_list_entry_sequence(
    CSA_session_handle      session,
    CSA_entry_handle        entry,
    CSA_date_time_range     time_range,
    CSA_uint32             *number_entries,
    CSA_entry_handle      **entry_list,
    CSA_extension          *list_entry_sequence_extensions
);

/* READ ENTRY ATTRIBUTES */
CSA_return_code
csa_read_entry_attributes(
    CSA_session_handle           session,
    CSA_entry_handle             entry,
    CSA_uint32                   number_names,
    CSA_attribute_reference     *attribute_names,
    CSA_uint32                  *number_attributes,
    CSA_attribute              **entry_attributes,
    CSA_extension               *read_entry_attributes_extensions
);
```

```
/* READ NEXT REMINDERS */
CSA_return_code
csa_read_next_reminder(
    CSA_session_handle         session,
    CSA_uint32                 number_names,
    CSA_attribute_reference   *reminder_names,
    CSA_date_time              given_time,
    CSA_uint32                *number_reminders,
    CSA_reminder_reference   **reminder_references,
    CSA_extension              *read_next_reminder_extensions
);

/* UPDATE ENTRY ATTRIBUTES */
CSA_return_code
csa_update_entry_attributes(
    CSA_session_handle    session,
    CSA_entry_handle      entry,
    CSA_enum              update_scope,
    CSA_boolean           update_propagation,
    CSA_uint32            number_attributes,
    CSA_attribute        *entry_attributes,
    CSA_entry_handle     *new_entry,
    CSA_extension        *update_entry_attributes_extensions
);

/* STRING BASED FUNCTIONS */

/* ADD EVENT */
CSA_return_code
csa_add_event(
    CSA_service_reference   calendar_service,
    CSA_string              calendar_address,
    CSA_string              logon_user,
    CSA_string              logon_password,
    CSA_string              attendee,
    CSA_enum                attendee_priority,
    CSA_enum                attendee_status,
    CSA_boolean             attendee_rsvp_requested,
    CSA_date_time           start_date,
    CSA_date_time           end_date,
    CSA_string              organizer,
    CSA_string              sponsor,
    CSA_string              summary,
    CSA_string              description,
    CSA_string              recurrence_rule,
    CSA_string              exception_rule,
    CSA_string              subtype,
    CSA_enum                classification,
    CSA_string              delimiters,
    CSA_string              add_event_extensions
);
```

```
/* ADD TODO */
CSA_return_code
csa_add_todo(
    CSA_service_reference   calendar_service,
    CSA_string              calendar_address,
    CSA_string              logon_user,
    CSA_string              logon_password,
    CSA_enum                attendee_priority,
    CSA_enum                attendee_status,
    CSA_boolean             attendee_rsvp_requested,
    CSA_date_time           start_date,
    CSA_date_time           due_date,
    CSA_uint32              priority,
    CSA_string              summary,
    CSA_string              description,
    CSA_enum                classification,
    CSA_string              delimiters,
    CSA_string              add_todo_extensions
);

/* ADD MEMO */
CSA_return_code
csa_add_memo(
    CSA_service_reference   calendar_service,
    CSA_string              calendar_address,
    CSA_string              logon_user,
    CSA_string              logon_password,
    CSA_date_time           start_date,
    CSA_string              summary,
    CSA_string              delimiters,
    CSA_string              add_memo_extensions
);

/* RETURN ERROR FLAGS */
#define CSA_ERROR_RSV_MASK    ((CSA_return_code)0x0000FFFF)
#define CSA_ERROR_IMPL_MASK   ((CSA_return_code)0xFFFF0000)

/* RETURN CODES */
#define CSA_SUCCESS                      ((CSA_return_code)0)
#define CSA_E_AMBIGUOUS_USER             ((CSA_return_code)1)
#define CSA_E_CALENDAR_EXISTS            ((CSA_return_code)2)
#define CSA_E_CALENDAR_NOT_EXIST         ((CSA_return_code)3)
#define CSA_E_CALLBACK_NOT_REGISTERED    ((CSA_return_code)4)
#define CSA_E_DISK_FULL                  ((CSA_return_code)5)
#define CSA_E_FAILURE                    ((CSA_return_code)6)
#define CSA_E_FILE_EXIST                 ((CSA_return_code)7)
#define CSA_E_FILE_NOT_EXIST             ((CSA_return_code)8)
#define CSA_E_INSUFFICIENT_MEMORY        ((CSA_return_code)9)
#define CSA_E_INVALID_ATTRIBUTE          ((CSA_return_code)10)
#define CSA_E_INVALID_ATTRIBUTE_VALUE    ((CSA_return_code)11)
#define CSA_E_INVALID_CALENDAR_SERVICE   ((CSA_return_code)12)
#define CSA_E_INVALID_CONFIGURATION      ((CSA_return_code)13)
```

```
#define CSA_E_INVALID_DATA_EXT          ((CSA_return_code)14)
#define CSA_E_INVALID_DATE_TIME         ((CSA_return_code)15)
#define CSA_E_INVALID_ENTRY_HANDLE      ((CSA_return_code)16)
#define CSA_E_INVALID_ENUM              ((CSA_return_code)17)
#define CSA_E_INVALID_FILE_NAME         ((CSA_return_code)18)
#define CSA_E_INVALID_FLAG              ((CSA_return_code)19)
#define CSA_E_INVALID_FUNCTION_EXT      ((CSA_return_code)20)
#define CSA_E_INVALID_MEMORY            ((CSA_return_code)21)
#define CSA_E_INVALID_PARAMETER         ((CSA_return_code)22)
#define CSA_E_INVALID_PASSWORD          ((CSA_return_code)23)
#define CSA_E_INVALID_RETURN_CODE       ((CSA_return_code)24)
#define CSA_E_INVALID_SESSION_HANDLE    ((CSA_return_code)25)
#define CSA_E_INVALID_USER              ((CSA_return_code)26)
#define CSA_E_LOCALE_MISMATCH           ((CSA_return_code)27)
#define CSA_E_LOGON_FAILURE             ((CSA_return_code)28)
#define CSA_E_NO_AUTHORITY              ((CSA_return_code)29)
#define CSA_E_NOT_SUPPORTED             ((CSA_return_code)30)
#define CSA_E_PASSWORD_REQUIRED         ((CSA_return_code)31)
#define CSA_E_READONLY                  ((CSA_return_code)32)
#define CSA_E_SERVICE_UNAVAILABLE       ((CSA_return_code)33)
#define CSA_E_TEXT_TOO_LARGE            ((CSA_return_code)34)
#define CSA_E_TOO_MANY_USERS            ((CSA_return_code)35)
#define CSA_E_UNABLE_TO_OPEN_FILE       ((CSA_return_code)36)
#define CSA_E_UNSUPPORTED_ATTRIBUTE     ((CSA_return_code)37)
#define CSA_E_UNSUPPORTED_CHARACTER_SET ((CSA_return_code)38)
#define CSA_E_UNSUPPORTED_DATA_EXT      ((CSA_return_code)39)
#define CSA_E_UNSUPPORTED_FLAG          ((CSA_return_code)40)
#define CSA_E_UNSUPPORTED_FUNCTION_EXT  ((CSA_return_code)41)
#define CSA_E_UNSUPPORTED_PARAMETER     ((CSA_return_code)42)
#define CSA_E_UNSUPPORTED_VERSION       ((CSA_return_code)43)
#define CSA_E_USER_NOT_FOUND            ((CSA_return_code)44)

/* CALENDAR ATTRIBUTES */

/* CALENDAR ATTRIBUTE NAMES */
#define CSA_CAL_ATTR_ACCESS_LIST                        \
    "-//XAPIA/CSA/CALATTR//NONSGML Access List//EN"
#define CSA_CAL_ATTR_CALENDAR_NAME                      \
    "-//XAPIA/CSA/CALATTR//NONSGML Calendar Name//EN"
#define CSA_CAL_ATTR_CALENDAR_OWNER                     \
    "-//XAPIA/CSA/CALATTR//NONSGML Calendar Owner//EN"
#define CSA_CAL_ATTR_CALENDAR_SIZE                      \
    "-//XAPIA/CSA/CALATTR//NONSGML Calendar Size//EN"
#define CSA_CAL_ATTR_CODE_PAGE                          \
    "-//XAPIA/CSA/CALATTR//NONSGML Character Set//EN"
#define CSA_CAL_ATTR_COUNTRY                            \
    "-//XAPIA/CSA/CALATTR//NONSGML Country//EN"
#define CSA_CAL_ATTR_DATE_CREATED                       \
    "-//XAPIA/CSA/CALATTR//NONSGML Date Created//EN"
#define CSA_CAL_ATTR_LANGUAGE                           \
    "-//XAPIA/CSA/CALATTR//NONSGML Language//EN"
#define CSA_CAL_ATTR_NUMBER_ENTRIES                     \
```

```
    "-//XAPIA/CSA/CALATTR//NONSGML Number Entries//EN"
#define CSA_CAL_ATTR_PRODUCT_IDENTIFIER                    \
    "-//XAPIA/CSA/CALATTR//NONSGML Product Identifier//EN"
#define CSA_CAL_ATTR_TIME_ZONE                             \
    "-//XAPIA/CSA/CALATTR//NONSGML Time Zone//EN"
#define CSA_CAL_ATTR_VERSION                               \
    "-//XAPIA/CSA/CALATTR//NONSGML Version//EN"
#define CSA_CAL_ATTR_WORK_SCHEDULE                         \
    "-//XAPIA/CSA/CALATTR//NONSGML Work Schedule//EN"

/* ENTRY ATTRIBUTES */

/* ENTRY ATTRIBUTES NAMES */
#define CSA_ENTRY_ATTR_ATTENDEE_LIST                         \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Attendee List//EN"
#define CSA_ENTRY_ATTR_AUDIO_REMINDER                        \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Audio Reminder//EN"
#define CSA_ENTRY_ATTR_CLASSIFICATION                        \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Classification//EN"
#define CSA_ENTRY_ATTR_DATE_COMPLETED                        \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Date Completed//EN"
#define CSA_ENTRY_ATTR_DATE_CREATED                          \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Date Created//EN"
#define CSA_ENTRY_ATTR_DESCRIPTION                           \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Description//EN"
#define CSA_ENTRY_ATTR_DUE_DATE                              \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Due Date//EN"
#define CSA_ENTRY_ATTR_END_DATE                              \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML End Date//EN"
#define CSA_ENTRY_ATTR_EXCEPTION_DATES                       \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Exception Dates//EN"
#define CSA_ENTRY_ATTR_EXCEPTION_RULE                        \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Exception Rule//EN"
#define CSA_ENTRY_ATTR_FLASHING_REMINDER                     \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Flashing Reminder//EN"
#define CSA_ENTRY_ATTR_LAST_UPDATE                           \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Last Update//EN"
#define CSA_ENTRY_ATTR_MAIL_REMINDER                         \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Mail Reminder//EN"
#define CSA_ENTRY_ATTR_NUMBER_RECURRENCES                    \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Number Recurrences//EN"
#define CSA_ENTRY_ATTR_ORGANIZER                             \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Organizer//EN"
#define CSA_ENTRY_ATTR_POPUP_REMINDER                        \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Popup Reminder//EN"
#define CSA_ENTRY_ATTR_PRIORITY                              \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Priority//EN"
#define CSA_ENTRY_ATTR_RECURRENCE_RULE                       \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Recurrence Rule//EN"
#define CSA_ENTRY_ATTR_RECURRING_DATES                      \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Recurring Dates//EN"
#define CSA_ENTRY_ATTR_REFERENCE_IDENTIFIER                  \
```

```
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Reference_identifier//EN"
#define CSA_ENTRY_ATTR_SEQUENCE_NUMBER                          \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Sequence Number//EN"
#define CSA_ENTRY_ATTR_SPONSOR                                  \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Sponsor//EN"
#define CSA_ENTRY_ATTR_START_DATE                               \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Start Date//EN"
#define CSA_ENTRY_ATTR_STATUS                                   \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Status//EN"
#define CSA_ENTRY_ATTR_SUBTYPE                                  \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Subtype//EN"
#define CSA_ENTRY_ATTR_SUMMARY                                  \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Summary//EN"
#define CSA_ENTRY_ATTR_TIME_TRANSPARENCY                        \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Transparency//EN"
#define CSA_ENTRY_ATTR_TYPE                                     \
    "-//XAPIA/CSA/ENTRYATTR//NONSGML Type//EN"

/* COMMON EXTENSIONS DECLARATIONS */

/* EXTENSION SET ID */

/* Common Extension Set */
#define CSA_XS_COM     ((CSA_uint32) 0)
/* Bilateral Extension Set */
#define CSA_XS_BLT     ((CSA_uint32) 256)

/* FUNCTION EXTENSIONS */

/* Query for extension support in implementation */
#define CSA_X_COM_SUPPORT_EXT    ((CSA_uint32) 1)

typedef struct CSA_TAG_XCOM{
    CSA_uint32     item_code;
    CSA_flags      flags;
} CSA_X_COM_support;

#define CSA_X_COM_SUPPORTED            ((CSA_flags) 1)
#define CSA_X_COM_NOT_SUPPORTED        ((CSA_flags) 2)
#define CSA_X_COM_DATA_EXT_SUPPORTED   ((CSA_flags) 4)
#define CSA_X_COM_FUNC_EXT_SUPPORTED   ((CSA_flags) 8)
#define CSA_X_COM_SUP_EXCLUDE          ((CSA_flags) 16)

#define CSA_X_UI_ID_EXT  ((CSA_uint32)2)
/* EXTENSION FLAGS */
#define CSA_X_LOGON_UI_ALLOWED         ((CSA_flags)0x1)
#define CSA_X_ERROR_UI_ALLOWED         ((CSA_flags)0x2)
#define CSA_X_LOOKUP_RESOLVE_UI        ((CSA_flags)0x4)
#define CSA_X_LOOKUP_DETAILS_UI        ((CSA_flags)0x8)
#define CSA_X_LOOKUP_ADDRESSING_UI     ((CSA_flags)0x10)
#define CSA_X_ADD_DEFINE_ENTRY_UI      ((CSA_flags)0x20)
```

```
/* EXTENSION RETURN CODES */
#define CSA_X_E_INVALID_UI_ID    ((CSA_return_code)1025)
#define CSA_X_E_LOGON_FAILURE    ((CSA_return_code)1026)
#define CSA_X_E_USER_CANCEL      ((CSA_return_code)1027)

#define CSA_X_XT_APP_CONTEXT_EXT       ((CSA_uint32)3)
```

Chapter 6

Extensions

6.1 Extension Registration

A set of common extensions are defined by this specification. Vendor specified extensions may be defined by any implementor of the CS API. Further extension sets may also be defined by future versions of this specification. Because of this, it is important to have a set of guidelines for the naming and definition of extensions. These guidelines are given below:

1. **item_code** ranges will be handed out to vendors or vendor groups in blocks of 256 for creating extension sets. A vendor/vendor group may get more than one item_code range if necessary for the extension set. The extension set identifier for all the sets item_code ranges will be the first location of the first block given out. This extension set identifier is used to query the service for support of a particular extension set.

   For example, the extension blocks for Vendor Group X may be 0x00000400, 0x00000900, and 0x00004300 and the extension set identifier would be 0x00000400 if that was the first block assigned to the vendor. Applications would ask a service if it supports extension set 0x00000400, for this vendor group's extensions.

2. An extension set will also have a specific prefix assigned to it for use in the names of all extensions in the extension set. The format of the prefix will be:

   - CSA_XS_[vendor id]
     for the extension set identifier
   - CSA_X_[vendor id]_[extension name]
     for the item codes of extensions in the set .

   In the example with Vendor Group X above, if its vendor id was CX, it would define its extensions as:

   ```
   #define      CSA_XS_CX       0x00000400
   #define      CSA_X_CX_EXT1   0x00000401
   #define      CSA_X_CX_EXT2   0x00000402
    .....
   ```

3. Extension sets defined by this specification will be allocated an extension set number and prefix from the X.400 API Association. Implementors may also obtain an extension set prefix, and a block of extension codes, from the X API Association by requesting such a number in writing. Pre-defined extension set numbers are defined in the C Language Declaration, in Chapter 5. Support for different extension sets is indicated through the configuration of the XCS implementation and can be queried through the function *csa_query_configuration*( ), using the CSA_X_COM_SUPPORT_EXT extension.

4. An extension set value of BILATERAL has also been allocated. Extensions may be defined within the BILATERAL set by any implementor. No registration of a extension set number is required. This set is provided so that implementors may define extensions without any formal registration. Because of this freedom, extensions from different vendors may conflict and inhibit application portability and the co-residency of different XCS implementations. The prefix for these extensions will be **CSA_X_BLT_** and the corresponding set identifier is **CSA_XS_BLT**.

5. To minimise portability issues, implementors are encouraged to specify extensions as generically as possible, and to contribute these extensions as proposed additions to the XCS-defined extension set. Through this process, the CS API set will evolve in a positive direction in a manner which continues to maximise portability.

6.2 Common Extension Set

The XCS interface common extension set contains those function and data extensions that are common to most calendaring and scheduling services, but are not in the base specification for various reasons. After the documentation for all of the extensions, a C declaration section is provided as the basis of a header file for this extension set. This section should be used as a model for creation of other extension sets. Explanations of extensions and extensions structures are provided in Chapter 2 and Chapter 3.

CSA_XS_COM

This extension identifier is used to represent all the extensions in the common extension set. This identifier should be used with the CSA_X_COM_SUPPORT_EXT extension (described below) on *csa_query_configuration*( ) and *csa_logon*( ) to determine support for the common extension set. By asking the implementation if it supports the entire common extension set, the application does not need to individually request all the extensions it might be interested in. If used during *csa_logon*( ) it will also indicate data extensions that should be attached to the structures for this session (as described below). The implementation should return the support level based on the description in the CSA_X_COM_SUPPORT_EXT extension.

Man-page definitions for the Function Extensions follow.

NAME

CSA_X_COM_SUPPORT_EXT — query the XCS implementation about which extensions it supports.

DESCRIPTION

This extension is used by client applications to query the XCS implementation about which extensions it supports. This can be used before a session is established to get preliminary information about support before logging on. When this extension is used with *csa_logon*( ) this extension will also indicate which data extensions the client wants added to the data structures for the session Note that some implementations may support different extensions based on what service the client application creates a session with, so using this extension at logon time is recommended to verify extension support. If any extensions are supported by a XCS implementation, this extension must be supported.

USED BY

csa_query_config( ), *csa_logon*( ).

INPUT

extension_flags

All XCS flags are valid. No further flags are defined.

item_data

count of items in array pointed to by item_reference.

item_reference

pointer to first element in array of structures listing extensions the application requests be supported by the implementation. The C declaration for this structure is:

```
typedef struct {
    CSA_uint32    item_code;
    CSA_flags     flags;
} CSA_X_COM_support;
```

The item_code in the structure is set to the item code of the extension the application is querying the service about. These can be either extension sets or individual extensions. An item code of null will be ignored.

The flags for the structures that are used on input are:

CSA_X_COM_SUP_EXCLUDE

exclude this item when deciding whether the implementation supports an extension set. On logon, do not attach this item to structures for this session even if other entries request that it be attached. This flag is used only with extension sets.

OUTPUT

extension_flags
unchanged
item_data
unchanged
item_reference

The flags in the structures are set by the implementation to indicate support for the extension. These flags will not be set if CSA_X_COM_SUP_EXCLUDE was set on input. The possible values are listed below.

CSA_X_COM_SUPPORTED

the extension for this item_code is supported. If it is a data extension and is passed at

logon, it will be included with the structures used for this session. For extension sets, the required function and data extensions in the set are supported.

CSA_X_COM_NOT_SUPPORTED
: the item_code is not supported. For extension sets, not all required function and data extensions for the set are supported. If this is a data extension or an extension set containing data extensions, the data will not be attached to structures for this session.

CSA_X_COM_DATA_EXT_SUPPORTED
: for extension sets only. This can be returned by the implementation to indicate that all the required data extensions for the set are supported, but not all of the required function extensions. As with CSA_X_COM_SUPPORTED, if this is returned on the logon call, the data extensions will be included with the data structures for this session.

CSA_X_COM_FUNC_EXT_SUPPORTED
: for extension sets only. This can be returned by the implementation to indicate that all the required function extensions for the set are supported, but not all of the required data extensions. Unlike CSA_X_COM_SUPPORTED, if this is returned on the *csa_logon*( ) call, the data extensions available will NOT be included with the data structures for this session and will need to be requested explicitly.

NAME

CSA_X_UI_ID_EXT — specify platform specific user interface information to the XCS functions.

DESCRIPTION

This extension is used by client applications to specify platform specific user interface information to the XCS functions. The user interface information may be used by the XCS implementation to present user dialogues for resolving additional arguments to the XCS call or any other questions that arise when the service performs the function. For example, in a windows-based environment, this would be the parent window handle for the calling application. Note that the XCS implementations are not required to provide a user interface and providing a user interface for one feature does not necessarily imply that a user interface is available for all features of the XCS.

USED BY

csa_add_calendar( )
csa_add_entry( )
csa_add_event( )
csa_add_memo( )
csa_add_todo( )
csa_delete_calendar( )
csa_free_time_search( )
csa_list_calendars( )
csa_list_calendar_attributes( )
csa_list_entries( )
csa_logon( )
csa_lookup( )
csa_query_configuration( )
csa_restore( )
csa_save( )

INPUT

extension_flags

All XCS extension flags are valid. Unspecified flags should always be passed as 0.

Additional flags used by this function extension include the following:

CSA_X_LOGON_UI_ALLOWED

Set if the function should display a dialog box to prompt for logon if required. If not set, the function will not display a dialog box and will return the error CSA_E_USER_NOT_LOGGED_ON if the user is not logged on. CSA_X_ERROR_UI_ALLOWED - Set if the function may display a dialog box on encountering recoverable errors. If not set, the function may not display a dialog box and will simply return an error code. CSA_X_LOOKUP_RESOLVE_UI -- This flag is only used in the *LookUp*( ) function. Set if the XCS implementation should attempt to disambiguate names by presenting a name resolution dialog to the user. If this flag is not set, resolutions which do not result in a single name will return the error CSA_E_AMBIGUOUS_USER on services that must resolve to a single name. Services that can return multiple names will return a list as indicated by other function parameters. This flag is optional for implementations to support.

CSA_X_LOOKUP_DETAILS_UI

This flag is only used in the *LookUp*( ) function. If set, the function will display details UI for the user pointed to in user_in. This will only act on the first user in the list. If the name resolves to more than one address, this will not be carried out and the error

CSA_E_AMBIGUOUS_USER will be returned. CSA_X_LOOKUP_ADDRESSING_UI - This flag is only used in the *LookUp*( ) function. If set, the function will display UI to allow creation of an attendee list for addressing a meeting request and general directory browsing. The attendee list passed to the function will be the original attendee list for the UI. The function will return the list of users selected by the user. This flag is optional for implementations to support. CSA_X_ADD_DEFINE_ENTRY_UI - This flag is only used in the *AddEntry*( ) function. Set if the function will display a UI to define the details for the entry. item_data zero item_reference A pointer to an identifier for a user interface (for example, dialog window) for use in resolving any questions which might otherwise result in an error and queries the use for additional information as required.

OUTPUT

extension_flags
unchanged

item_data

- Set to CSA_X_INVALID_UI_ID if the specified user interface id is invalid or no longer valid.
- Set to CSA_X_LOGON_FAILURE if the logon failed.
- Set to CSA_X_USER_CANCEL if the user canceled the function.

item_reference
unchanged

NAME

CSA_X_XT_APP_CONTEXT_EXT — specify the Xt application context to the *csa_register_callback*( ) function.

DESCRIPTION

This extension is used by Xt based applications to specify the Xt application context to the *csa_register_callback*( ) function. The application context may be used XCS implementations to facilitate the monitoring of update events. When the application context is provided, the application will be able to get update callbacks asynchronously. Other types of applications will have to call *csa_call_callbacks*( ) to activate the callback process.

USED BY

csa_register_callback( )

INPUT

extension_flags
All XCS extension flags are valid. No further flags are defined.

item_data
Xt application context (XtAppContext).

item_reference
NULL.

OUTPUT

extension_flags
Unchanged.

item_data
Unchanged.

item_reference
Unchanged.

Chapter 7

Platform-specific Information

XCS implementors are encouraged to provide run-time binding interfaces to their XCS service implementations. In general, these interfaces are platform and/or operating system dependent. This section provides several general requirements and platform-specific requirements for several common platforms and operating systems. Unless specified otherwise below, the following definitions apply to all platforms:

16 bit int	CSA_sint16
32 bit long int	CSA_sint32
unsigned char	CSA_uint8
16 bit unsigned int	CSA_uint16
32 bit unsigned long int	CSA_uint32
32 bit pointer	CSA_buffer
32 bit char pointer	CSA_string
CSA_uint32	CSA_entry_handle
CSA_uint32	CSA_session_handle

7.1 Explicit and Implicit Binding

All functions in the CS API should be link-able implicitly and explicitly. Implicit linking builds the linkage of the application and the XCS service implementation into the application. Explicit linking requires the application to contain run-time code that links a XCS service implementation. It is also recommended that all extension functions be loaded explicitly, since their absence on some XCS implementations would otherwise prevent the application from loading. Static and dynamic linking mechanisms are defined for several common platforms below.

7.2 Apple Macintosh Binding

For static linking, applications should use the Pascal calling convention and 32-bit flat pointers to call an Apple Macintosh XCS implementation. For dynamic linking, contact Apple Computer, Inc. The XCS implementation should always attempt to provide Apple International Strings (ISTRING).

7.3 MS-DOS Binding

For static linking, applications should use *far* calls, the C calling convention, and 32-bit segmented *far* pointers to call an MS-DOS XCS implementation. This is compatible with the Microsoft C *large* memory model. Any future changes to this mechanism will be published by Microsoft. The XCS implementation should always attempt to provide code page 437 or 850.

7.4 MS-Windows 3.x Binding

For dynamic linking, MS-Windows 3.x XCS implementations should use Dynamic Linked Libraries and link by name to the XCS functions. At run time, to determine if a XCS service is available, applications should call to look for the XCS variable in the **CALENDAR** section of WIN.INI. If this variable is present and non-zero, it indicates that a CSA.DLL library is available. If the XCS variable is not found or is zero, then the functions cannot be called. Any future changes to this mechanism will be published by Microsoft. XCS functions should be called *far*, using the Pascal calling convention, and 32-bit segmented *far* pointers. XCS structures will be aligned to every 4 byte (32 bit) boundaries. This will not apply to the byte fields in the time structure or the counted string structure. The XCS implementation should always attempt to provide code page 1252.

7.5 MS-Windows NT Binding

For dynamic linking, MS-Windows NT XCS implementations should use Dynamic Linked Libraries and link by name to the XCS functions. At run time, to determine if a XCS service is available, applications should query the registry to see if XCS is available. The exact mechanism for this will be published by Microsoft. XCS functions should be called using the STDCALL calling convention.

7.6 OS/2 1.x and 2.x 16-bit DLL Binding

For dynamic linking, OS/2 1.x and 2.x 16-bit XCS implementations should use Dynamic Linked Libraries and link by name to the functions. At run time, to determine if a XCS service is available, applications should call *WinQueryProfileInt*( ) to look for the XCS variable in the **CALENDAR** section of OS2.INI. The variable will indicate whether the DLL is 16-bit or 32-bit. If this variable is present and non-zero, it indicates that a CSA.DLL library is available. If the XCS variable is not found or is zero, then the functions cannot be called. Any future changes to this mechanism will be published by IBM. XCS functions should be called *far*, using the System calling convention, and 32-bit segmented *far* pointers. The XCS implementation should always attempt to provide code page 850.

7.7 OS/2 2.0 32-bit DLL Binding

For dynamic linking, OS/2 2.0 32-bit XCS implementations should use Dynamic Linked Libraries and link by name to the functions. At run time, to determine if a XCS service is available, applications should *WinQueryProfileInt*( ) look for the XCS variable in the **CALENDAR** section of OS2.INI. The variable will indicate whether the DLL is 16-bit or 32-bit. If this variable is present and non-zero, it indicates that a CSA.DLL library is available. If the XCS variable is not found or is zero, then the functions cannot be called. Any future changes to this mechanism will be published by IBM. XCS functions should be called *far*, using the System calling convention, and 32-bit flat *far* pointers. The XCS implementation should always attempt to provide code page 850.

7.8 UNIX SVR4 Binding

For dynamic linking, implementations should comply with the UNIX System V Release 4.0 System V Application Binary Interface (ABI) specification and link by name to the functions. At run time, to determine if a XCS service is available, applications should look for the XCS implementation on the absolute path /usr/lib/XAPI/libCSA.so. The implementation for the system will be placed in this location. Any future changes to this mechanism will be published by your UNIX vendor. XCS functions and structures should use the System calling convention. The XCS implementation should always attempt to provide code page 850.

Chapter 8

Extended Recurrence Rule Grammar

The material in this section is included in this specification for reference information. This section defines an extended recurrence rule grammar that may be useful to implementations wishing to extend the capability of the basic recurrence rule defined by this specification. The material is equally applicable to extended support of the exception rules for repeating events.

8.1 Rule Introduction

A recurrence rule is made up of one or more recurrence frequencies. The frequencies express the granularity of the repeating event. The smallest granularity is based on minutes, the largest is based on years. Each frequency is immediately followed by an interval. The interval helps to define how often the frequency repeats (daily, every third day, etc):

```
D2
```

where D is the *frequency* and 2 is the *interval*.

```
M5     Repeat every five minutes
D1     Repeat daily
D2     Repeat every other day
D3     Repeat every third day
W1     Repeat weekly
W2     Repeat every other week
W3     Repeat every third week
```

The meaning of the interval depends on the frequency. As an example, the 5 in M5 is in minutes while the 3 in D3 is in days.

A rule can end with the duration symbol, #, followed by a number. This defines the number of times the repetition occurs (including the first time).

```
D2 #5
```

where, #5 is the Duration. In this example, the event occurs every other day and the duration indicates it occur 5 times.

There may be other information between the frequency and the duration that supplements the meaning of the rule:

```
D2 1200 1600 #5
```

In this example, the event occurs every other day at 1200 and 1600 for a total of 10 events. The duration controls the number of times the rule occurs. In this case the rule defines two events (1200 and 1600) so a total of 10 (2 x 5) events are generated.

A rule can be made up of several recurrence rules:

```
MP6 1+ MO #5 D2 1200 1600 #5 M5 #3
```

This recurrence rule is made up of three recurrence rules. Every time the first rule executes (every 6 months) it executes the next rule to the right. If there is not a rule to the right an event is generated. In this case there is a daily frequency rule to the right of the monthly frequency rule. It executes twice a day; starting on the first Monday of the month. The daily frequency rule executes a total of ten times. Since there is a rule following the daily rule it executes it each time the daily frequency rule executes. The minute frequency rule is executed three times, every time

the daily frequency rule executes, for a total of six times a day. The above rules generate a total of 150 (5 * (2 * 5) * 3) events.

An occurrence specifier may be used in some rules. The occurrence specifier controls which occurrence of a weekday in a month an event occurs on: 1+, 2+, ... 5+ for the first occurrence, second, ...fifth occurrence of the month. 1-, 2-...5- for the last occurrence, second to last occurrence, etc.

```
MP2 1+ 2- FR #3
```

where, 1+ and 2- are Occurrence Specifiers. This rule defines an event which happens every other month on the first Friday and second to the last Friday of the month. It generates a total of six events.

The end marker symbol $ can be used to terminate a rule early:

```
W1 0100$ 0200 0300 #4
```

The above rule generates events weekly for 4 weeks at 0100, 0200 and 0300. On the last (fourth week) an event is generated at 0100 only. Recurrence rules are written from largest granularity to smallest.

8.2 Grammar

The following grammar defines the extended recurrence rule syntax.

{}	0 or more
[]	0 or 1
start	<minuteop> [<enddate>] | <daily> [<enddate>] | <weekly> [<enddate>] | <monthlybypos> [<enddate>] | <monthlybyday> [<enddate>] | <yearlybymonth> [<enddate>] | <yearlybyday> [<enddate>]
endmarker	$
enddate	ISO 8601 (clause 5.4.1) string(e.g. 19940712T101530Z) in UTC
interval	<digits>
duration	#<digits>
lastday	LD
plus	+
minus	-
daynumber	<1-31>[<endmarker>] | <lastday>
daynumberlist	daynumber {<daynumber>}
month	<1-12>[<endmarker>]
monthlist	<month> {<monthlist>}
day	<1-366>[<endmarker>]
daylist	<day> {<daylist>}
occurrence	<1-5><plus>[<endmarker>] | <1-5><minus>[<endmarker>]
occurrencelist	<occurrence> {<occurrencelist>}
weekday	<SU | MO | TU | WE | TH | FR | SA>[<endmarker>]
weekdaylist	<weekday> {<weekdaylist>}
time	<hhmm>[<endmarker>]
timelist	<time> {<timelist>}
daytime	<weekday> {<timelist>} {<daytime>}
weekdaytime	<occurrencelist> <weekdaylist> {<timelist>} {<weekdaytime>}
minuteop	M<interval> [<duration>]
daily	D<interval> [<timelist>] [<duration>] [<minuteop>]
weekly	W<interval> <daytime> [<duration>] [<minuteop>]
monthlybypos	MP<interval> [<weekdaytime>] [<duration>][<weekly> | <daily> | <minuteop>]
monthlybyday	MD<interval> [<daynumberlist>] [<duration>] [<weekly> | <daily> | <minuteop>]
yearlybymonth	YM<interval> [<monthlist>] [<duration>] [<monthlybyday> | <monthlybypos> | <weekly> | <daily> | <minuteop>]
yearlybyday	YD<interval> [<daylist>] [<duration>] [<monthlybyday> | <monthlybypos> | <weekly> | <daily> | <minuteop>]

8.3 Rule Grammar Glossary

enddate Controls when a repeating event terminates. The enddate is the last time an event can occur.

enddate Controls when a repeating event terminates. interval Defines the frequency in which a rule repeats.

duration Controls the number of events a rule generates.

lastday Can be used as a replacement to daynumber to indicate the last day of the month.

daynumber A number representing a day of the month.

month A number representing a month of the year.

day A number representing a day of the year.

occurrence Controls which week of the month a particular weekday event occurs.

weekday A symbol representing a day of the week.

timeThe time in hours and minutes using a 24 hour clock.

daytime Controls which days (and hours) of the week a weekly event occurs.

weekdaytime Used in monthly events to specify which weeks and days an event occurs.

minuteop Defines a rule that repeats on a particular minute interval.

daily Defines a rule that repeats on a daily basis.

weekly Defines a rule that repeats on a weekly basis.

monthlybypos Defines a rule that repeats on a monthly basis on a relative day and week.

monthlybyday Defines a rule that repeats on a monthly basis on an absolute day.

yearlybymonth Defines a rule that repeats on specific months of the year.

yearlybyday Defines a rule that repeats on specific days of the year.

8.4 Policies

1. The duration portion of a rule defines the total number of events the rule generates, including the first event. As an example, the rule MP1 #3 W1 #3 starting on 1/1/94 would generate events on 1/1/94, 1/8, 1/15, 2/5/94, 2/12, 2/19, 3/5/94, 3/12, 3/19.
2. The duration granularity is defined by the recurrence frequency immediately preceding the duration portion of the rule. For example, D1 #5 M15 #4 establishes a repeating event which happens for five days, four times per day.
3. Information, not contained in the rule, necessary to determine the next event time and date is derived from the event start time.
4. If no specific time is indicated in the recurrence rule it is taken from the event.
5. If an end date and a duration for the first rule in a nested rule are specified in the rule, then the recurring event ceases when the end date is reached or the number of events indicated in the duration occur; whichever comes first.
6. If the duration or and end date is not established in the rule (for example ‘‘D2’’) the event occurs twice. That is D2 is equivalent to D2 #2.
7. If an endmark is used in a second or later rule of a nested rule, then the endmark is applied each time that rule is executed by the previous rule. YM1 1 6 #1 MD1 7$ 14 generates events on 1/7 1/14 2/7 6/7 6/14 7/7.
8. If an endmark is used on a day of the week which is followed by several times (TU$ 1200 1300) or an endmark is used on a week occurrence that is followed by several weekdays (1+$ TU WE) the repeating event stops after the last time or week day in the list is executed.
9. If a rule has an ambiguity with respect to whether it will repeat on a specific day (12th of the month) vs on a relative day (2nd Friday of the month), the specific day takes precedence. The only exception to this policy is policy 14.
10. A duration of #0 means repeat this event forever.
11. Nested rules can not have a duration of 0. These are not allowed:

```
YM1 6 #10 MP1 1+ SA #0
D5 0600 0800 #5 M5 #0
```

12. Using the occurrence specifier 5+ (for example 5th Friday) or 5- (for example 5th from last Friday) in a month that does not contain 5 weeks does not generate an event and thus does not count against the duration. The same applies to providing a day of the month that does not occur in the month: 31st.
13. The start time and date of an event must be in-sync with one of the event slots defined by its occurrence rule. The following are not allowed:

```
Initial Appt Time:    1300
Recurrence Rule:      D1 1400 #5
Initial Appt Date:    7/1/94 (Friday)
Recurrence Rule:      W1 MO TH #5
```

The following are acceptable:

```
Initial Appt Time:    1300
Recurrence Rule:      D1 #5 or D1 1300 #5
Initial Appt Date:    7/1/94  (Friday)
Recurrence Rule:      W1 MO FR #5 or W1 #5
```

14. If the optional <weekdaytime> information is missing from a <monthlybypos> frequency, the information is derived from the initial event. The <occurrence> used in the recurring event is a count from the beginning of the month to the event date and the <weekday> used is the day of the week the initial event is scheduled to occur on. If the <monthlybypos> frequency does not list a week day (for example, SU) in the rule, then the week day is established from the initial event information. As an example, the rule MP1 #3 used in an event with a start date of 7/20/94 (which is the third Wed of the month) will repeat on 8/17/94 which is the third Wed of the month.

15. The next event of a higher order rule does not execute until all the events of a subrule are generated. If the next event of a higher order rule comes earlier in time than the last event of a subrule then the missed events are not generated. In other words, subrules can not interleave events with other subrules. The following results in indeterminate results because the minute subrule which begins to execute at 0630 generates events beyond 0700 which is when the daily subrule begins executing again:

    ```
    D1 0630 0700 #4 M45 #5
    ```

    Another incorrect rule:

    ```
    xMP1 1+ 1- #3 W2 TU TH #5
    ```

8.5 Examples

Hourly for 12 hours (12:00, 1:00,...10:00, 11:00):

```
M60 #12
```

Every 5 minutes for 1 hour (1:00, 1:05, 1:10,...1:50, 1:55):

```
M5 #12
```

Daily, for 5 days:

```
D1 #5
```

Daily, for 5 days repeating at 10 minute intervals for 1 hour: for example 6/1 at 12:00, 12:10, 12:20, ... 12:50; 6/2 at 12:00, 12:10, ...

```
D1 #5 M10 #6
```

Every other day, two times:

```
D2
```

Every other day at 6AM, 12noon and 3PM for a duration of two events (span of three days): for example 6/1/94 at 6, 12 and 3PM and 6/3/94 at 6, 12 and 3PM.

```
D2 0600 1200 1500 #2
```

Every other day at 6AM, 12noon and 3PM for a duration of three events (a span of 5 days) stopping at noon on the fifth day: for example 6/1/94 at 6, 12, and 3, 6/3/94 at 6, 12 and 3 and 6/5/94 at 6 and 12.

```
D2 0600 1200$ 1500 #3
```

Weekly at 6am (repeat every 15 minutes for an hour) for five weeks: for example 6:00, 6:15, 6:30, 6:45 on 6/1, 6/8, 6/15, 6/22 and 6/29.

```
D7 0600 #5 M15 #4
```

Weekly at 6am (repeat every 15 minutes for an hour) for four weeks stopping at 6AM on the last event day: for example 6:00, 6:15, 6:30, 6:45 on 6/1, 6/8, 6/15 and 6:00 on 6/22.

```
D7 0600$ #4 M15 #4
```

Weekly at 6am (repeat every 15 minutes for an hour) for 1 week stopping at 6:45AM: for example 6:00, 6:15, 6:30, 6:45 on 6/1.

```
D7 0600 #1 M15 #4
or
D7 #1 M15 #4 /* start time defined in appt entry */
or
M15 #4 /* start time defined in appt entry */
```

Weekly for four weeks:

```
W1 #4
```

Biweekly on Monday and Tuesday for 2 occurrences ending on a Monday:

```
W2 MO$ TU #2
```

Weekly on Tuesday and Thursday at the time specified in the appt and repeated at time + 5 minutes:

```
W1 TU TH #3 M5 #2
```

Weekly on Tuesday at 1200 and 1230 and Thursday at 1130 and 1200 for 10 weeks:

```
W1 TU 1200 TH 1130 #10 M30
or
W1 TU 1200 1230 TH 1130 1200 #10
```

Weekly on Tuesday at 1200 and 1230 and Thursday at 1130 and 1200 for 10 weeks stopping on the last TU at 1230:

```
W1 TU$ 1200 TH 1130 #10 M30
or
W1 TU$ 1200 1230 TH 1130 1200 #10
```

Weekly on Tuesday at 1200 and 1230 and Thursday at 1130 and 1200 for 10 weeks stopping on the last Tuesday at 1200:

```
W1 TU 1200$ 1230 TH 1130 1200 #10
```

Monthly for 1 year:

```
MP1 #12
```

Every other month on the first and last Friday of the month for 5 months:

```
MP2 1+ 1- FR #3
```

Every other month on the first and last Friday of the month for 5 months stopping on the first Friday in the fifth month:

```
MP2 1+$ 1- FR #3
```

Every six months on the first Monday of the month (repeat for 5 days) for 24 months:

```
MP6 1+ MO #5 D1 #5
```

Every six months on the first Monday of the month (repeat every other day at 0600, 1200 and 1500 for 20 days) for 24 months:

```
MP6 1+ MO #5 D2 0600 1200 1500 #10
```

Every six months on the first Monday of the month (repeat every other day at 0600, 1200 and 1500 for 20 days (repeat every 5 minutes for 3 times)) for 24 months:

```
MP6 1+ MO #5 D2 0600 1200 1500 #10 M5 #3
```

Every six months on the first Monday of the month and the second to last Thursday of the month (repeat five minutes later) for 24 months:

```
MP6 1+ MO 2- TH #5 M5 #2
```

Every six months on the first SU and MO at Noon, the second TU and WE at 1:00PM and the third TH and FR at 2:00PM:

```
MP6 1+ SU MO 1200 2+ TU WE 1300 3+ TH FR 1400 #4
```

Every month on the 7th for 12 months:

```
MD1 7 #12
```

Every month on the 7th, 14th, 21st, 28th for 12 months

```
MD1 7 14 21 28 #12
```

Every month on the 10th and 20th for 24 months - daily for 5 days at 0600, 1200 and 1600 - every 15 minutes for an hour:

```
MD1 10 20 #24 D1 0600 1200 1600 #5 M15 #4
```

Yearly on the 1st, 6th and 12 month on the first Monday and last Friday of the month:

```
YM1 1 6 12 #5 MP1 1+ MO 1- FR
```

Every other year on the 6th month (on the 12th day) for 5 years.

```
YM2 6 #3 MD1 12
```

Yearly on the 7th 14th 21st and 28th of the 1st 3rd and 8th month and on the 7th and 14th of the 2nd, 4th and 9th months ending on the 4th month, 14th day of the 5th year:

```
YM1 1 3$ 8 #5 MD1 7 14$ 21 28
```

Yearly on the 6th, 9th and 10th month on all weekends of the month:

```
YM1 6 9 10 #10 MP1 1+ 2+ 3+ 4+ 1- SA SU #1
```

Yearly on the 6th month for 10 years, weekly on Tuesday and Thursday at 1100 and 1300 for 4 weeks:

```
YM1 6 #10 W1 TU TH 1100 1300 #4
```

Yearly on the 1st, 100th, 200th and 300th day for 4 years:

```
YD1 1 100 200 300 #4
```

Yearly on the 1st - 5th days and 100th - 104th days:

```
YD1 1 100 #5 D1 #5
```

Yearly on the 1st - 5th days and 100th - 104th days stopping on 1/2/99:

```
YD1 1 100 D1 #5 19990102T000000Z
```

Glossary

API
Application Program Interface

CSA
Calendaring and Scheduling Api. Used as the prefix name for the Calendaring and Scheduling API headers and definitions. The term CSA, when used by the XAPIA to refer to this API, is synonymous with XCS.

interval
Defines the frequency in which a rule repeats.

day
A number representing a day of the year.

daily
Defines a rule that repeats on a daily basis.

daynumber
A number representing a day of the month.

duration
Controls the number of events a rule generates.

enddate
Controls when a repeating event terminates. The enddate is the last time an event can occur.

lastday
Can be used as a replacement to daynumber to indicate the last day of the month.

minuteop
Defines a rule that repeats on a particular minute interval.

month
A number representing a month of the year.

monthlybyday
Defines a rule that repeats on a monthly basis on an absolute day.

monthlybypos
Defines a rule that repeats on a monthly basis on a relative day and week.

occurrence
Controls which week of the month a particular weekday event occurs.

weekday
A symbol representing a day of the week.

weekly
Defines a rule that repeats on a weekly basis.

XAPIA
X Application Programming Interface Association.

XCS
X/Open Calendaring and Scheduling: refers to this API specification document.

yearlybymonth
Defines a rule that repeats on specific months of the year.

yearlybyday
Defines a rule that repeats on specific days of the year.

Index