INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.446
(08/97)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATION

Message Handling Systems

Common messaging call API

ITU-T Recommendation X.446

(Previously CCITT Recommendation)

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATION

PUBLIC DATA NETWORKS	X.1–X.199
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEM INTERCONNECTION	X.200–X.299
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	X.300–X.399
General	X.300–X.349
Satellite data transmission systems	X.350–X.399
MESSAGE HANDLING SYSTEMS	**X.400–X.499**
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	X.600–X.699
Networking	X.600–X.629
Efficiency	X.630–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	X.700–X.799
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	X.850–X.899
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999

For further details, please refer to ITU-T List of Recommendations.

ITU-T RECOMMENDATION X.446

COMMON MESSAGING CALL API

Summary

This Recommendation specifies a simple call interface through which messaging-reliant applications may invoke the services of MHS across a standardized programming interface. The Recommendation was generated cooperatively with the XAPI Association and defines the application programming interface being implemented for MHS by the world's major vendors and service providers.

Source

ITU-T Recommendation X.446 was prepared by ITU-T Study Group 7 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on the 9th of August 1997.

FOREWORD

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

CONTENTS

Page

COMMON MESSAGING CALL API

(Geneva, 1997)

1 Introduction

This clause introduces the Common Messaging Call (CMC) Application Program Interface (API) and its specifications. It indicates the purpose of the interface, provides an overview of it, details abbreviations, provides document references, explains the level of abstraction of the interface, defines C naming conventions, and specifies conformance requirements.

This Recommendation is an enhancement of the first version of the CMC API, published in June 1993 by the X.400 API Association. This Recommendation extends the messaging-aware application support in the original document with support for messaging-reliant applications.

1.1 Purpose

The purpose of this Recommendation is to specify a high-level messaging application program interface that can be supported by most messaging services deployed today. The API is intended to enable application programmers to easily integrate messaging, and thus communications, into their applications, creating a large body of *messaging-enabled applications*.

This Recommendation is directed toward messaging service developers who might wish to support such an application program interface. This Recommendation may also guide application developers in understanding implementation-independent features of the Common Messaging Call API. The application developers must follow manuals provided by the system they are using for messaging support.

1.2 Overview

The Common Messaging Call Application Program Interface provides a set of high-level functions for messaging-enabled applications to send and receive electronic messages.

Within the range of messaging-enabled applications, there are messaging-aware applications and messaging-reliant applications.

Messaging-aware applications are those that can function quite satisfactorily as stand-alone applications, but which might connect to a messaging service to provide enhanced functionality. An example would be a word processing or spreadsheet application that has the capability to send the document or file using a FILE-SEND option off of the menu.

Messaging-reliant applications are those which are inherently dependent on the existence of a messaging service to carry out their functionality. Examples of these are Electronic Data Interchange (EDI), information distribution applications, conferencing/collaboration applications, and possibly some distributed databases.

This interface is designed to be independent of the actual messaging protocol employed between sender and recipient. The interface will support the creation and reception of standard message formats such as X.400 and SMTP/MIME (RFC 822/RFC 1521) as well as proprietary message formats. This is achieved through generic definition of capabilities common to most messaging protocols, plus a mechanism for defining extensions, which can be used to invoke protocol-specific services.

The interface is also designed to be independent of the operating system and underlying hardware used by the messaging service.

Another important consideration in the design of this API is to enable simple application actions to be taken with a minimum number of function calls while allowing more complex actions to be possible as well. To achieve these often conflicting objectives, the CMC API has two interfaces: a Simple CMC interface and a Full CMC interface. The Simple CMC interface provides a minimum number of function calls needed to send or receive a message by messaging-aware applications. The Full CMC provides a more complete set of function calls in order to provide for more robust message-reliant applications.

The CMC API is designed to be complementary to existing XAPIA-X/OPEN APIs such as the XMHS and XMS API.

The CMC interface is designed to allow a common interface over virtually any electronic messaging service. For each CMC implementation, the view/capabilities presented by CMC must be mapped to the view/capabilities of the underlying messaging service.

To maximize interoperability between CMC applications which use similar underlying messaging services, it is critical that a common mapping be defined by the industry segment representing the relevant messaging protocol or interface.

To that end:

- the Recommendation defines the common mapping between Simple CMC and the X.400 message store protocol;
- standards bodies, vendors, or vendor groups representing a specific messaging protocol or interface are encouraged to define a common mapping between CMC and the relevant messaging protocol or interface.

To maximize interoperability between CMC applications which use differing underlying messaging services, it is critical that mapping definitions be designed with such interoperability in mind.

To that end, the following guidelines are offered:

- map message text strings to international character sets, wherever appropriate or possible;
- map message attachment types to commonly recognized attachment types, wherever appropriate or possible.

This list is not comprehensive; additional guidance may be offered in the future once implementations are deployed.

1.3 Terminology

1.3.1 Definitions

This Recommendation defines the following terms:

1.3.1.1 full CMC: A messaging-enabled API that provides the functions to support message-reliant applications.

1.3.1.2 simple CMC: A messaging-enabled API that provides the functions to support messaging-aware applications.

1.3.1.3 T.611: ITU-T PCI for use with facsimile, telex, and teletex services.

1.3.2 Abbreviations

This Recommendation uses the following abbreviations:

API	Application Program Interface
CMC	Common Messaging Call
XAPIA	X.400 Application Program Interface Association
XMHS API	X/OPEN Application Program Interface to Electronic Mail (X.400)

XMS API	X/OPEN Message Store Application Program Interface
XOM API	X/OPEN OSI-Abstract-Data Manipulation API
UI	User Interface

1.4 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

1.4.1 Identical Recommendations – International Standards

– ITU-T Recommendation X.402 (1995) | ISO/IEC 10021-2:1996, *Information technology – Message Handling Systems (MHS): Overall architecture*.

– ITU-T Recommendation X.411 (1995) | ISO/IEC 10021-4:1997, *Information technology – Message Handling Systems (MHS): Message transfer system: Abstract service definition and procedures*.

– UIT-T Recommendation X.413 (1995) | ISO/IEC 10021-5:1996, *Information technology – Message Handling Systems (MHS): Message store: Abstract service definition*.

– UIT-T Recommendation X.419 (1995) | ISO/IEC 10021-6:1996, *Information technology – Message Handling Systems (MHS): Protocol specifications*.

– UIT-T Recommendation X.420 (1996) | ISO/IEC 10021-7:1997, *Information technology – Message Handling Systems (MHS): Interpersonal messaging system*.

1.4.2 Paired Recommendations – International Standards equivalent in technical content

– CCITT Recommendation X.208 (1988), *Specification of Abstract Syntax Notation One (ASN.1)*.

  ISO/IEC 8824:1990, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*.

– CCITT Recommendation X.209 (1988), *Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)*.

  ISO/IEC 8825:1990, *Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*.

– ITU-T Recommendation X.400/F.400 (1996), *Message Handling System and Service Overview*.

  ISO/IEC 10021-1:1997, *Information technology – Text communication – Message-Oriented Text Interchange Systems (MOTIS) – Part 1: System and Service Overview*.

1.4.3 Additional references

– ISO 8601:1988, *Data elements and interchange formats – Information interchange – Representation of dates and times*.

– ISO 9070:1991, *Information technology – SGML support facilities – Registration procedures for public text owner identifiers*.

– ISO/IEC 10021-3:1990, *Information technology – Text Communication – Message-oriented Text Interchange Systems (MOTIS) – Part 3: Abstract Service Definition Conventions*.

– IMAP – "Internet Message Access Protocol", Version 4, RFC 1730, December 1994.

– MIME – "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993.

– RFC 822 – "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

– SMTP – "Simple Mail Transfer Protocol", RFC 821, August 1982.

– XMHS API – API to Electronic Mail (X.400), CAE Specification, X/Open Company Limited and X.400 API Association, 1991.

- XMS API – Message Store API, Preliminary Specification, X/Open Company Limited and X.400 API Association, 1991.
- XOM API – OSI-Abstract-Data Manipulation API, CAE Specification, X/Open Company Limited and X.400 API Association, 1991.
- ANSI C – American National Standard for Information Systems – Programming Language C, X3.159-1989.

1.5 Levels

This Recommendation defines the CMC API at two levels of abstraction. It defines a "generic" interface independent of any particular programming language, and a C language interface based on the American National Standard for the C Programming Language. The "generic" interface is included to guide the development of other language-specific specifications, e.g. PASCAL.

For readability, the specifications of the generic and C interfaces are combined. In clause 4, the CMC data structures are described generically, but include a C declaration. In clause 6, the CMC functions are specified generically, but include a synopsis written in C. For clarity, constants and error codes throughout this Recommendation are written in the C syntax described below. Annex A gives a summary of the C declarations and constants used throughout the Recommendation.

1.6 C naming conventions

How an identifier for an element of the C interface is derived from the name of the corresponding element of the generic interface depends on the element's type, as specified in Table 1 below. 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.

Table 1/X.446 – Derivation of C naming conventions

Element Type	Prefix	Case
Data type	CMC_	Lower
Data value	CMC_	Upper
Function	cmc_	Lower
Function argument	none	Lower
Function result	none	Lower
Constant	CMC_	Upper
Error	CMC_E_	Upper
Macro	CMC_	Upper
Object Class	CMC_OC_	Upper
Content Type	CMC_CT_	Upper
Property	CMC_PT_	Upper
Structure Tag	CMC_TAG_	Upper
Reserved for extension sets	CMC_XS_	any
Reserved for extensions	CMC_X_	any
Reserved for use by implementors	CMCP	any

Elements with the prefix "CMCP" (any case) are reserved for internal proprietary use by implementors of the CMC service. They are not intended for direct use by programs written using the CMC interface.

The prefixes "CMC_XS_" and "CMC_X_" (in either upper or lower case) are reserved for extensions of the interface by vendors or groups.

For constant data values, there is usually an additional string appended to "CMC_" to indicate the data structure or function to which the constant data value pertains.

2 CMC architecture

This clause describes the functional architecture underlying the CMC API. It defines the CMC functional model, the CMC configuration model, the CMC API computational model, and CMC object model. The functional model defines the messaging functions standardized by this Recommendation. The configuration model defines how multiple CMC implementations may coexist within a given platform. The computational model defines common characteristics of the CMC programming interface. The object model describes characteristics of objects defined by this Recommendation.

2.1 Functional model

The CMC interface is defined between a messaging-enabled application and a messaging service. The messaging service in turn may support multiple messaging protocol services, each using different messaging formats and protocols, e.g. X.400, RFC 822 and RFC 1521. All functions in this interface are designed to be independent of the messaging protocol services. However, the API does allow protocol-specific functions to be invoked through defining implementation-specific properties and through the use of extensions (see 2.2.5, Extensions).

The CMC interface is depicted in Figure 1 below.

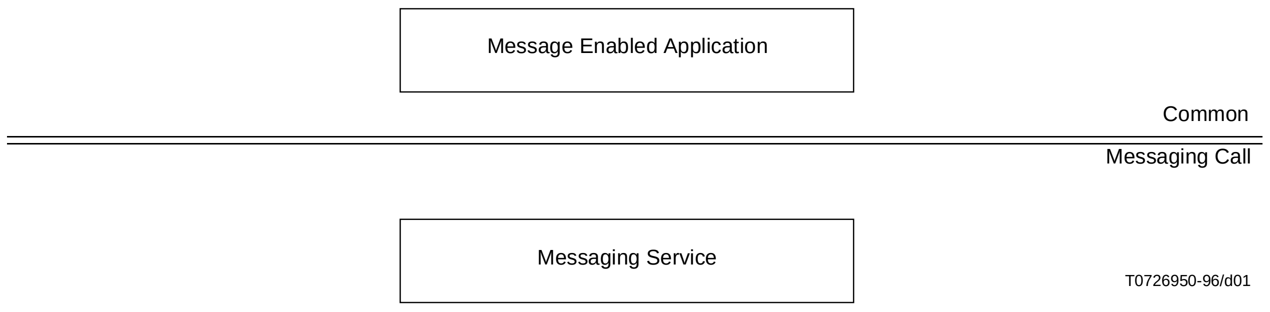


Figure 1/X.446 – Positioning of the Common Messaging Call API

The functional components underlying the CMC API are shown in Figure 2.

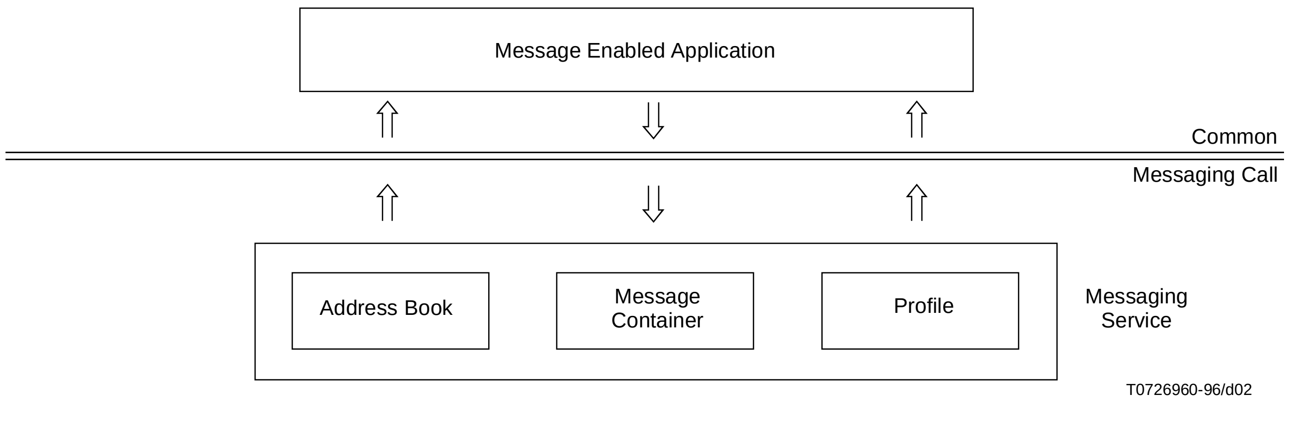


Figure 2/X.446 – Model of the Common Messaging Call API

There are three functional components in the CMC API: address book, message container and profile. The address book holds recipient and distribution list information for message addressing. The message container holds messages. Common message containers are the inbox, outbox, and sent mailbox. The profile contains information related to the CMC implementation and user information.

The interaction of a messaging-enabled application and these functional components is specified by the CMC computational model.

2.2 Computational model

The CMC computational model defines the interfaces defined by the specification and common characteristics of these interfaces. These common characteristics include the concept of a CMC session, character set support, an extension mechanism, and event notification.

2.2.1 Interfaces

The CMC API defines two interfaces: Simple CMC and Full CMC. Simple CMC is intended to offer basic messaging functionality for messaging-aware applications. Full CMC is designed to offer enhanced messaging functionality for messaging-enabled applications.

2.2.1.1 Simple CMC

The Simple CMC interface is backwards compatible with the CMC 1.0 implementation. Simple CMC adds a new message type, CMC: REPORT, to CMC 1.0, to allow delivery and non-delivery reports related to an original message to be consolidated in a single report message, consistent with several X.400 implementations.

The Simple CMC interface supports three principle tasks: sending messages, reading messages, and looking up addressing information. The functions of this interface are intended to provide messaging-enabled support to messaging-aware applications. These are applications that do not depend on mail services to accomplish their basic functions (e.g. word processor, spreadsheet, image, or document management applications). The access to mail services permits these applications to be better utilized within an enterprise computing environment.

To send a message, the messaging-enabled application must first establish a session with the messaging service through the **cmc_logon()** function or interactively by setting the LOGON_UI_ALLOWED flag in the extensions argument of the **cmc_send()** function. An application submits a message to the submission messaging service through a **cmc_send()** function. The messaging-enabled application is responsible for populating the CMC message structure used in the **cmc_send()** function. The messaging-enabled application may also use a more limited **cmc_send_documents()** function to send a message. This function is primarily intended for calling from a macro language. The closure of a session is accomplished through the **cmc_logoff()** function.

To retrieve a message, the messaging-enabled application establishes a session through the **cmc_logon()** function. The application can then retrieve a summary of mailbox information through the **cmc_list()** function. Individual messages can be retrieved through the **cmc_read()** function. The **cmc_act_on()** function allows the user to act on a message in the mailbox (e.g. delete it). Memory allocated by the system for structures is released by passing the returned pointer to the **cmc_free()** function. The closure of a session is accomplished through the **cmc_logoff()** function. Simple CMC only standardizes access to the Inbox message container. Access to other message containers through the Simple CMC interface may be provided through vendor-specific extensions.

To look up names, the messaging-enabled application establishes a session through the **cmc_logon()** function or interactively by setting the LOGON_UI_ALLOWED flag in the extensions argument of the **cmc_look_up()** function. The application then uses **cmc_look_up()** to translate a user-friendly name into a messaging address. Memory allocated by the system for structures is released by passing the returned pointer to the **cmc_free()** function. The closure of a session is accomplished through the **cmc_logoff()** function. The address books searched via the **cmc_look_up()** function is implementation-dependent. Searches of specific address books through the Simple CMC may be provided through vendor-specific extensions.

2.2.1.2 Full CMC

The Full CMC interface augments the messaging-aware functions provided by the Simple CMC interface with additional messaging-enabled functions. The principle tasks provided for include: message composition, access and modification of message folders, stream access to large content information, address book modification. Message-reliant applications depend on messaging services to accomplish their basic functions (e.g. mail front end or agent and workflow management applications). The access to mail services is a prerequisite to the functioning of these applications.

The enhanced functions of the Full CMC interface are facilitated by a number of additional data structures. The capabilities provided for by these data structures include: an object-based data model, property model definition for objects that permits the extensible definition of message service objects, content naming that facilitates the support for multimedia content within messages and a robust set of message types (e.g. calendaring and scheduling, workflow, EDI, active messages), and nested container objects to support foldering in message stores and address books.

2.2.2 Session

In both the Simple and Full CMC interfaces, CMC function calls occur within the context of a session. A session is established with the **cmc_logon()** function and terminated with a **cmc_logoff()** function. The **cmc_logon()** function also authenticates the user to the messaging service and sets session attributes. The context of a session is identified by an opaque session id that is returned by the **cmc_logon()** function. Session context attributes include character set and version number. Currently, there is no support for sharing sessions among applications.

For gateway applications, a single user, representing the gateway, may establish sessions on behalf of multiple individual users and therefore have permissions beyond those of an individual user.

2.2.3 Wide character support

The Full CMC interface supports double-byte character strings (e.g. UNICODE). This is accomplished by defining the constant CMC_WCHAR within an application development environment, before the xcmc.h file is included (i.e. set CMC_WCHAR=1). If CMC_WCHAR is not defined, character sizes are single byte. The CMC_WCHAR definition forces all character string definitions in the Full CMC interface to be two bytes per character. Double-byte character strings are supported in Full CMC only. The xcmc.h file prototypes single- or double-byte counterpart function definitions for each API call depending on whether CMC_WCHAR is defined.

This ensures backward compatibility to CMC 1.0 and allows for double-byte character string support in Simple and Full CMC. Implementations export the double-byte functions in a separate DLL. Applications are not allowed to mix the two paradigms together within the same instance of the application. Double-byte character string support is not required for minimum conformance.

2.2.4 Event notification

The Simple CMC interface does not support the notification of events in the underlying service such as notification of new messages. Four functions have been provided in the Full CMC interface to support this functionality. Two modes of notification are supported: polling and callback.

In the polling mode of notification, the application registers an interest in polling for an event with the **cmc_register_event()** function. The application then polls the implementation with an optional time-out period to check whether an event has occurred with the **cmc_check_event()** function. If the event has occurred, the function returns successfully. In addition, event-specific data may be returned by the function. If the application is no longer interested in an event, it calls the **cmc_unregister_event()** function.

The second mode of interaction uses callbacks to application-defined functions. In this mode, the application registers a callback function with the implementation with the **cmc_register_event()** function. The application's callback function is then called automatically when the event occurs. The application may also want to force a callback with the **cmc_call_callbacks()** function. This function is useful in environments where an implementation can only call callbacks when the implementation's code is executing. If the application is no longer interested in an event, it calls the **cmc_unregister_event()** function.

This Recommendation: there is only one standard event to signal the arrival of a new message in a container (CMC_EVENT_NEW_MESSAGES). Data structures associated with this event are specified in 4.6 under the heading Callback Data Structures. When registering for the new message event, the application indicates the containers to be checked for new messages. Multiple containers may be checked. If the application does not register a callback for the event, the application may poll for new messages events on the set of containers specified in the **cmc_check_event()** function. If the application registers a callback function, the function is called when a new message arrives in a container specified in the **cmc_register_event()** function.

This event notification architecture allows new events to be added in future extensions of CMC and through vendor extensions.

2.2.5 Extensions

In both the Simple and Full CMC interfaces, data structures and functions defined in this Recommendation 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 CMC service or passed as output parameters from CMC 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 CMC service allocates the storage for the extension result, if necessary. In this case, the application must free the allocated storage with a call to the **cmc_free()** function.

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

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

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

For more information on extension registration and the extensions defined in this Recommendation, see the annexes.

2.3 Configuration model

The CMC configuration model permits multiple CMC implementations to coexist in a single environment by specifying a CMC Manager as a broker among CMC implementations. Figure 3 shows the relationship of the CMC Manager and CMC implementations.

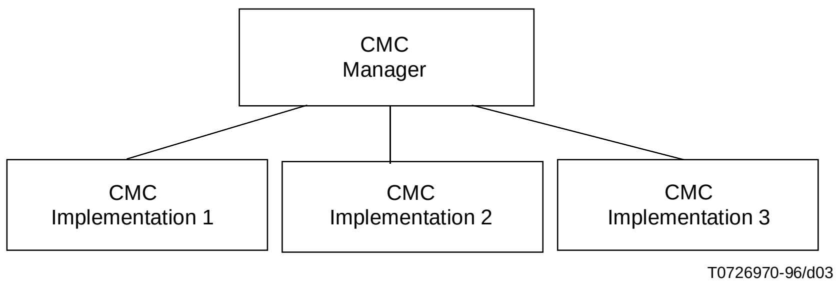


Figure 3/X.446 – CMC manager and CMC implementations

2.3.1 CMC manager

The CMC Manager brokers dispatch tables back to the application through the use of the **cmc_bind_implementation()** function. The dispatch table represents an array of CMC function pointers whose ordinal positions must match the order specified in the CMC header file. The application calls the appropriate CMC implementation function through the implementation's associated dispatch table. This implies that the application must keep copies of pointers to dispatch tables for each CMC implementation that it wishes to bind to. The **cmc_free()** function is used for freeing the dispatch table created on the **cmc_bind_implementation()** function call. The **cmc_unbind_implementation()** function is used to clean up anything else set up by the CMC Manager or CMC implementation. CMC implementations are named as globally unique identifiers (GUIDs). Applications obtain implementation names and GUIDs from vendor header files, vendor supplied documentation, or by convention. The CMC Manager is responsible for mapping the implementation's dispatch table to the address space of the application on platforms whose applications may reside in different address spaces. The CMC Manager may wish to create and manage local copies of any dispatch tables passing through it. They must create and manage their own copies in the different address space case mentioned above. The flow of execution is now defined:

1) The application calls **cmc_bind_implementation()** to obtain a pointer to a dispatch table for a desired CMC implementation.

2) The CMC Manager receives the call and calls the appropriate CMC implementation's **cmc_bind_implementation()** function. The CMC Manager must supply a platform-specific means of determining which CMC implementations exist and where they reside.

3) The CMC implementation receives the **cmc_bind_implementation()** call and creates and populates a dispatch table which is sent back to the CMC Manager. The CMC Manager may at that point wish to create a local copy of the dispatch table.

4) The CMC Manager completes its received **cmc_bind_implementation()** function and returns the pointer to the dispatch table to the application unless remapping needs to be done first.

5) The application proceeds to make calls into any bound CMC implementation which now concurrently exist.

6) The application may at any point call **cmc_free()** to free the memory associated with the dispatch tables created by the CMC Manager and/or the CMC implementation. The **cmc_unbind_implementation()** function is called by the application to signal the CMC Manager to clean up data associated with the binding of particular CMC implementation. The CMC Manager must then make a call to the specified CMC implementation to do the same.

7) When all CMC implementations are unbound, the application may exit or do another **cmc_bind_implementation()**. Bindings which are not symmetrically unbound with **cmc_unbind_implementation()** run the risk of memory leaks and unpredicted resulting behavior.

If applications are using CMC 1.0, they should call the CMC implementation directly rather than through the CMC Manager. CMC 1.0 does not support access to multiple implementations.

2.3.2 Guidelines for platform bindings

CMC 2.0 supports a CMC Manager and multiple implementations of CMC on a single platform. The following guidelines are needed to support the CMC Manager and multiple implementations of CMC:

- Each platform binding must specify a mechanism for implementations of CMC to register and deregister themselves with a CMC Manager.
- The CMC Manager must support at least the CMC 2.0 functions, **cmc_bind_implementation()** and **cmc_unbind_implementation()**.
- The CMC Manager may also support any of the following:
  - interworking with CMC implementations in another address space;
  - interworking with CMC implementations on another machine;
  - browsing for registered CMC implementations.
- Certain platforms may require the CMC implementations to modify the names to the CMC functions to support multiple implementations. The CMC Manager will need to broker mappings to these modified function names.

2.3.3 Query for configuration information

The persistent configuration of the service is available for query by the messaging-enabled application. The application may query the service to determine its support for different version(s) of the CMC API, extensions, and environmental parameters that comprise the configuration. No function is defined in this API for the modification of this configuration information. The form in which this information is stored (e.g. file format) is left undefined by this Recommendation.

Two mechanisms are provided for querying the configuration information. The Simple CMC interface includes a **cmc_query_configuration()** function call. The Full CMC interface uses its enumeration functions to retrieve configuration information from a Profile Container.

2.4 Object model

The CMC specification is based on a robust, object-oriented data model. In addition, a very general access method is defined by a group of functions oriented at managing these objects within the message service. These generic functions provide a very robust but simple method for creating and managing the object and object properties defined by the CMC specification.

The object model of the CMC specification is rather transparent to the user of the Simple CMC interface. This set of messaging-enabled functions was designed to simplify the access of message service functions. On the other hand, the Full CMC interface provides an enhanced set of messaging-enabled functions to access the robust characteristics of a message service and its object model.

This subclause provides an overview of the CMC objects, object classes and illustrates sample properties for each object.

2.4.1 Model components

Within the CMC specification, the object model contains objects, object classes, and properties. Figure 4 illustrates the CMC object model components. An object is a collection of properties. Objects are classified by their type or object class. A property is an attribute of the object.

2.4.1.1 Objects

Objects are identified by their session-specific object handle. The object handle encapsulates the session id. A handle for a new object is returned by the **cmc_open_object_handle()** function. The content information, that defines the particular messaging service object, can be added with the **cmc_add_properties()** function. An individual property can only exist once within an object. So, this same function can be used to update or modify the content information associated with a particular property. The **cmc_delete_properties()** function can be used to delete one or more individual properties from an object. The properties within an object can be listed with the **cmc_list_properties()** function. The content information for one or more properties can be read with the **cmc_read_properties()** function.

This Recommendation allows for multi-valued properties; multi-valued properties are used in conjunction with certain objects in this Recommendation.

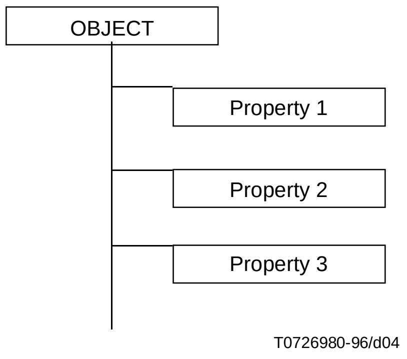


Figure 4/X.446 – Object model

Containers are a special kind of an object. They are a collection of not only properties but also other objects.

Once the object has been defined, it must be added to a particular container and committed to the persistent storage of that container with the **cmc_copy_object()** and **cmc_commit_object()** functions, respectively. An object can be deleted from a container object by the **cmc_delete_object()** function.

Enumeration of container objects are facilitated by the container cursor. A cursor is an implementation-specific construct that is used to sort and filter the elements of a container object. The cursor can also be used to facilitate the display of a "thumb" on a scroll bar, depicting the relative position within a container. The **cmc_open_cursor()** function is used to define a cursor. The cursor context is maintained by referencing the cursor by its opaque cursor handle. The cursor handle, as well as the object and session ids, are allocated by the message service. They need to be freed by the **cmc_free()** function when they are no longer needed. The relative position of the cursor can be read by the **cmc_read_cursor()** function. The relative position can be updated by the **cmc_update_cursor_position()** function. The cursor can also be updated to a position reflected by the relative position of an object within a container by the **cmc_update_cursor_position_with_seed()** function. The number of objects in a container that match the filter restrictions of a cursor can be listed by the **cmc_list_number_matched()** function.

A list of objects within the container associated with a cursor can be listed with the **cmc_list_objects()** function. The objects are referenced by the object handles returned by this function. The **cmc_copy_object_handle()** function can be used to make a copy of the reference to one of these objects.

It is possible that containers may not hold any objects such as an empty message container or an empty address book, for example.

Support for the nesting of containers is not mandatory in the Full CMC interface. Support for message container and address book nesting is not required. The appropriate error code for an implementation to return is CMC_E_UNSUPPORTED_ACTION. When a message with an embedded message is received, the implementation cannot guarantee that the embedded message will be passed on. Implementations must accept nested messages from the application. Nesting may not be preserved as such after the handle to the nested object is freed. Message objects created by the implementation may not use nesting. Implementations have the option to generate nested objects or not.

2.4.1.2 Object classes

Object classes are the types of objects defined by this Recommendation. Object classes in CMC contain properties and possibly other objects. Clause 3 describes the object classes and clause 5 describes the properties of each object class. The objects that another object class may contain are given by a containment hierarchy. This containment hierarchy is given in Figure 5. The containment hierarchy does not illustrate the full extent of recursion of CMC objects.

There is no explicit class hierarchy defined by this Recommendation. However, properties are duplicated among some object classes.

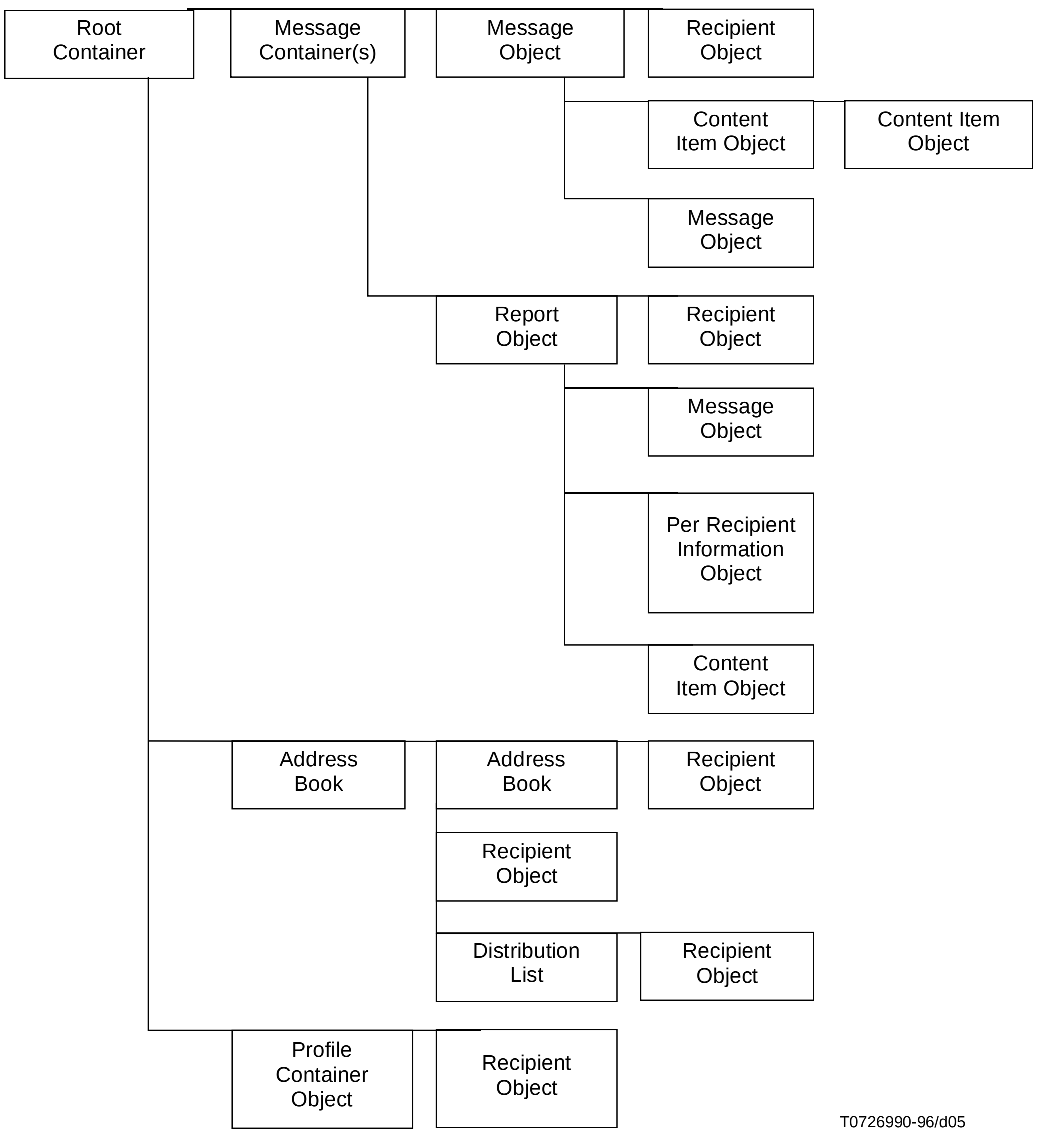


Figure 5/X.446 – CMC 2.0 Containment hierarchy

2.4.1.3 Object properties

Properties are attributes of a particular object. Properties define the object. They are represented by a unique name, or alternatively by an implementation specific identifier, a value type, and the value data or content information. A property is uniquely identified by an integer and a string name based on the formal public identifier of ISO 9070.

Some implementations may provide for user-defined properties. This capability allows for customization of the underlying service. User-defined properties are distinguished by their property name. A unique property identifier for a user-defined property is generated by a platform-specific mechanism. The **cmc_identifier_to_name()** function is provided to map between a property identifier and its associated property name. The **cmc_name_to_identifier()** function is provided to map between a property name and its associated property identifier. Both property identifiers and property names are provided by the service to permit access to the numerous properties by the most expedient method. The property identifier number space is divided into XAPIA-defined, implementation-defined, and user-defined numbers. User-defined numbers run the risk of possible duplication across implementations or versions.

Some object properties consist of large amounts of content information. For example, a multimedia message might have a megabyte of video or audio content. Stream functions have been added to the Full CMC interface to facilitate the reading and writing of this large content information. The content information is accessed in a manner similar to normal C Language file access. A property is opened for read or write stream operations by the **cmc_open_stream()** function. This function returns a stream handle that maintains the context of that stream in the session. This handle is allocated by the service and should be freed by the **cmc_free()** function when no longer needed. The **cmc_read_stream()** and **cmc_write_stream()** functions are used to read and write streamed content information, respectively. The **cmc_seek_stream()** function is used to go to a particular byte position within the stream. The stream is closed as a by-product of calling **cmc_free()**.

Different implementations may impart a different performance cost to read different properties. Properties with large amounts of content information may have a major cost. Properties with a small amount of content information or information that is readily available to the service may have a minor or no associated cost performance if read. The relative, implementation-specific cost performance for reading each property can be determined by the **cmc_read_property_costs()** function.

3 CMC object classes

3.2 CMC API object classes

The following subclauses define the CMC API object classes, provide the names of the classes, detail support requirements for the object classes, and how objects of each class are created, added, and modified.

Table 2 summarizes the object classes. The first column provides the name of the object class. The second column states whether the object class is mandatory or optional. The third column specifies whether or not the object class is read-only. A "no" in this column means that the object class can be added, deleted, or committed by **cmc_copy_object()**, **cmc_delete_object()**, or **cmc_commit_object()**, respectively, unless otherwise indicated by a star "*". The fourth column specifies how many instances the object is permitted to have. The last column states the creator of the object class as the implementation (I), the caller (C), or either (E).

The properties of each object class are given in clause 5 of this Recommendation.

Table 2/X.446 – Object Class summary

Object Class	M/O	Read-Only	Instances	Creator
Address Book	O	No	Any	E
Content Item	O	No	Any	E
Distribution List	O	No	Any	E
Message	M	No	Any	E
Message Container – Drafts	O	No	Any	C
Message Container – Filed	O	No	Any	C
Message Container – Inbox	O	No*	Zero or More	I
Message Container – Outbox	O	No*	One	I
Message Container – Sent, Deleted	O	No	One	I
Per Recipient Information	O	No	One or More	E
Profile Container	M	Yes	One	I
Recipient	M	No	Any	E
Report	O	No	Any	E
Root Container	O	No*	One	I

3.1.1 Address book

NAME

Address Book

C DECLARATION

```
#define CMC_OC_ADDRESS_BOOK                  \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Address Book//EN"
```

DESCRIPTION

The address book container class includes containers to hold recipient and distribution list objects. Address book containers may be nested, although implementations are not required to support the nesting of address book containers. A CMC implementation is not required to support address book containers. The subtypes of address book containers include global and personal. Address books hold recipient objects, distribution list objects, and optionally, other address books.

3.1.2 Content item

NAME

Content Item

C DECLARATION

```
#define CMC_OC_CONTENT_ITEM                  \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Content Item//EN"
```

DESCRIPTION

This object class identifies objects associated with the content of a message. It is used to represent attachments and notes, although no distinction is made between the two at the programming interface. Content item objects are typed by globally unique identifiers. Content items objects may be nested, although nesting is optional for support in an implementation.

Implementations may limit the number of content items per message or on the size of a content item. If a content item exceeds the number of content items permitted, a call to add the item may generate the error CMC_E_TOO_MANY_CONTENT_ITEMS. If the content item exceeds the size limit of the implementation, it may generate the error CMC_E_TEXT_TOO_LARGE.

3.1.3 Distribution list

NAME

Distribution List

C DECLARATION

```
#define CMC_OC_DISTRIBUTION_LIST             \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Distribution List//EN"
```

DESCRIPTION

The distribution list object class identifies objects that represent groups of recipient objects. Distribution lists contain recipient objects and, optionally, other distribution lists. The nesting of distribution lists may not be preserved after the handle to the distribution list has been freed. Implementations need not support distribution lists or the nesting of distribution lists. These distribution lists are identified by Recipient objects whose Type property is "group".

The use of the CMC to construct a distribution list does not imply that the messaging system must support the access to distribution lists whose members are administered by an address book or directory service disjoint from that supported by the CMC implementation.

3.1.4 Message

NAME

Message

C DECLARATION

```
#define CMC_OC_MESSAGE                  \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Message//EN"
```

DESCRIPTION

This object class identifies message objects that are vehicles for passing content information through a messaging service. These message objects may be mail and receipts. Message objects may be nested by applications and implementations must accept such messages. Nesting may not be preserved after the handle to the nested object is freed. Implementations need not support the nesting of messages. Message objects may contain recipient objects, content item objects, and nested message objects.

3.1.5 Message container

NAME

Message Container

C DECLARATION

```
#define CMC_OC_MESSAGE_CONTAINER              \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Message Container//EN"
```

DESCRIPTION

Message containers are a collection of message container properties, message objects, and possibly, other message containers. This container object also provides the enhancements for specialized collections such as an inbox, outbox, deletion folder, or user-defined message folders.

The message container object class provides a folder capability to hold message objects and possibly report objects and other message container objects. Message containers may be nested although implementations are not required to support nesting of message container objects. The subtypes of the message containers defined by CMC include drafts, deleted, sent, filed, inbox, and outbox.

3.1.5.1 Message container class: Drafts

The drafts message container holds messages that have been created but have not been sent. Support for the draft message container is optional.

3.1.5.2 Message container classes: Deleted, Sent

The deleted message container holds deleted messages. The sent message container contains messages that have been sent. Support of the sent and deleted message containers is optional.

3.1.5.3 Message container class: Filed

The sent message container contains filed messages. Support of the filed message containers is optional.

3.1.5.4 Message container class: Inbox

The message container class subtype inbox stores incoming messages. Support of an inbox is optional; there may be more than one inbox.

3.1.5.5 Message container class: Outbox

The outbox contains messages that are to be sent. Support of an outbox is optional; only one outbox is permitted.

3.1.6 Per Recipient Information

NAME

Per Recipient Information

C DECLARATION

```
#define CMC_OC_PER_RECIPIENT_INFORMATION          \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Per Recipient Information//EN"
```

DESCRIPTION

This object class identifies objects that report the delivery or non-delivery of a message for a single recipient. Objects of this class are contained in report objects. At least one of these objects must be present for the Report object. Support of this class is optional in general, but mandatory for implementations that support Report objects. Per Recipient Information objects may not be nested.

3.1.7 Profile Container

NAME

Profile Container

C DECLARATION

```
#define CMC_OC_PROFILE_CONTAINER              \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Profile Container//EN"
```

DESCRIPTION

The profile container class includes session context and configuration information. There is only one profile container. It exists underneath the root container object. The container object is created by the underlying messaging service, is read-only, and cannot be modified by the user. The contents of the profile container object are also created by the underlying messaging service, are read-only, and cannot be modified by the user. Support for the profile container object is mandatory for implementations conforming to this Recommendation.

The profile container contains a recipient object corresponding to the user logged on. If the implementation supports shared logon, then it may also contain additional recipient objects corresponding to the other logged-on users. This provides support for bulletin board or discussion forum capabilities. Additionally, the profile container object contains profile container attributes properties that correspond to individual session context or configuration attributes. There are properties defined for both the Simple CMC and Full CMC configuration attributes. The profile container properties are read-only.

3.1.8 Recipient

NAME

Recipient

C DECLARATION

```
#define CMC_OC_RECIPIENT                 \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Recipient//EN"
```

DESCRIPTION

The recipient object class identifies users within the messaging service. Recipient objects may include individuals and groups. The recipient type can be an individual, a group of recipients (e.g. distribution list), or an unknown type. An individual implementation may provide implementation specific properties for a recipient object.

3.1.9 Report

NAME

Report

C DECLARATION

```
#define CMC_OC_REPORT                    \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Report//EN"
```

DESCRIPTION

The report object class identifies objects that report the delivery status of a message. The objects in this class include delivery and non-delivery notifications. Certain message transfer systems (e.g. SMTP) may not support the generation of reports. Report objects may contain recipient objects, per recipient information objects, content item objects, and message objects.

3.1.10 Root Container

NAME

Root Container

C DECLARATION

```
#define CMC_OC_ROOT_CONTAINER                     \
    "-//XAPIA/CMC/OBJECT CLASS//NONSGML Root Container//EN"
```

DESCRIPTION

The root container class includes top-level containers for a user's messaging objects. There is only one type of root container and only one root container per user. It must be supported by CMC implementations. The root container contains message containers, a profile container, and optionally address book containers.

4 Data structures

This clause defines, and Table 3 lists, the data structures used in the CMC API.

Table 3/X.446 – CMC data structures

Data type name	Description
Attachment	Message attachment structure
Boolean	A value that indicates logical true or false
Buffer	Pointer to a data item
Callback Data Structures	Type definitions for a callback function data values
Counted String	String with an explicit length designation
Cursor Handle	Opaque handle for a container cursor
Cursor Restriction	Restriction for filtering the enumeration of objects within a container
Cursor Sort Key	Defines the sort order for elements enumerated by a cursor within a container
Dispatch Table	A structure containing pointers to the functions in a CMC implementation
Enumerated	Data type containing a value from an enumeration
Events	Data type for messaging service events
Extension	Extension structure
Flags	Container for flag bits
Guid	Globally unique identifier
ISO Date And Time	A date and time string value formatted in accordance with ISO 8601
Message	Message structure

Table 3/X.446 – CMC data structures *(concluded)*

Data type name	Description
Message Reference	Message Reference structure
Message Summary	Message Summary structure
Object Handle	Opaque handle for the CMC object
Object Identifier	Object Identifier structure
Opaque Data	A counted byte string of application specific data
Property	A piece of object content information
Identifier	Implementation specific, unique identifier
Name	Unique name
Recipient	Originator/recipient structure
Report	Status message for delivery, non-delivery, receipt, etc., notifications
Return Code	Return value indicating either that a function succeeded or why it failed
Session Id	Opaque handle for session
Stream Handle	Opaque handle for the property stream
String	Character string pointer
Time	Time structure
User Interface Id	User interface handle

4.1 Basic data types

Some data types are defined in terms of the following "intermediate data types", whose precise definitions in C are system-defined:

float32	The floating point number represented in 32 bits.
float64	The floating point number represented in 64 bits.
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. float              CMC_float32;
typedef system-defined, e.g. double             CMC_float64;
typedef system-defined, e.g. int                CMC_sint16;
typedef system-defined, e.g. long int           CMC_sint32;
typedef system-defined, e.g. unsigned char      CMC_uint8;
typedef system-defined, e.g. unsigned int       CMC_uint16;
typedef system-defined, e.g. unsigned long int  CMC_uint32;
```

4.2 Array data types

This Recommendation supports multi-valued properties using arrays of basic and non-basic data types. The array data types are defined as:

array_boolean	An array of Booleans.
array_buffer	An array of pointers to storage locations in memory.
array_counted_string	An array of strings with an explicit length designation.
array_enum	An array of enumerated data types.

array_extension	An array of extension data types.
array_float32	An array of floating point numbers represented in 32 bits.
array_float64	An array of floating point numbers represented in 64 bits.
array_guid	An array of globally unique identifiers.
array_iso_date_time	An array of ISO date and time data structures.
array_object_handle	An array of object handles.
array_opaque_data	An array of counted byte strings of application specific data.
array_return_code	An array of return codes.
array_sint16	An array of positive and negative integers representable in 16 bits.
array_sint32	An array of positive and negative integers representable in 32 bits.
array_string	An array of strings.
array_time	An array of time structures.
array_uint16	An array of non-negative integers representable in 16 bits.
array_uint32	An array of non-negative integers representable in 32 bits.

C DECLARATION

```
typedef struct CMC_TAG_ARRAY_BOOLEAN {
   CMC_uint32                    count;
   CMC_boolean                   *bits;
} CMC_array_boolean;

typedef struct CMC_TAG_ARRAY_BUFFER {
   CMC_uint32                    count;
   CMC_buffer                    *buffer;
} CMC_array_buffer;

typedef struct CMC_TAG_ARRAY_COUNTED_STRING {
   CMC_uint32                    count;
   CMC_counted_string            *string;
} CMC_array_counted_string;

typedef struct CMC_TAG_ARRAY_ENUM {
   CMC_uint32                    count;
   CMC_enum                      *set;
} CMC_array_enum;

typedef struct CMC_TAG_ARRAY_EXTENSION {
   CMC_uint32                    count;
   CMC_extension                 *extension;
} CMC_array_extension;

typedef struct CMC_TAG_ARRAY_FLOAT32 {
   CMC_uint32                    count;
   CMC_float32                   *number;
} CMC_array_float32;

typedef struct CMC_TAG_ARRAY_FLOAT64 {
   CMC_uint32                    count;
   CMC_float64                   *number;
} CMC_array_float64;

typedef struct CMC_TAG_ARRAY_GUID {
   CMC_uint32                    count;
   CMC_guid                      *guid;
} CMC_array_guid;

typedef struct CMC_TAG_ARRAY_ISO_DATE_TIME {
   CMC_uint32                    count;
   CMC_date_time                 *time;
} CMC_array_iso_date_time;

typedef struct CMC_TAG_ARRAY_OBJECT_HANDLE {
   CMC_uint32                    count;
   CMC_object_handle             *ohandles;
} CMC_array_object_handle;

typedef struct CMC_TAG_ARRAY_OPAQUE_DATA {
   CMC_uint32                    count;
   CMC_opaque_data               *data;
} CMC_array_opaque_data;
```

```
typedef struct CMC_TAG_ARRAY_RETURN_CODE {
   CMC_uint32                 count;
   CMC_return_code            *code;
} CMC_array_return_code;

typedef struct CMC_TAG_ARRAY_SINT16 {
   CMC_uint32                 count;
   CMC_sint16                 *number;
} CMC_array_sint16;

typedef struct CMC_TAG_ARRAY_SINT32{
   CMC_uint32                 count;
   CMC_sint32                 *number;
} CMC_array_sint32;

typedef struct CMC_TAG_ARRAY_STRING {
   CMC_uint32                 count;
   CMC_string                 *string;
} CMC_array_string;

typedef struct CMC_TAG_ARRAY_TIME {
   CMC_uint32                 count;
   CMC_time                   *time;
} CMC_array_time;

typedef struct CMC_TAG_ARRAY_UINT16 {
   CMC_uint32                 count;
   CMC_uint16                 *number;
} CMC_array_uint16;

typedef struct CMC_TAG_ARRAY_UINT32 {
   CMC_uint32                 count;
   CMC_uint32                 *number;
} CMC_array_uint32;
```

DESCRIPTION

A data value of these types includes a length identifier for the size of the array.

Support for multivalued properties is optional for implementations.

4.3 Attachment

NAME

Attachment – Type definition for a CMC message attachment structure.

C DECLARATION

```
typedef struct {
   CMC_string                    attach_title;
   CMC_object_identifier         attach_type;
   CMC_string                    attach_filename;
   CMC_flags                     attach_flags;
   CMC_extension                 *attach_extensions;
} CMC_attachment;
```

DESCRIPTION

A data value of this type is an attachment. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. An attachment has the following components:

1) *attach_title*: Optional title for attachment, e.g. original filename of attachment.

2) *attach_type*: Object identifier that specifies type of attachment. The format of the CMC_object_identifier is defined in 4.24. A NULL value designates an undefined attachment type.

   Two Object Identifiers have been predefined for use by applications and CMC implementations.

CMC_ATT_OID_BINARY	Data in file should be treated as binary data. This is the default.

CMC_ATT_OID_TEXT	Data in file should be treated as a text string. It should be assumed to be in the character set for the session on input and mapped to the character set for the session on output if possible.

3) *attach_filename*: Name of file where attachment content is located. The location of the file is implementation-dependent, but should ensure access by the calling application.

4) *attach_flags*: Bits for Boolean attributes. Unused bits must be clear.

   a) CMC_ATT_APP_OWNS_FILE

      Set: Indicates on output that the application now owns the file and is responsible for deleting it. This is ignored on input.

      Clear: Indicates on output that the CMC implementation owns the file and the application can only read the file.

   b) CMC_ATT_LAST_ELEMENT

      Set: Identifies the last structure in an array of such structures.

      Clear: This is not the last array element.

5) *attach_extensions*: Pointer to first element in array of per-attachment extensions. A value of NULL indicates that no extensions are present.

4.4 Boolean

NAME

Boolean – Type definition for a Boolean data value.

C DECLARATION

```
typedef   CMC_uint16   CMC_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 {CMC_FALSE}, and true is denoted by any other integer, although the symbolic constant {CMC_TRUE} refers to the integer one specifically.

4.5 Buffer

NAME

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

C DECLARATION

```
typedef        void * CMC_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.

4.6 Callback Data Structures

NAME

Callback Data Structures – Type definitions for a callback function data values.

C DECLARATION

```
typedef struct CMC_TAG_NEW_MESSAGE_CB_DATA {
   CMC_object_handle                    *available;
} CMC_new_message_callback_data;

typedef struct CMC_TAG_NEW_MESSAGE_CHECK_DATA {
   CMC_uint32                           number_containers;
   CMC_object_handle                    *containers;
} CMC_new_message_check_data;

typedef CMC_new_message_check_data      CMC_new_message_register_data;

typedef CMC_new_message_check_data      CMC_new_message_unregister_data;

typedef void (*CMC callback) (
   CMC_session_id                       session,
   CMC_event                            event,
   CMC_buffer                           callback_data,
   CMC_buffer                           register_data,
   CMC_extension                        *callback_extensions
);
```

DESCRIPTION

Callback procedures allow the service to inform applications that an event has occurred. All callback procedures are of type **cmc_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 **cmc_call_callbacks()** is called. Effectively, the CMC application running at the time of the callback invocation will be blocked until the callback returns. Responsiveness of the CMC application will be impacted if the callback function does not return quickly.

The callback function prototype components include the following:

– **session** – The opaque handle which represents a session with the messaging service.

– **event** – A bitmask of events. Exactly one bit will be set which indicates the event that occurred and how to interpret the **callback_data** argument. The following flags are defined:

  CMC_EVENT_NEW_MESSAGES

  See the Event data type for the definition of this flag.

– **callback_data** – A pointer to the callback data structure specific to the event.

– **register_data** – A pointer to the data structure passed when registering the callback in the **cmc_register()** function specific to the event.

– **callback_extensions** – A pointer to an array of CMC_extension structures for this callback function.

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

  The callback data structure is the mechanism that the messaging 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 register data structure. The callback data structure is allocated by the CMC implementation; the register data structure is allocated by the application.

  When a callback is unregistered, it may also specify unregistered data associated with the **cmc_unregister_event()** function to provide a context for the removal of registration (e.g. to await new messages on a smaller set of containers). The unregister data structure is allocated by the application. The valid types of arguments for each event are provided here.

This Recommendation, the application may also poll for events with the **cmc_check_event()** function. Events may have a context through a check data structure within which the **cmc_check_event()** function call is made. The check data structure is allocated by the application. The valid types of arguments for each event are provided here.

This Recommendation, the only event specified is CMC_EVENT_NEW_MESSAGES. An application may poll for new messages using **cmc_check_event()** or register callbacks to be called when new messages are received. If polling is used, it may be restricted to specific containers specified in the check data argument in the **cmc_check_event()** function with the structure CMC_TAG_new_message_check_data. The data elements in this structure include:

– **number_containers** – The number of container handles in the **containers** argument. If the event is independent of a container, this argument should be 0.

– **containers** – An array of handles of the containers to be checked for events. If the event is independent of a container, this argument should be NULL.

Upon return, **cmc_check_event()** returns the structure CMC_TAG_new_message_callback_data. The data elements in this data structure include:

– **available** – The handle of a message container (among the ones specified by the **containers** argument) to which the event corresponds. If no event occurred, the value is set to CMC_NULL_HANDLE.

When a callback is registered, the structure CMC_TAG_new_message_register_data is passed by reference in the register_data argument to the **cmc_register_event()** function. The data elements in this data structure are identical to the data elements in the CMC_TAG_new_message_check_data structure.

If a callback is registered and an event occurs, the structure CMC_TAG_new_message_callback_data is passed to the callback function. In addition, the CMC_TAG_new_message_register_data is passed to the callback function.

When a callback is unregistered, the structure CMC_TAG_new_message_unregister_data is passed by reference in the register_data argument to the cmc_unregister function. The data elements in this data structure include:

– **number_containers** – The number of container handles in the **containers** argument. If the event is independent of a container, this argument should be 0.

– **containers** – An array of handles of the containers for which the application is no longer interested in receiving notification of new messages. This array should be a subset of the handles specified in the containers argument in the register_data argument in the **cmc_register()** function. If the event is independent of a container, this argument should be NULL.

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

4.7 Counted String

NAME

Counted String – Type definition for a CMC counted string structure.

C DECLARATION

```
typedef struct {
   CMC_uint32      length;
   char            string[1];
} CMC_counted_string;
```

DESCRIPTION

A data value of this type is a counted string where the length of the string is explicitly specified preceding the character array. The string is not required to be null-terminated.

Support for a counted string data type is optional. Its purpose is to provide support for character sets in which embedded nulls are allowed.

See the CMC_string type for information about determining the character set.

The components of a counted string are:

1) *length*: Byte length of string that follows.

2) *string*: The characters that make up the string.

4.8 Cursor Handle

NAME

Cursor Handle – Type definition for a CMC cursor handle structure.

C DECLARATION

```
typedef system-defined, e.g. uint32    CMC_cursor_handle;
```

DESCRIPTION

A data value of this type is an opaque cursor handle. The CMC cursor handles are defined in an implementation-specific manner. The handle maintains a session context with a container cursor. The cursor facilitates the enumeration of objects within a container. It is also used to display a "thumb" on a scroll bar windowing control to illustrate the relative position within a collection of objects. Cursor handles cannot be copied.

4.9 Cursor Restriction

NAME

Cursor Restriction – Type definition for a CMC cursor restriction data type.

C DECLARATION

```
typedef struct CMC_TAG_RESTRICTION_AND {
   CMC_uint32                             count;
   struct CMC_TAG_RESTRICTION_CURSOR      *restriction;
} CMC_restriction_and;

typedef struct CMC_TAG_RESTRICTION_OR {
   CMC_uint32                             count;
   struct CMC_TAG_RESTRICTION_CURSOR      *restriction;
} CMC_restriction_or;

typedef struct CMC_TAG_RESTRICTION_NOT {
   CMC_uint32                             count;
   struct CMC_TAG_RESTRICTION_CURSOR      *restriction;
} CMC_restriction_not;

typedef struct CMC_TAG_RESTRICTION_STRING {
   CMC_enum                    exactness;
   CMC_id                      property;
   CMC_string                  string_constant;
} CMC_restriction_string;

typedef struct CMC_TAG_RESTRICTION_CONTENT {
   CMC_enum                    logical;
   CMC_id                      property;
   CMC_buffer                  property_value;
} CMC_restriction_content;

typedef struct CMC_TAG_RESTRICTION_COMPARISON {
   CMC_enum                    logical;
   CMC_id                      property1;
   CMC_id                      property2;
} CMC_restriction_comparison;

typedef struct CMC_TAG_RESTRICTION_BITTEST {
   CMC_uint32                  comparison;
   CMC_id                      property;
   CMC_uint32                  bitmask;
} CMC_restriction_bitmask;
```

```
typedef struct CMC_TAG_RESTRICTION_SIZE {
   CMC_enum                    logical;
   CMC_id                      property;
   CMC_uint32                  byte_size;
} CMC_restriction_size;

typedef struct CMC_TAG_RESTRICTION_EXIST {
   CMC_id                      property;
} CMC_restriction_exist;

typedef struct CMC_TAG_RESTRICTION_CURSOR {
   CMC_enum                    type;
   union {
      CMC_restriction_and        restriction_and;
      CMC_restriction_or         restriction_or;
      CMC_restriction_not        restriction_not;
      CMC_restriction_string     restriction_string;
      CMC_restriction_content    restriction_content;
      CMC_restriction_comparison restriction_comparison;
      CMC_restriction_bitmask    restriction_bittest;
      CMC_restriction_size       restriction_size;
      CMC_restriction_exist      restriction_exist;
   } cr;
   CMC_extension               *property_extensions;
} CMC_cursor_restriction;
```

DESCRIPTION

A data value of this type is a CMC cursor restriction. A cursor restriction is the definition of a filter on the enumeration of the contents of a container object. A cursor restriction has the following components:

1) *type*: The type of cursor restriction. The following valid restriction types are supported:

   CMC_RESTRICTION_AND
   CMC_RESTRICTION_OR
   CMC_RESTRICTION_NOT
   CMC_RESTRICTION_STRING
   CMC_RESTRICTION_CONTENT
   CMC_RESTRICTION_COMPARISON
   CMC_RESTRICTION_BITTEST
   CMC_RESTRICTION_SIZE
   CMC_RESTRICTION_EXIST

CMC_RESTRICTION_AND – Filters for the subrestrictions being all true.
CMC_RESTRICTION_OR – Filters for any one or more of the subrestrictions being true.
CMC_RESTRICTION_NOT – Filters for the subrestriction(s) being all false.
CMC_RESTRICTION_STRING – Filters for exactness in a string match with a property value.
CMC_RESTRICTION_CONTENT – Filters for a logical comparison of a constant and a property value.
CMC_RESTRICTION_COMPARISON – Filters for a logical comparison of two property values.
CMC_RESTRICTION_BITTEST – Filters for a property value matching the specified bitmask test.
CMC_RESTRICTION_EXIST – Filters for a property existing in the object or not.

2) *restriction*: Specifies the cursor restriction value.

3) *property_extensions*: Pointer to first element in array of property extensions.

The exactness structure element has the following valid string-exactness enumerated values:

   CMC_EXACTNESS_PRECISE
   CMC_EXACTNESS_STARTS_WITH
   CMC_EXACTNESS_MIXED_CASE

CMC_EXACTNESS_PRECISE – Property value matches exactly with the string constant.
CMC_EXACTNESS_STARTS_WITH – Property value starts with the string constant.
CMC_EXACTNESS_MIXED_CASE – Property value matches independent of the case.

The logical structure element has the following valid logical-operator enumerated values:

CMC_LOGICAL_LT
CMC_LOGICAL_LE
CMC_LOGICAL_EQ
CMC_LOGICAL_NE
CMC_LOGICAL_GT
CMC_LOGICAL_GE

CMC_LOGICAL_LT – Less than.
CMC_LOGICAL_LE – Less than or equal to.
CMC_LOGICAL_EQ – Equal to.
CMC_LOGICAL_NE – Not equal to.
CMC_LOGICAL_GT – Greater than.
CMC_LOGICAL_GE – Greater than or equal to.

The comparison structure element has the following valid bitmask-comparison enumerated values:

CMC_COMPARISON_OR
CMC_COMPARISON_AND

CMC_COMPARISON_OR – Property value is equal to the logical OR of the bitmask.
CMC_COMPARISON_AND – Property value is equal to the logical AND of the bitmask.

4.10 Cursor Sort Key

NAME

Cursor Sort Key – Type definition for a CMC cursor sort key data type.

C DECLARATION

```
typedef struct CMC_TAG_CURSOR_SORT_KEY {
   CMC_id                        property;
   CMC_enum                      order;
} CMC_cursor_sort_key;
```

DESCRIPTION

A data value of this type is a CMC cursor sort key. A cursor sort key defines the order in which elements of a container are sorted when enumerated by a cursor. An implementation may have an array of cursor sort keys. A cursor sort key has the following components:

1) *property*: Specifies the property on which the enumerated elements will be sorted.

2) *order*: Specifies the order in which the enumerated elements will be sorted. The valid sort orders are one of the following:

CMC_SORT_DEFAULT
CMC_SORT_ASCEND
CMC_SORT_DESCEND

CMC_SORT_DEFAULT – The elements of the container will not necessarily be sorted, but will be left in their default order. The result of this order is implementation specific.

CMC_SORT_ASCEND – Sorts the elements of the container object in ascending order. Objects that do not have the property listed by the sort key are placed last.

CMC_SORT_DESCEND – Sorts the elements of the container object in descending order. Objects that do not have the property listed by the sort key are placed last.

4.11 Dispatch Table

NAME

Dispatch Table – Type definition for a structure with pointers to the functions of a CMC implementation.

C DECLARATION

```
typedef struct {
CMC_extension                *dispatch_table_extensions;

/* SEND */
CMC_return_code
(*cmc_send)(
   CMC_session_id                session,
   CMC_message                   *message,
   CMC_flags                     send_flags,
   CMC_ui_id                     ui_id,
   CMC_extension                 *send_extensions
);

/* SEND DOCUMENT */
CMC_return_code
(*cmc_send_documents)(
   CMC_string                    recipient_addresses,
   CMC_string                    subject,
   CMC_string                    text_note,
   CMC_flags                     send_doc_flags,
   CMC_string                    file_paths,
   CMC_string                    file_names,
   CMC_string                    delimiter,
   CMC_ui_id                     ui_id
);

/* ACT ON */
CMC_return_code
(*cmc_act_on)(
   CMC_session_id                session,
   CMC_message_reference         *message_reference,
   CMC_enum                      operation,
   CMC_flags                     act_on_flags,
   CMC_ui_id                     ui_id,
   CMC_extension                 *act_on_extensions
);

/* LIST */
CMC_return_code
(*cmc_list)(
   CMC_session_id                session,
   CMC_string                    message_type,
   CMC_flags                     list_flags,
   CMC_message_reference         *seed,
   CMC_uint32                    *count,
   CMC_ui_id                     ui_id,
   CMC_message_summary           **result,
   CMC_extension                 *list_extensions
);

/* READ */
CMC_return_code
(*cmc_read)(
   CMC_session_id                session,
   CMC_message_reference         *message_reference,
   CMC_flags                     read_flags,
   CMC_message                   **message,
   CMC_ui_id                     ui_id,
   CMC_extension                 *read_extensions
);
```

```
/* LOOK UP */
CMC_return_code
(*cmc_look_up)(
   CMC_session_id                session,
   CMC_recipient                 *recipient_in,
   CMC_flags                     look_up_flags,
   CMC_ui_id                     ui_id,
   CMC_uint32                    *count,
   CMC_recipient                 **recipient_out,
   CMC_extension                 *look_up_extensions
);

/* FREE */
CMC_return_code
(*cmc_free)(
   CMC_buffer                    memory
);

/* LOGOFF */
CMC_return_code
(*cmc_logoff)(
   CMC_session_id                session,
   CMC_ui_id                     ui_id,
   CMC_flags                     logoff_flags,
   CMC_extension                 *logoff_extensions
);

/* LOGON */
CMC_return_code
(*cmc_logon)(
   CMC_string                    service,
   CMC_string                    user,
   CMC_string                    password,
   CMC_object_identifier         character_set,
   CMC_ui_id                     ui_id,
   CMC_uint16                    caller_cmc_version,
   CMC_flags                     logon_flags,
   CMC_session_id                *session,
   CMC_extension                 *logon_extensions
);

/* QUERY CONFIGURATION */
CMC_return_code
(*cmc_query_configuration)(
   CMC_session_id                session,
   CMC_enum                      item,
   CMC_buffer                    reference,
   CMC_extension                 *config_extensions
);

/* FULL CMC */

/* COPY OBJECT */
CMC_return_code
(*cmc_copy_object)(
   CMC_object_handle             container,
   CMC_object_handle             source_object,
   CMC_object_handle             *new_object,
   CMC_extension                 *copy_object_extensions
);

/* ADD PROPERTIES */
CMC_return_code
(*cmc_add_properties)(
   CMC_object_handle             object,
   CMC_uint32                    number_properties,
   CMC_property                  *properties,
   CMC_extension                 *add_properties_extensions
);

/* COMMIT OBJECT */
CMC_return_code
(*cmc_commit_object)(
   CMC_object_handle             source_object,
   CMC_extension                 *commit_object_extensions
);
```

```
/* COPY OBJECT HANDLE */
CMC_return_code
(*cmc_copy_object_handle)(
   CMC_object_handle              source_object,
   CMC_object_handle              *new_object,
   CMC_extension                  *copy_object_handle_extensions
);

/* CREATE DERIVED MESSAGE OBJECT */
CMC_return_code
(*cmc_create_derived_message_object)(
   CMC_object_handle              original_message,
   CMC_enum                       derived_action,
   CMC_boolean                    inherit_contents,
   CMC_object_handle              *derived_message,
   CMC_boolean                    modified_message,
   CMC_extension                  *create_derived_object_extensions
);

/* DELETE OBJECTS */
CMC_return_code
(*cmc_delete_objects)(
   CMC_uint32                     number_objects,
   CMC_object_handle              *object,
   CMC_extension                  *delete_objects_extensions
);

/* DELETE PROPERTIES */
CMC_return_code
(*cmc_delete_properties)(
   CMC_object_handle              object,
   CMC_uint32                     number_properties,
   CMC_id                         *property_ids,
   CMC_extension                  *delete_properties_extensions
);

/* GET ROOT HANDLE */
CMC_return_code
(*cmc_get_root_handle)(
   CMC_session_id                 session,
   CMC_object_handle              *root_object_handle,
   CMC_extension                  *get_root_handle_extensions
);

/* IDENTIFIER TO NAME */
CMC_return_code
(*cmc_identifier_to_name)(
   CMC_id                         identifier,
   CMC_name                       *name,
   CMC_extension                  *identifier_to_name_extensions
);

/* LIST CONTAINED PROPERTIES */
CMC_return_code
(*cmc_list_contained_properties)(
   CMC_cursor_handle              *cursor,
   CMC_sint32                     *number_object,
   CMC_sint32                     *number_properties,
   CMC_id                         *property_ids,
   CMC_property                   **properties,
   CMC_extension                  *list_contained_properties_extensions
);

/* LIST NUMBER MATCHED */
CMC_return_code
(*cmc_list_number_matched)(
   CMC_cursor_handle              *cursor,
   CMC_uint32                     *number_matches,
   CMC_extension                  *list_number_matched_extensions
);
```

```
/* LIST OBJECTS */
CMC_return_code
(*cmc_list_objects)(
   CMC_cursor_handle              *cursor,
   CMC_sint32                     *number_objects,
   CMC_object_handle              *objects,
   CMC_extension                  *list_objects_extensions
);

/* LIST PROPERTIES */
CMC_return_code
(*cmc_list_properties)(
   CMC_object_handle              *object,
   CMC_uint32                     *number_properties,
   CMC_id                         *property_ids,
   CMC_extension                  *list_properties_extensions
);

/* NAME TO IDENTIFIER */
CMC_return_code
(*cmc_name_to_identifier)(
   CMC_name                       name,
   CMC_id                         *identifier,
   CMC_extension                  *name_to_identifier_extensions
);

/* OPEN CURSOR */
CMC_return_code
(*cmc_open_cursor)(
   CMC_object_handle              object,
   CMC_cursor_restriction         *restrictions,
   CMC_uint32                     number_sort_orders,
   CMC_cursor_sort_key            *sort_keys,
   CMC_cursor_handle              *cursor,
   CMC_extension                  *open_cursor_extensions
);

/* OPEN OBJECT HANDLE */
CMC_return_code
(*cmc_open_object_handle)(
   CMC_session_id                 session,
   CMC_object_handle              *new_object,
   CMC_id                         object_class,
   CMC_extension                  *open_object_handle_extensions
);

/* READ CURSOR */
CMC_return_code
(*cmc_read_cursor)(
   CMC_cursor_handle              *cursor,
   CMC_uint32                     *position_numerator,
   CMC_uint32                     *position_denominator,
   CMC_extension                  *read_cursor_extensions
);

/* READ PROPERTIES */
CMC_return_code
(*cmc_read_properties)(
   CMC_object_handle              object,
   CMC_uint32                     *number_properties,
   CMC_id                         *property_ids,
   CMC_property                   **properties,
   CMC_extension                  *read_properties_extensions
);

/* READ PROPERTY COSTS */
CMC_return_code
(*cmc_read_property_costs)(
   CMC_object_handle              object,
   CMC_uint32                     *number_properties,
   CMC_id                         *property_ids,
   CMC_enum                       *costs,
   CMC_extension                  *read_property_costs_extensions
);
```

```
/* RESTORE OBJECT */
CMC_return_code
(*cmc_restore_object)(
   CMC_object_handle             container,
   CMC_string                    file_specification,
   CMC_object_handle             *restored_object,
   CMC_flags                     restore_flags,
   CMC_extension                 *restore_object_extensions
);

/* SAVE OBJECT */
CMC_return_code
(*cmc_save_object)(
   CMC_object_handle             object,
   CMC_string                    file_specification,
   CMC_flags                     save_flags,
   CMC_extension                 *save_object_extensions
);

/* SEND MESSAGE OBJECT */
CMC_return_code
(*cmc_send_message_object)(
   CMC_object_handle             message_to_send,
   CMC_extension                 *send_message_object_extensions
);

/* UPDATE CURSOR POSITION */
CMC_return_code
(*cmc_update_cursor_position)(
   CMC_cursor_handle             *cursor,
   CMC_uint32                    position_numerator,
   CMC_uint32                    position_denominator,
   CMC_extension                 *update_cursor_position_extensions
);

/* UPDATE CURSOR POSITION WITH SEED */
CMC_return_code
(*cmc_update_cursor_position_with_seed)(
   CMC_cursor_handle             cursor,
   CMC_object_handle             seed,
   CMC_extension                 *update_cursor_position_with_seed_extensions
);

/* CHECK EVENT */
CMC_return_code
(*cmc_check_event)(
   CMC_session_id                session,
   CMC_event                     event_type,
   CMC_uint32                    minimum_timeout,
   CMC_buffer                    check_event_data,
   CMC_buffer                    *callback_data,
   CMC_extension                 *check_event_extensions
);

/* REGISTER EVENT */
CMC_return_code
(*cmc_register_event)(
   CMC_session_id                session,
   CMC_event                     event_type,
   CMC_callback                  callback,
   CMC_buffer                    register_data,
   CMC_extension                 *register_event_extensions
);

/* UNREGISTER EVENT */
CMC_return_code
(*cmc_unregister_event)(
   CMC_session_id                session,
   CMC_flags                     event_type,
   CMC_callback                  callback,
   CMC_buffer                    unregister_data,
   CMC_extension                 *unregister_event_extensions
);
```

```
/* CALL CALLBACKS */
CMC_return_code
(*cmc_call_callbacks)(
   CMC_session_id              session,
   CMC_event                   event_type,
   CMC_extension               *call_callbacks_extensions
);

/* EXPORT STREAM */
CMC_return_code
(*cmc_export_stream)(
   CMC_stream_handle           stream,
   CMC_string                  file_specification,
   CMC_uint32                  count,
   CMC_flags                   export_flags,
   CMC_extension               *export_stream_extensions
);

/* IMPORT FILE TO STREAM */
CMC_return_code
(*cmc_import_file_to_stream)(
   CMC_stream_handle           stream,
   CMC_string                  file_specification,
   CMC_uint32                  file_offset,
   CMC_extension               *import_file_to_stream_extensions
);

/* OPEN STREAM */
CMC_return_code
(*cmc_open_stream)(
   CMC_object_handle           object,
   CMC_property                *property,
   CMC_enum                    operation,
   CMC_stream_handle           **stream,
   CMC_extension               *open_stream_extensions
);

/* READ STREAM */
CMC_return_code
(*cmc_read_stream)(
   CMC_stream_handle           stream,
   CMC_uint32                  *count,
   CMC_buffer                  content_information,
   CMC_extension               *read_stream_extensions
);

/* SEEK STREAM */
CMC_return_code
(*cmc_seek_stream)(
   CMC_stream_handle           stream,
   CMC_enum                    operation,
   CMC_uint32                  *location,
   CMC_extension               *seek_stream_extensions
);

/* WRITE STREAM */
CMC_return_code
(*cmc_write_stream)(
   CMC_stream_handle           *stream,
   CMC_uint32                  *count,
   CMC_buffer                  *content_information,
   CMC_extension               *write_stream_extensions
);

/* GET LAST ERROR */
CMC_return_code
(*cmc_get_last_error)(
   CMC_session_id              session,
   CMC_object_handle           objRef,
   CMC_string                  **error_buffer,
   CMC_extension               *get_last_error_extensions
);

} CMC_dispatch_table;
```

```
/* BIND IMPLEMENTATION     */
CMC_return_code
cmc_bind_implementation (
   CMC_guid                         implementation_name,
   CMC_dispatch_table               **dispatch_table,
   CMC_extension                    *cmc_bind_extensions
);
/* UNBIND IMPLEMENTATION */
CMC_return_code
cmc_unbind_implementation (
   CMC_guid                         implementation_name,
   CMC_extension                    *cmc_unbind_implementation_extensions
);
```

DESCRIPTION

A data value of this data type is a dispatch table for a CMC implementation. The dispatch table includes an entry for each function in a CMC implementation. Refer to the examples in C.2 (bind.c and cmc_bind.c) on the use of the dispatch table.

4.12 Enumerated

NAME

Enumerated – Type definition for an enumerated data value.

C DECLARATION

```
typedef  CMC_sint32  CMC_enum;
```

DESCRIPTION

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

4.13 Events

NAME

Events – Type definition for a CMC event.

C DECLARATION

```
typedef   CMC_uint32      CMC_event;
```

DESCRIPTION

A data value of this type contains 32 event bits. Undocumented events are reserved. Event bits set to zero are referred to as "clear". Event bits set non-zero are referred to as "set". Unspecified event bits should always be clear.

Set:	New messages have arrived in a message container.
Clear:	No new messages have arrived in a message container.

In this Recommendation, the only valid event type is:

CMC_EVENT_NEW_MESSAGES

4.14 Extension

NAME

Extension – Type definition for a CMC extension structure.

C DECLARATION

```
typedef struct {
   CMC_uint32       item_code;
   CMC_uint32       item_data;
   CMC_buffer       item_reference;
   CMC_flags             extension_flags;
} CMC_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 CMC function calls *and* CMC 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 CMC service. For example, some **cmc_act_on()** implementations might allow saving of partially completed messages to the inbox for later reading and sending by using the CMC_X_COM_SAVE_MESSAGE extension to pass in the message structure and receive back the resulting message reference. For the complete list of common message extensions specified in this Recommendation, see 4.11 and 4.14.

For CMC extension arrays that may contain output extension storage allocated by the CMC service, callers must use **cmc_free()** to free the pointer returned in the item_reference field. These structures are identified by the output flag CMC_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. For example, the message_extensions array resulting from **cmc_read()** is implicitly freed when **cmc_free()** is called for the enclosing message structure.

An extension has the following components:

1) *item_code*: A code that uniquely identifies this extension.

2) *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.

3) *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.

4) *extension flags*: Bits for Boolean attributes. The upper 16 bits are reserved for definition by the CMC specification. Any unused bits of these must be clear. The lower 16 bits of flags are reserved for definition by the extension.

   a) CMC_EXT_REQUIRED

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

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

   b) CMC_EXT_OUTPUT

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

   c) CMC_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.

4.15 Flags

NAME

Flags – Type definition for a CMC flag.

C DECLARATION

```
typedef   CMC_uint32       CMC_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.

4.16 GUID

NAME

GUID – Type definition for a CMC globally unique identifier (GUID) structure.

C DECLARATION

```
typedef CMC_string     CMC_guid
```

DESCRIPTION

A data value of this type is a globally unique identifier. The string is formatted according to the formal public identifier text of ISO 9070 to guarantee uniqueness. The CMC GUIDs have the following format:

–//XAPIA/CMC/*name type*/NONSGML *name*//EN

where *name type* is the type of name and *name* is the name of the object to which the GUID is being assigned. For example, the object class CONTENT ITEM is:

```
–//XAPIA/CMC/OBJECT CLASS//NONSGML Content Item//EN
```

Some of the CMC GUID values may be defined in terms of an ISO/OSI Object Identifier (OID). The OID can be encapsulated into an ISO 9070 formal public identifier. The FPI encapsulation is accomplished as follows:

```
–//XAPIA/CMC/OID//NONSGML <oid>//EN
```

where **<oid>** is the numeric form of the OSI OID as defined by the object identifier data type in 4.24.

4.17 Identifier

NAME

Identifier – Type definition for an implementation specific, unique identifier.

C DECLARATION

```
typedef system-defined, e.g. uint32     CMC_id;
```

DESCRIPTION

A data value of this type is an implementation-specific, unique identifier. This data type is used for locally unique identifiers such as property id and object class id.

4.18 ISO Date and Time

NAME

ISO Date and Time – Type definition for an ISO 8601 formatted date and time data value.

C DECLARATION

```
typedef CMC_string     CMC_date_time;
```

DESCRIPTION

A data value of this data type is a date and time value consistent with the combined date and time of the day representation of ISO 8601. The format of this data type supports the time of the day represented as either local time, or the clock time in public use locally; Coordinated Universal Time (UTC), or the time scale maintained by the Bureau International de l'Heure that forms the basis of a coordinate dissemination of standard frequencies and time signals; or the local time difference between UTC.

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. If the time is in UTC, the character [Z] is used as time-zone designator for UTC. If the time-zone designator is absent, the time is in local time. 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, and [ss] is the seconds past the minute string.

For local time as the difference from UTC, the date and time is represented by the string ccyymmddThhmmss+hhmm, ccyymmddThhmmss+hh, ccyymmddThhmmss–hhmm, or ccyymmddThhmmss–hh, 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, and [ss] is the seconds past the minute string. The time-zone designator is absent and the date and time string is concatenated with the hour and minute or hour offset from UTC. The difference between local time and UTC is expressed in hours and minutes, or hours only independently of the precision of the local time expression associated with it. It is expressed as positive (i.e. with the leading plus sign [+]) if the local time is ahead of UTC and as negative (i.e. with the leading minus sign [–]) if it is behind UTC. For example, 19850414T152746+0100 would be April 14, 1985 and the time of 27 minutes 46 seconds past 15 hours locally in a location normally one hour ahead of UTC. The string 19850414T152746–05 would be April 14, 1985 and the time of 27 minutes 46 seconds past 15 hours locally in a location normally five hours behind UTC.

1) **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 19850414.

2) **time** – The time of the day, expressed as either the local time, equivalent Coordinated Universal Time (UTC), or local time difference from UTC. The time format is the complete representation, basic format, as defined in ISO 8601, clauses 5.3.3 and 5.3.3.1. For example, UTC time 20 minutes and 30 seconds past 23 hours would be represented by the string 232030Z. The local time 10 minutes and 15 seconds past 12 hours would be represented by the string 121510. The same local time as the difference from UTC would be represented by the string 121510–06 or 121510–0600 if local time was six hours behind UTC.

4.19 Message

NAME

Message – Type definition for a CMC message structure.

C DECLARATION

```
typedef struct {
    CMC_message_reference    *message_reference;
    CMC_string                  message_type;
    CMC_string                  subject;
    CMC_time                        time_sent;
    CMC_string                  text_note;
    CMC_recipient                   *recipients;
    CMC_attachment              *attachments;
    CMC_flags                       message_flags;
    CMC_extension                   *message_extensions;
} CMC_message;
```

DESCRIPTION

A data value of this type is a message. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. A message has the following components:

1) *message_reference*: Identifies the message. The message reference is unique within a mailbox.

2) *message_type*: String that identifies the type of the message. Three different string identifiers may be used:

   a) Object Identifiers – Used for types identified by object identifiers as defined in Recommendation X.208.

   b) CMC Registered Values – Used for types defined in this Recommendation.

   c) Bilaterally Defined Values – Used for types that are unregistered.

NOTE – Bilaterally defined values are not ensured to be unique.

The format of each type is given below. White space can be any combination of tabs or spaces. "*" indicates one or more of the denoted token (separated by white space) is valid. Quoted strings are case insensitive.

```
message_type_value      ::= oid | cmc_reg | bilat_def
oid                     ::= "OID:" object_identifier
cmc_reg                 ::= "CMC:" cmc_registered_value
bilat_def               ::= "BLT:" string
object_identifier       ::= object_id_component*
object_id_component     ::= integer
cmc_registered_value    ::= "IPM" | "IP RN" | "IP NRN" | "DR" | "NDR" | "REPORT"
```

These registered values are defined as follows:

"CMC: IPM"	Interpersonal message: An interpersonal message is a memo-like message containing a recipient list, an optional subject, an optional text note, and zero or more attachments. The "Message" structure is optimized to accommodate a message of type IPM.
"CMC: IP RN"	Receipt notification for an interpersonal message: A receipt notification indicates that a message has been read by the recipient.
"CMC: IP NRN"	Non-receipt notification for an interpersonal message: A non-receipt notification indicates that a message has been removed from the recipient’s mailbox without being read (for instance, the message has been discarded by the user or the service or it has been auto-forwarded to another recipient).
"CMC: DR"	Delivery report: A delivery report indicates that the service was able to deliver a message to the recipient.
"CMC: NDR"	Non-delivery report: A non-delivery report indicates that the service was not able to deliver a message to the recipient.
"CMC: REPORT"	Both delivery and non-delivery reports when the original message is destined for multiple recipients: This is to indicate that the underlying messaging service is able to deliver the message to some recipients but not to the others.

The format of these message types within the structures defined depend upon the messaging protocols that have been employed by the messaging service. Often non-IPM messages take the form of a program-generated message, which follows a memo-like format (similar to that of an IPM) but whose purpose is to convey information about a previously sent message.

NOTE – These message types correspond to X.400 message types; however, the types may be used with non-X.400 messaging services. Thus, these CMC message types are meant to apply generically and not specifically to X.400.

Example valid identifiers are:

```
OID: 1 2 840 113556 3 2 850
CMC: IPM
BLT: my special message type
```

A canonical form of these types is also defined to allow an application to easily compare these strings. The CMC implementation will always return the canonical form. In the canonical form:

1) all white space is converted to a single space, and all tokens will be separated by a white space;

2) the type identifiers (i.e. OID, CMC, BLT) are converted to upper case.

Some CMC implementations will only support the interpersonal message type (CMC: IPM). Other types of messages may be treated as IPM messages or may generate an error on those implementations.

It is undefined what the implementation will do with strings that are not in one of these formats.

3) *subject*: Message's subject string.

4) *time_sent*: Date/time message was sent (submitted).

5) *text_note*: Message's text note string. If the value is NULL, there is no text note. If the CMC_TEXT_NOTE_AS_FILE flag is set, the text note is in the first attachment.

   The format of the text note, regardless of whether it is passed in memory or in a file, is a sequence of paragraphs, with the appropriate line terminator for the platform (CR for Macintosh, LF for Unix, CR/LF for DOS and Windows, etc.) terminating each paragraph. Long lines (paragraphs) may be word wrapped by the CMC implementation.

NOTE – There is no guaranteed fidelity (e.g. a long paragraph may be returned by the CMC read functions as a series of shorter paragraphs).

6) *recipients*: Pointer to first element in array of recipients of the message.

7) *attachments*: Pointer to first element in array of attachments for the message.

8) *message_flags*: Bits for Boolean attributes. Unused bits must be clear.

   a) CMC_MSG_READ

      Set: Message has been read.

      Clear: Message has not been read.

   b) CMC_MSG_TEXT_NOTE_AS_FILE

      Set: Text-note field is ignored and the text_note text is contained in the file referred to by the first attachment.

      Clear: Text_note text is contained in the text note string.

   c) CMC_MSG_UNSENT

      Set: Message has not been sent (i.e. it is a draft). This type of message can be created with the CMC_X_COM_SAVE_MESSAGE extension.

      Clear: Message has been sent.

   d) CMC_MSG_LAST_ELEMENT

      Set: Identifies the last structure in an array of such structures.

      Clear: This is not the last array element.

9) *message_extension*: Pointer to first element in array of per-message extensions.

4.20 Message Reference

NAME

Message Reference – Type definition for a CMC message reference structure.

C DECLARATION

```
typedef   CMC_counted_string  CMC_message_reference;
```

DESCRIPTION

A data value of this type is a counted string that is the message handle used by the mailbox. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. A Message Reference is only guaranteed to be valid for the life of the session and has no guaranteed correspondence to any message identifier used by the underlying messaging system. Within the session lifetime, it may be copied by the application program.

4.21 Message Summary

NAME

Message Summary – Type definition for a CMC message summary structure.

C DECLARATION

```
typedef struct {
    CMC_message_reference    *message_reference;
    CMC_string                  message_type;
    CMC_string                  subject;
    CMC_time                        time_sent;
    CMC_uint32                  byte_length;
    CMC_recipient                   *originator;
    CMC_flags                       summary_flags;
    CMC_extension                   *message_summary_extensions;
} CMC_message_summary;
```

DESCRIPTION

A data value of this type is a message summary. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. A message summary has the following components:

1) *message_reference*: See definition in Message Structure.

2) *message_type*: See definition in Message Structure.

3) *subject*: See definition in Message Structure.

4) *time_sent*: See definition in Message Structure.

5) *byte_length*: Message size. The value should include all associated features of the message – attachments, envelope and heading fields, etc. Implementations may return an approximate value or the constant CMC_LENGTH_UNKNOWN if the length is unknown or unavailable.

6) *originator*: Message originator.

7) *summary_flags*: Bits for Boolean attributes. Unused bits must be clear.

    a) CMC_SUM_READ

        Set: Message has been read.

        Clear: Message has not been read.

    b) CMC_SUM_UNSENT

        Set: Message has not been sent (i.e. it is a draft).

        Clear: Message has been sent.

    c) CMC_SUM_LAST_ELEMENT

        Set: Identifies the last structure in an array of such structures.

        Clear: This is not the last array element.

    d) CMC_SUM_HAS_ATTACHMENTS

        Set: Message has attachments.

        Clear: Message has no attachments.

8) *message_summary_extensions*: Pointer to first element in array of per-message-summary extensions.

4.22 Name

NAME

Name – Type definition for a unique CMC 2.0 name.

C DECLARATION

```
typedef CMC_string     CMC_name;
```

DESCRIPTION

A data value of this type is a unique name. The string is formatted according to the formal public identifier text of ISO 9070 to guarantee uniqueness. The CMC names have the following format:

```
–//XAPIA/CMC/name type//NONSGML name//EN
```

where *name type* is the type of name and *name* is the name of the object. For example, the object class CONTENT ITEM is:

```
–//XAPIA/CMC/OBJECT CLASS//NONSGML Content Item//EN
```

4.23 Object Handle

NAME

Object Handle – Type definition for a CMC object handle structure.

C DECLARATION

```
typedef system-defined, e.g. uint32     CMC_object_handle;
```

DESCRIPTION

A data value of this type is an opaque object handle. The CMC object handles are unique to the message service. The handles are persistent for the duration of the session or until they are destroyed. The handle provides the context to a CMC object. The handle encapsulates the session id. To copy an object handle, use the **cmc_copy_object_handle()** function.

The notion of "no handle" needs to be stored in an object handle. In this case, the constant CMC_NULL_HANDLE is used and is system-defined.

4.24 Object Identifier

NAME

Object Identifier – Type definition for a CMC object identifier structure.

C DECLARATION

```
typedef   CMC_string      CMC_object_identifier;
```

DESCRIPTION

A data value of this type is an object identifier as defined in Recommendation X.208. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. It is globally unambiguous. Its syntax as used in this Recommendation shall match the Number form in Recommendation X.208. This syntax is:

```
object_identifier          ::= object_id_component*
object_id_component        ::= integer
```

An example of an object identifier is:

1 2 840 113556 3 2 850

NOTE – The format of the object_identifier string is the same as the one used in the OID message type.

4.25 Opaque Data

NAME

Opaque Data – Type definition for an opaque data value.

C DECLARATION

```
typedef struct CMC_TAG_OPAQUE_DATA {
    CMC_size                size;
    CMC_byte                *data;
} CMC_opaque_data;
```

DESCRIPTION

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

1) **size** – Specifies the number of 8-bit, bytes of opaque data pointed to by **data**.

2) **data** – A pointer to an array of 8-bit values. There is no explicit semantics to this data.

4.26 Property

NAME

Property – Type definition for a CMC property data type.

C DECLARATION

```
typedef struct CMC_TAG_PROPERTY{
    CMC_id                          property id;
    CMC_enum                            type;
    union {
        CMC_boolean                 CMC_pv_boolean;
        CMC_byte                        CMC_pv_byte;
        CMC_buffer                  CMC_pv_buffer;
        CMC_counted_string          CMC_pv_counted_string;
        CMC_enum                        CMC_pv_enum;
        CMC_extension                   CMC_pv_extension;
        CMC_float32                 CMC_pv_float32;
        CMC_float64                 CMC_pv_float64;
        CMC_flags                       CMC_pv_flags;
        CMC_guid                        CMC_pv_guid;
        CMC_iso_date_time       CMC_pv_iso_date_time;
        CMC_object_handle           CMC_pv_object_handle;
        CMC_opaque_data             CMC_pv_opaque_data;
        CMC_return_code             CMC_pv_return_code;
        CMC_sint16                  CMC_pv_sint16;
        CMC_sint32                  CMC_pv_sint32;
        CMC_string                  CMC_pv_string;
        CMC_time                        CMC_pv_time;
        CMC_uint16                  CMC_pv_uint16;
        CMC_uint32                  CMC_pv_uint32;
        CMC_array_boolean           CMC_pv_array_boolean;
        CMC_array_buffer            CMC_pv_array_buffer;
        CMC_array_counted_string CMC_pv_array_counted_string;
        CMC_array_enum              CMC_pv_array_enum;
        CMC_array_extension         CMC_pv_array_extension;
        CMC_array_float32       CMC_pv_array_float32;
        CMC_array_float64       CMC_pv_array_float64;
        CMC_array_guid              CMC_pv_array_guid;
        CMC_array_iso_date_time CMC_pv_array_iso_date_time;
        CMC_array_object_handle     CMC_pv_array_object_handle;
        CMC_array_opaque_data       CMC_pv_array_opaque_data;
        CMC_array_return_code       CMC_pv_array_return_code;
        CMC_array_sint16            CMC_pv_array_sint16;
        CMC_array_sint32            CMC_pv_array_sint32;
        CMC_array_string            CMC_pv_array_string;
        CMC_array_time              CMC_pv_array_time;
        CMC_array_uint16            CMC_pv_array_uint16;
        CMC_array_uint32            CMC_pv_array_uint32;
    } value
    } CMC_property;
```

DESCRIPTION

A data value of this type is a CMC_array property. A property is the method for specifying CMC_array specific content information. A property has the following components:

1) *id*: Uniquely identifies the property.

2) *type*: Specifies the data type for the property.

3) *value*: Defines the value for the property.

4) *property_extensions*: Pointer to first element in array of property extensions.

4.27 Recipient

NAME

Recipient – Type definition for originator/recipient structure.

C DECLARATION

```
typedef struct {
    CMC_string        name;
    CMC_enum          name_type;
    CMC_string        address;
    CMC_enum          role;
    CMC_flags         recip_flags;
    CMC_extension     *recip_extensions;
} CMC_recipient;
```

DESCRIPTION

A data value of this type is an originator or recipient. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. This structure has the following components:

1) *name*: Recipient display name. Whether to interpret the name as an individual first, then as a group, if such an individual is not found, or vice versa, is left up to the implementation when resolving the name to an address.

2) *name_type*: Recipient type, enumerated:

CMC_TYPE_UNKNOWN (= 0)	Unknown recipient type.
CMC_TYPE_INDIVIDUAL	Recipient is an individual.
CMC_TYPE_GROUP	Name is a group of recipients.

NOTE – This is meaningful only if name is present. It is set by the implementation on output. On input it can be used as a hint to optimize resolution of the name.

3) *address*: Recipient address which is acceptable to the underlying messaging service. The format of the address string is not defined by this Recommendation. It is intended to accommodate any string notation(s) supported by a given implementation, as configured at a given installation. End users should consult the manager of their local service to discover what string notation(s) are supported at their installation.

4) *role*: Role of recipient, enumerated:

CMC_ROLE_TO	TO (primary) recipient.
CMC_ROLE_CC	CC recipient.
CMC_ROLE_BCC	BCC recipient.
CMC_ROLE_ORIGINATOR	Originator of message.
CMC_ROLE_AUTHORIZING_USER	Authorizing user of message.
CMC_ROLE_REPLY_TO	Recipient to receive replies.

A CC recipient may (silently) be converted to a TO recipient if the underlying messaging service cannot support CC recipients. Services that cannot support BCCs should reject messages containing them. For the same recipient to be present with more than one role, multiple recipient entries, differing in role, are required.

The CMC implementation should return the recipient array in the following order on output. The originator should be the first element in the array, followed by the REPLY TO, TO, CC, and BCC recipients grouped together in that order. The authorizing user, if one exists, should be the final recipient in the array. There is no ordering required on input.

5) *recip_flags*: Bits for Boolean attributes. Unused bits must be clear.

   a) CMC_RECIP_IGNORE

      Set: Ignore this recipient (useful for re-using an incoming message's recipient list for a reply).

      Clear: Do not ignore this recipient.

   b) CMC_RECIP_LIST_TRUNCATED

      Set: Indicates that not all recipient structures requested were returned by the system. This is only used on the **cmc_look_up()** function when the complete list of recipients matching the search name could not be returned. This flag will only be set in the last structure in the array.

      Clear: The complete recipient array was returned.

   c) CMC_RECIP_LAST_ELEMENT

      Set: Identifies the last structure in an array of such structures.

      Clear: This is not the last array element.

6) *recip_extensions*: Pointer to first element in array of per-recipient extensions.

4.28 Report

NAME

Report – Type definition for combination of report and non-delivery report structure.

C DECLARATION

```
typedef struct {
    CMC_recipient            *msg_recipient;
    CMC_enum                 report_type;
    CMC_time                 delivered_time;
    CMC_uint32           reason_code;
    CMC_flags                report_flags;
} CMC_report;
```

DESCRIPTION

A data value of this type is a report, non-delivery report, or both. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. A report has the following components:

1) *report_type*: Enumerated value that identifies the type of report. The report type can be:

   CMC_X400_DR ((CMC_enum) 0)

   CMC_X400_NDR ((CMC_enum) 1)

2) *delivered_time*: Date/time the original message was delivered to the recipient. It is NULL for CMC_X400_NDR, or report delivered time for CMC_X400_DR.

3) *reason_code*: The reason for the non-delivery of a message. The value is ZERO for CMC_X400_DR, or, the following value for CMC_X400_NDR:

   reason_code.<higher order 16 bits> = X.411.NonDeliveryReasonCode.

   reason_code.<lower order 16 bits> = X.411.NonDeliveryDiagnosticCode.

4) *report_flags*: Bits for Boolean attributes. Unused bits must be clear.

– CMC_REPORT_LAST_ELEMENT

Set: Identifies the last structure in an array of such structures.

Clear: This is not the last array element.

NOTE – CMC defines specific message types for delivery reports ("CMC:DR") and non-delivery reports ("CMC:NDR") which can be acted on independently since they are viewed as separate messages. In Recommendation X.400, both the delivery and non-delivery information is conveyed in a generic report information base. It is possible that an X.400 report contains delivery reports for some recipients and non-delivery reports for another recipient when a message is destined for multiple recipients of the same MTA. This does not map well to the CMC:DR or CMC:NDR on output (X.400 to CMC) because Recommendation X.400 does not view them nor stored them as separate information base and therefore cannot be acted on individually. Thus, a new message type "CMC: REPORT" is added to handle the X.400 report requirements.

4.29 Return Code

NAME

Return Code – Type definition for a value returned from all CMC functions.

C DECLARATION

```
typedef   CMC_uint32       CMC_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 Recommendation. 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 CMC error.

Errors may be resolved within the scope of a CMC call using, for example, dialogues available through the user interface. If a dialogue is invoked to resolve the error, but the error remains unresolved after the dialogue has ended, the bit flag defined in CMC_ERROR_UI_DISPLAYED is set in the error to indicate that the error has already been displayed to the user.

4.30 Session Id

NAME

Session Id – Type definition for a CMC session id.

C DECLARATION

```
typedef   system-defined, e.g. uint32CMC_session_id;
```

DESCRIPTION

Opaque session id. The context identified by the session id contains per-session information such as the character set in use and handles for any open session(s) with underlying messaging service(s). The CMC_session_id is created by the CMC Logon function and destroyed by the CMC Logoff function.

See B.2.4 for the definition for a specific platform.

4.31 Stream Handle

NAME

Stream Handle – Type definition for a CMC stream handle structure.

C DECLARATION

```
typedef system-defined, e.g. uint32 CMC_stream_handle;
```

DESCRIPTION

A data value of this type is an opaque stream handle. The CMC stream handles are unique to the message service. The handles are persistent for the duration of the session or until they are destroyed. The handle provides the context to a stream of content information. The stream encapsulates the session id and object handles. Stream handles cannot be copied.

4.32 String

NAME

String – Type definition for a CMC character string.

C DECLARATION

```
typedef   cmc_string*      CMC_string;
```

DESCRIPTION

A data value of this type is a string. The character array pointed to is interpreted as a null-terminated array of character 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.

If an application wishes to use counted strings instead of null-terminated and the CMC implementation supports it, the application will set the CMC_COUNTED_STRING_TYPE flag when logging into the session. The data pointed to by CMC_string will then be assumed to be in the data format of CMC_counted_string. If implicit logon is done with a function, this flag must be set in the flags parameter.

To determine the character set of characters in the string, the CMC 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.

4.33 Time

NAME

Time – Type definition for a CMC time structure.

C DECLARATION

```
typedef struct {
    CMC_sint8       second;
    CMC_sint8       minute;
    CMC_sint8       hour;
    CMC_sint8       day;
    CMC_sint8       month;
    CMC_sint8       year;
    CMC_sint8       isdst;
    CMC_sint16      tmzone;
} CMC_time;
```

DESCRIPTION

A data value of this type is a time value. This data structure is included to provide support for CMC 1.0 and Simple CMC implementations. A time value has the following components:

1) *second*: Seconds; range 0..59.

2) *minute*: Minutes; range 0..59.

3)	*hour*:	Hours since midnight; range 0..23.
4)	*day*:	Day of the month; range 1..31.
5)	*month*:	Months since January; range 0..11.
6)	*year*:	Years since 1900.
7)	*isdst*:	Daylight savings time flag; non-zero implies daylight savings.
8)	*tmzone*:	Time zone, in minutes relative to Greenwich Mean Time. The defined value, CMC_NO_TIMEZONE, indicates that time zone is not available.

All time values are in the appropriate local time. For example, the time_sent field in the CMC_message and CMC_message_summary structures is in the local time of the sender.

NOTE – If the tmzone field is set to any value other than CMC_NO_TIMEZONE, then the time value can be converted into the local time of the caller, although the actual conversion functionality falls outside the scope of CMC.

4.34 User Interface Identifier

NAME

User Interface Identifier – Type definition for a CMC user interface handle.

C DECLARATION

```
typedef   system-defined, e.g. uint32CMC_ui_id;
```

DESCRIPTION

Value used for passing user interface information to CMC functions. For example, in a windows-based environment this would be the parent-window handle for the calling application.

A value of NULL is always valid, with the appropriate default behaviour defined by the implementation.

NOTE – CMC implementations are not required to provide UI, and providing a user interface for one feature does not necessarily imply that a user interface is available for all features of CMC.

See B.2.4 for the definition for a specific platform.

5 Object properties

This clause defines the object properties for object classes of the Common Messaging Call API. Each object is a collection of properties. Object properties are defined herein in an effort to standardize their representation within this Recommendation.

The object property definitions are preceded with tables summarizing the properties for each object class. The following object property summary tables list the property name and the value type of all defined object properties in columns one and two. The third column provides a description of the property. The fourth column lists possible values for each property. Starred values are defaults. If no star is present, the property has no default value. The fifth column states whether the property is mandatory (M) or optional (O). The sixth column states whether the property is read-only. A "No" in this column means that the property can be modified, updated, or deleted by a call to **cmc_update_properties()**, **cmc_add_properties()**, or **cmc_delete_properties()** respectively, unless otherwise stated. The last column specifies the creator of the property as the implementation (I), the caller (C), or either (E).

Default values can be associated with properties. However, when an implementation creates an object, the implementation should populate the object with explicit values for all the supported properties that have defaults. This will simplify enumerations of properties by the application.

Table 4/X.446 – CMC address book property summary

Address book					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Child Allowed	boolean	CMC_TRUE, CMC_FALSE	O	No	CMC_FALSE
Comment	string	Any Valid String	O	No	None
Location	enum	LOCAL SERVER UNKNOWN[a)]	O	No	UNKNOWN[a)]
Name	string	Any Valid String	O	No	Null String
Object Class	enum	ADDRESS BOOK[b)]	M	Yes	NA
Parent	object_handle	Any Valid Object Handle	M, If nested	No	None
Server Name	string	Any Valid String	O	No	None
Shared	boolean	CMC_TRUE, CMC_FALSE	O	No	CMC_FALSE
Type	enum	GLOBAL, PERSONAL[c)]	O	No	PERSONAL[b)]

a) "CMC_ADDRESS_BOOK_LOCATION_" value prefix.

b) "CMC_OBJECT_TYPE_" value prefix.

c) "CMC_ADDRESS_BOOK_TYPE_" value prefix.

Table 5/X.446 – CMC content item property summary

Content item					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Character Set	guid	GUID For Any Character Set	O	No	Platform – Dependent
Content Information	opaque_data	Any Data	O	No	None
Content Type	guid	GUID For Any Content Type	O	No	None
Create Time	iso_date_time	Any ISO 8601 Date and Time	O	Yes	None
Encoding Type	guid	GUID For Any Encoding Type	O	No	7-BIT[a)]
File Directory	string	Any Valid File Directory	O	No	None
File Name	string	Any Valid File Name	O	No	None
Item Number	uint32	Up to an implementation-defined maximum	M, for more than 1 content item	No	None
Item Type	enum	NOTE ATTACHMENT ANNOTATION[b)]	O	No	NOTE
Last Modified	iso_date_time	Any ISO 8601 Date and Time	O	Yes	None
Object Class	enum	CONTENT ITEM[c)]	M	Yes	NA
Render Position	uint32	Byte position within container	O	No	None
Size	uint32	Byte size	O	No	None
Title	string	Any Valid String	O	No	None

a) "CMC_ADDRESS_BOOK_TYPE_" value prefix.

b) "CMC_IT_" value prefix.

c) "CMC_OBJECT_TYPE_" value prefix.

Table 6/X.446 – CMC distribution list property summary

Distribution list					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Address	string	Any Valid Address	O	Yes	None
Comment	string	Any Valid String	O	No	None
Last Modification Time	iso_date_time	Any ISO 8601 Date and Time	O	Yes	None
Name	string	Any Valid String	M	No	Null String
Object Class	enum	DISTRIBUTION LIST[a)]	M	Yes	NA
Parent	object_handle	Any Valid Object Handle	M, If Nested	No	None
Shared	boolean	CMC_TRUE, CMC_FALSE	O	No	CMC_FALSE

a) "CMC_OBJECT_TYPE_" value prefix.

Table 7/X.446 – CMC message property summary

Message					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Application ID	string	Any Valid String	O	No	None
Application Message Status	flags	Draft	O	No	None
Auto-Action	flags	CMC_AA_ DELETE	O	No	None
Deferred Delivery Time	iso_date_time	Any ISO 8601 Date and Time	O	No	None
In Message Status	flags	NEW, READ, CHANGED[a)]	O	Yes	None
ID	string	Any Valid String	M	Yes	None
In Reply To	string	Any Valid String	O	No	None
Item Count	uint32	Up to an implementation defined maximum	M	Yes	None
NRN Diagnostic	string	Any Valid String	O	No	None

Table 7/X.446 – CMC message property summary *(end)*

Message					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
NRN Reason	string	Any Valid String	M, If Message Type Receipt	No	None
Object Class	object_class	MESSAGE[b)]	M	Yes	NA
Out Message Status	flags	DELETED, SUBMITTED, SENT[a)]	O	Yes	None
Priority	enum	URGENT NORMAL LOW[c)]	O	No	Normal
Receipt Requested	boolean	CMC_TRUE CMC_FALSE	O	No	CMC_FALSE
Receipt Type	enum	RN, NRN[d)]	O	No	None
Report Requested	enum	DR, NDR, BOTH, NONE[e)]	O	No	None
Role	enum	ORIGINAL RETURNED FORWARDED REPLIED OBSOLETED RESENT[f)]	O	No	None
Sensitivity	enum	PERSONAL PRIVATE CONFIDENTIAL NONE[g)]	O	No	None
Size	uint32	Any Valid Byte Value	O	No	None
Subject	string	Any Valid String	O	No	None
Time Received	iso_date_time	Any ISO 8601 Date and Time	M	Yes	None
Time Sent	iso_date_time	Any ISO 8601 Date and Time	M	Yes	None
Type	enum	IPM, REPORT[h)]	M	No	IPM

a) "CMC_MESSAGE_STATUS_" value prefix.

b) "CMC_OBJECT_TYPE_" value prefix.

c) "CMC_PRIORITY_" value prefix.

d) "CMC_RECEIPT_" value prefix.

e) "CMC_REPORT_" value prefix.

f) "CMC_MESSAGE_ROLE_" value prefix.

g) "CMC_MESSAGE_SENSITIVITY_" value prefix.

h) "CMC_MT_" value prefix.

Table 8/X.446 – CMC message container property summary

Message container					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Auto-Action	flags	CMC-AA_Delete	O	Yes	Clear
Child Allowed	boolean	CMC_TRUE CMC_FALSE	O	No	CMC_FALSE
Comment	string	Any Valid String	O	No	None
Location	enum	LOCAL, SERVER, UNKNOWN[a)]	O	No	None
Name	string	Any Valid String	O	No	None
Object Class	enum	MESSAGE CONTAINER[b)]	M	Yes	NA
Parent	object_handle	Any Valid Object Handle	M, if nested	No	None
Server Name	string	Any Valid String	O	No	None
Shared	boolean	CMC_TRUE CMC_FALSE	O	No	CMC_FALSE
Type	enum	DELETED DRAFTS INBOX OUTBOX SENT[c)]	O	Yes	None

a) "CMC_MESSAGE_CONTAINER_" value prefix.

b) "CMC_OBJECT_TYPE_" value prefix.

c) "CMC_MCT_" value prefix.

Table 9/X.446 – CMC per recipient information property summary

Per recipient information					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Comment	string	Any Valid String	O	No	None
Delivery Time	iso_date_time	Any ISO 8601 Date and Time	O	No	None
Diagnostic	string	Any Valid String	O	No	None
Object Class	enum	PER RECIPIENT INFORMATION[a)]	M	Yes	NA
Reason	string	Any Valid String	O	No	None
Recipient Address	string	Any Valid String	M	Yes	NA
Recipient Name	string	Any Valid String	M	No	NA
Type	enum	DR, NDR, UNKNOWN[b)]	M	Yes	NA

a) "CMC_OBJECT_TYPE_" value prefix.

b) "CMC_PRI_" value prefix.

Table 10/X.446 – CMC profile container property summary

Profile container					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Auto-Action	flags	CMC-AA_Delete	O	Yes	Clear
Character Set	array_of_guid	1 or More GUID for Any Character Set	M	Yes	NA
Comment	string	Any Valid String	O	No	None
Conformance	enum	SIMPLE_CMC, FULL_CMC[a)]	M	Yes	NA
Default Service	string	Any Valid String	M	Yes	NA
Default User	string	Any Valid String	M	Yes	NA
Line Terminator	enum	CRLF, LF, CR[b)]	M	Yes	NA
Object Class	enum	Profile Container[c)]	M	Yes	NA

Table 10/X.446 – CMC profile container property summary *(end)*

Profile container					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Object Extensions Supported	array_of_guid	1 or More GUID for CMC Objects	M	Yes	NA
Objects Supported	array_of_guid	1 or More GUID for CMC Objects	M	Yes	NA
Properties Supported	array_of_guid	1 or More GUID for CMC Properties	M	Yes	NA
Property Extensions Supported	array_of_guid	1 or More GUID for CMC Properties	M	Yes	NA
Required Password	enum	NO, OPT, YES[d)]	M	Yes	NA
Required Service	enum	NO, OPT, YES[d)]	M	Yes	NA
Required User	enum	NO, OPT, YES[d)]	M	Yes	NA
Support Counted Strings	boolean	CMC_TRUE CMC_FALSE	M	Yes	NA
Support No Mark As Read	boolean	CMC_TRUE CMC_FALSE	M	Yes	NA
User Interface Available	boolean	CMC_TRUE CMC_FALSE	M	Yes	NA
Users	array_string	Recipient Names	O	Yes	NA
Version of the Implementation	uint16	100 or 200	M	Yes	NA
Version of the Specification	uint16	100 or 200	M	Yes	NA

a) "CMC_CONF_" value prefix.
b) "CMC_LINE_TERM_" value prefix.
c) "CMC_OBJECT_TYPE_" value prefix.
d) "CMC_REQUIRED_" value prefix.

Table 11/X.446 – CMC recipient property summary

Recipient					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Address	string	Any Valid String	M	No	None
Content Return Requested	boolean	CMC_TRUE CMC_FALSE	O	No	None
Name	string	Any Valid String	O	No	None
Object Class	enum	RECIPIENT[a)]	M	Yes	NA
Receipt Requested	enum	RN, NRN, BOTH, NONE[b)]	O	No	None
Report Requested	enum	DR, NDR, BOTH, NONE[c)]	O	No	None
Responsibility Flag	boolean	CMC_TRUE CMC_FALSE	M	No	CMC_TRUE
Role	enum	TO, CC, BCC, ORIGINATOR, AUTHORIZING_ USER, REPLY_TO, FORWARDED, ACTUAL, INTENDED[d)]	O	No	None
Type	enum	UNKNOWN, INDIVIDUAL, GROUP[e)]	M	No	INDIVIDUAL

a) "CMC_OBJECT_TYPE_" value prefix.

b) "CMC_RECEIPT_" value prefix.

c) "CMC_REPORT_" value prefix.

d) "CMC_RECIPIENT_ROLE_" value prefix.

e) "CMC_RCT_" value prefix.

Table 12/X.446 – CMC report property summary

Report					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Application ID	string	Any Valid String	O	No	None
ID	guid	Any Valid ISO 9070 String	M	Yes	NA
Item Count	uint32	Any Valid Integer	M	Yes	NA
Messaging System ID	string	Any Valid String	O	No	None
Object Class	enum	REPORT[a)]	M	Yes	NA
Read	boolean	CMC_TRUE CMC_FALSE	O	No	None
Size	uint32	Byte Size Of Report	O	No	None
Subject	string	Any Valid String	M	No	None
Subject Message ID	string	Any Valid String	O	No	None
Time Received	iso_date_time	Any ISO 8601 Date and Time	M	Yes	None
Time Sent	iso_date_time	Any ISO 8601 Date and Time	M	Yes	None
Unsent	boolean	CMC_TRUE CMC_FALSE	O	No	None

a) "CMC_OBJECT_TYPE_" value prefix.

Table 13/X.446 – CMC root container property summary

Root container					
Property name	**Type (CMC_pv_)**	**Possible values**	**Classification**	**Read-only**	**Default**
Child Allowed	boolean	CMC_TRUE CMC_FALSE	O	No	CMC_FALSE
Comment	string	Any Valid String	O	No	None
Location	enum	LOCAL SERVER UNKNOWN[a)]	O	No	None
Name	string	Any Valid String	O	No	None
Object Class	enum	ROOT CONTAINER[b)]	M	Yes	NA
Shared	boolean	CMC_TRUE CMC_FALSE	O	No	CMC_FALSE
a) "CMC_ROOT_CONTAINER_LOCATION_" value prefix. b) "CMC_OBJECT_TYPE_" value prefix.					

The manual pages for these properties are given in subsequent pages.

5.1 Address book object properties

An address book is a container object composed of addresses of entities and may contain other address books. Support for address books is not mandatory. The following subclauses define, declare, and describe address book properties.

5.1.1 Child allowed

NAME

Address Book Child Allowed

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_CHILD_ALLOWED     \
        "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Child Allowed//EN"
```

DESCRIPTION

The property which permits or denies the existence of a child of the address book.

The default value of this property is CMC_FALSE.

This is a **CMC_pv_boolean** type of property.

5.1.2 Comment

NAME

Address Book Comment

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_COMMENT         \
        "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Comment//EN"
```

DESCRIPTION

A descriptive comment about the address book.

This is a **CMC_pv_string** type of property.

5.1.3 Location

NAME

Address Book Location

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_LOCATION        \
    "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Location//EN"
```

DESCRIPTION

This property specifies the location of the address book.

The valid values for this property include:

CMC_ADDRESS_BOOK_LOCATION_LOCAL
CMC_ADDRESS_BOOK_LOCATION_SERVER
CMC_ADDRESS_BOOK_LOCATION_UNKNOWN

CMC_ADDRESS_BOOK_LOCATION_LOCAL – Specifies that the location of the address book is local and not on the messaging server.

CMC_ADDRESS_BOOK_LOCATION_SERVER – Specifies that the location of the address book is on the messaging server.

CMC_ADDRESS_BOOK_LOCATION_UNKNOWN – Specifies that the location of the address book is unknown. This is the default value.

This is a **CMC_pv_enum** type of property.

5.1.4 Name

NAME

Address Book Name

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_NAME        \
    "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Name//EN"
```

DESCRIPTION

This property specifies the name of the address book.

This is a **CMC_pv_string** type of property.

5.1.5 Object class

NAME

Address Book Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
    "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as address book.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_ADDRESS_BOOK which specifies that the object's class is an address book.

This is a **CMC_pv_enum** type of property.

5.1.6 Parent

NAME

Address Book Parent

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_PARENT          \
   "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Parent//EN"
```

DESCRIPTION

This property specifies the parent of the address book. If the implementation supports the nesting of address books, this property specifies the parent address book. If the address book is the top level, this property is not present. Otherwise, it is mandatory.

This is a **CMC_pv_object_handle** type of property.

5.1.7 Server name

NAME

Address Book Server Name

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_SERVER_NAME        \
   "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Server Name//EN"
```

DESCRIPTION

This property specifies the name of the server on which the address book is located.

This is a **CMC_pv_string** type of property.

5.1.8 Shared

NAME

Address Book Shared

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_SHARED             \
   "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Shared//EN"
```

DESCRIPTION

This property indicates whether more than one user has access to this address book.

The default value for this property is CMC_FALSE if supported.

This is a **CMC_pv_boolean** type of property.

5.1.9 Type

NAME

Address Book Type

C DECLARATION

```
#define CMC_PT_ADDRESS_BOOK_TYPE            \
   "–//XAPIA/CMC/PROPERTY//NONSGML Address Book Type//EN"
```

DESCRIPTION

This property specifies the type of the address book.

The valid values for this property include:

CMC_ADDRESS_BOOK_TYPE_GLOBAL
CMC_ADDRESS_BOOK_TYPE_PERSONAL

CMC_ADDRESS_BOOK_TYPE_GLOBAL – Specifies that the address book is of a global, or enterprise-wide, subtype. A global address book is not necessarily a shared address book.

CMC_ADDRESS_BOOK_TYPE_PERSONAL – Specifies that the address book is of a personal, locally originated and maintained, type.

The default value for this property is CMC_ADDRESS_BOOK_TYPE_PERSONAL.

This is a **CMC_pv_enum** type of property.

5.2 Content item object properties

A content item in this Recommendation is an object associated with the content of a message. It is used to represent attachments and notes, although no distinction is made between the two at the programming interface. The following subclauses define, declare, and describe attachment object properties.

5.2.1 Character set

NAME

Content Item Character Set

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_CHARACTER_SET      \
        "–//XAPIA/CMC/PROPERTY//NONSGML Content Item Character Set//EN"
```

DESCRIPTION

This property specifies the character set of embedded content information within the content item. In the absence of this property, the default character set for embedded content information within the content item is the same as that of the session context.

The property value is a string representing the formal public identifier for the character set. The formal public identifier can be one of the following:

```
#define   CMC_CHARSET_437         "-//XAPIA/CHARSET//NONSGML IBM 437//EN"
#define   CMC_CHARSET_850         "–//XAPIA/CHARSET//NONSGML IBM 850//EN"
#define   CMC_CHARSET_1252        "–//XAPIA/CHARSET//NONSGML Microsoft 1252//EN"
#define   CMC_CHARSET_ISTRING     "–//XAPIA/CHARSET//NONSGML Apple ISTRING//EN"
#define   CMC_CHARSET_UNICODE     "–//XAPIA/CHARSET//NONSGML UNICODE//EN"
#define   CMC_CHARSET_T61         "–//XAPIA/CHARSET//NONSGML TSS T61//EN"
#define   CMC_CHARSET_IA5         "–//XAPIA/CHARSET//NONSGML TSS IA5//EN"
#define   CMC_CHARSET_ISO_10646 "–//XAPIA/CHARSET//NONSGML ISO 10646//EN"
#define   CMC_CHARSET_ISO_646     "–//XAPIA/CHARSET//NONSGML ISO 646//EN"
#define   CMC_CHARSET_ISO_8859_1  "–//XAPIA/CHARSET//NONSGML ISO 8859-1//EN"
```

Implementations may provide for other character sets.

This is a **CMC_pv_guid** type of property.

5.2.2 Content information

NAME

Content Item Content Information

C DECLARATION

```
    #define CMC_PT_CONTENT_ITEM_CONTENT_INFORMATION   \
        "–//XAPIA/CMC/PROPERTY//NONSGML Content Item Content Information//EN"
```

DESCRIPTION

This property holds the content of a content item.

This is a **CMC_pv_opaque_data** type of property.

5.2.3 Content type

NAME

Content Item Content Type

C DECLARATION

```
    #define CMC_PT_CONTENT_ITEM_CONTENT_TYPE       \
        "–//XAPIA/CMC/PROPERTY//NONSGML Content Item Content Type//EN"
```

DESCRIPTION

This property specifies the content type of the content item. A NULL value designates an undefined content item type.

The following GUID values are valid for the Type property of the Content Item objects.

```
#define CMC_CT_PLAIN_TEXT                                         \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Plain Text//EN"
#define CMC_CT_GIF_IMAGE                                          \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML GIF Image//EN"
#define CMC_CT_JPEG_IMAGE                                         \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML JPEG Image//EN"
#define CMC_CT_BASIC_AUDIO                                        \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Basic Audio//EN"
#define CMC_CT_MPEG_VIDEO                                         \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML MPEG Video//EN"
#define CMC_CT_MESSAGE                                            \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Message//EN"
#define CMC_CT_PARTIAL_MESSAGE                                    \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Partial Message//EN"
#define CMC_CT_EXTERNAL_MESSAGE                                   \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML External Message//EN"
#define CMC_CT_APPLICATION_OCTET_STREAM                           \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Application Octet Stream//EN"
#define CMC_CT_APPLICATION_POSTSCRIPT                          \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Application PostScript//EN"
#define CMC_CT_ALTERNATIVE_MULTIPART                                  \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Alternative Multipart//EN"
#define CMC_CT_DIGEST_MULTIPART                                   \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Digest Multipart//EN"
#define CMC_CT_MIXED_MULTIPART                                    \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Mixed Multipart//EN"
#define CMC_CT_OLE                                                \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML OLE//EN"
#define CMC_CT_MIXED_MULTIPART                                    \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML Mixed Multipart//EN"
#define CMC_CT_X400_G3_FAX                                        \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 G3 Fax//EN"
#define CMC_CT_X400_G4_FAX                                        \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 G4 Fax//EN"
#define CMC_CT_X400_ENCRYPTED                                     \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Encrypted//EN"
#define CMC_CT_X400_NATIONALLY_DEFINED                            \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Nationally Defined//EN"
#define CMC_CT_X400_FILE_TRANSFER                                 \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 File Transfer//EN"
#define CMC_CT_X400_VOICE                                         \
    "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Voice//EN"
```

```
#define CMC_CT_X400_VIDEOTEX                                            \
        "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Videotex//EN"
#define CMC_CT_X400_MIXED_MODE                                          \
        "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Mixed Mode//EN"
#define CMC_CT_X400_PRIVATELY_DEFINED_6937                              \
        "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Privately Defined 6937//EN"
#define CMC_CT_X400_EXTERNAL_TRACE                                      \
        "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 External Trace//EN"
#define CMC_CT_X400_INTERNAL_TRACE                                      \
        "–//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Internal Trace//EN"
#define CMC_CT_SMTP_SESSION_TRANSCRIPT                                  \
        "–//XAPIA/CMC/CONTENT TYPE//NONSGML SMTP Session Transcript//EN"
```

CMC_CT_PLAIN_TEXT – Specifies plain or unformatted text content.

CMC_CT_GIF_IMAGE – Specifies image data content in the form of the Graphics Image Format used by MIME and the World Wide Web.

CMC_CT_JPEG_IMAGE – Specifies image data content in the form of the ISO Joint Picture Encoding Group standard used by MIME and the World Wide Web.

CMC_CT_BASIC_AUDIO – Specifies audio data content in the form of audio encoded using 8-bit ISDN mu-law or PCM defined by Recommendation G.711 with a sample rate of 8000 Hz and with a single channel.

CMC_CT_MPEG_VIDEO – Specifies video content in the form of the ISO Motion Picture Encoding Group, ISO 11172 standard used by MIME and the World Wide Web.

CMC_CT_MESSAGE – Specifies that the content is an encapsulated message.

CMC_CT_PARTIAL_MESSAGE – Specifies that the content is a portion of another message. This content type allows a large message to be delivered as several separate pieces to facilitate receipt.

CMC_CT_EXTERNAL_MESSAGE – Specifies that the content is external to the message. The content information property contains a textual reference to the external content information.

CMC_CT_APPLICATION_OCTET_STREAM – Specifies that the content is an application-dependent stream of octets.

CMC_CT_APPLICATION_POSTSCRIPT – Specifies that the content is an Adobe Systems, Inc. PostScript program.

CMC_CT_ALTERNATIVE_MULTIPART – Specifies that the content is one of an alternative form of content to another note or content item within the message object.

CMC_CT_DIGEST_MULTIPART – Specifies that the content is one of a group of related messages within the message object. The messages may serve as a sequence of discussions captured in a thread of messages as is found on bulletin board systems.

CMC_CT_MIXED_MULTIPART – Specifies that the content is one of an ordered sequence of messages within the message object.

CMC_CT_PARALLEL_MULTIPART – Specifies that the content is one of a group of arbitrary ordered sequence of messages within the message object.

CMC_CT_OLE – Specifies that the content item type is OLE (Object Linking and Embedding) object content item.

CMC_CT_X400_G3_FAX – Specifies that the content represents Group 3 facsimile images, a sequence of bit strings. Each G3 data component encodes a single page of data as dictated by Recommendations T.4 and T.30.

CMC_CT_X400_G4_FAX – Specifies that the content represents a final-form document of the sort that is processable by Group 4 class 1 facsimile terminals.

CMC_CT_X400_ENCRYPTED – Specifies that the content is bit strings and encoded in accordance with the basic encoding rules of Recommendation X.209.

CMC_CT_X400_NATIONALLY_DEFINED – Specifies the content is an information object whose semantics and abstract syntax are nationally defined by a country whose identity is bilaterally agreed by the message's originator and all of its potential recipients.

CMC_CT_X400_FILE_TRANSFER – Specifies that the content information consists of relatively large amounts of data. The content information property contains the textual reference to its semantics and abstract syntax, which are denoted by an object identifier.

CMC_CT_X400_VOICE – Specifies that the content is the digitized speech, a bit string. Its encoding are currently not defined in the 1988 version of Recommendation X.420.

CMC_CT_X400_VIDEOTEX – Specifies that the content represents videotex data. Its syntax is defined in Recommendations T.100 and T.101.

CMC_CT_X400_MIXED_MODE – Specifies that the content represents a final-form document of the sort that is processable by mixed-mode Teletex terminals and Group 4 class 2 and 3 facsimile terminals.

CMC_CT_X400_PRIVATELY_DEFINED_6937 – Specifies that the content is privately defined. The content is encoded in accordance to ISO 6937 specified character sets and encoding rules.

CMC_CT_X400_EXTERNAL_TRACE – Specifies that the content contains X.400 external trace information for diagnostic purpose.

CMC_CT_X400_INTERNAL_TRACE – Specifies that the content contains X.400 internal trace information for diagnostic purpose.

CMC_CT_SMTP_SESSION_TRANSCRIPT – Specifies that the content contains SMTP session transcript information for diagnostic purpose.

This is a **CMC_pv_guid** type of property.

5.2.4 Create time

NAME

Content Item Create Time

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_CREATE_TIME        \
    "–//XAPIA/CMC/PROPERTY//NONSGML Content Item Create Time//EN"
```

DESCRIPTION

This property specifies the date and time that the content item was created.

This is a **CMC_pv_iso_date_time** type of property.

5.2.5 Encoding type

NAME

Content Item Encoding Type

C DECLARATION

```
#define CMC_PT_CONTENT ITEM_ENCODING_TYPE      \
    "–//XAPIA/CMC/PROPERTY//NONSGML Content Item Encoding Type//EN"
```

DESCRIPTION

This property specifies the encoding type of the content of the content item.

The default value for this property is CMC_ET_7_BIT.

The following values are valid for the Encoding Type property of content item object:

```
#define CMC_ET_7_BIT                                 \
   "–//XAPIA/CMC/ENCODING TYPE//NONSGML 7 Bit//EN"
#define CMC_ET_BASE64                                \
   "–//XAPIA/CMC/ENCODING TYPE//NONSGML Base64//EN"
```

```
#define CMC_ET_BINARY                                 \
   "-//XAPIA/CMC/ENCODING TYPE//NONSGML Binary//EN"
#define CMC_ET_8_BIT                                  \
   "-//XAPIA/CMC/ENCODING TYPE//NONSGML 8 Bit//EN"
#define CMC_ET_QUOTED_PRINTABLE                       \
   "-//XAPIA/CMC/ENCODING TYPE//NONSGML Quoted Printable//EN"
```

CMC_ET_7_BIT – Specifies that no encoding has been performed on the content information. Additionally, it means that the content information consists of octets of 7-bit data.

CMC_ET_BASE64 – Specifies that the content information has been encoded in the Base 64 form of RFC 1521/MIME for arbitrary sequence of octets.

CMC_ET_BINARY – Specifies that no encoding has been performed on the content information. Additionally, it means that the content information consists of relatively large amounts of data and that the octets may have the high-order bit set.

CMC_ET_8_BIT – Specifies that no encoding has been performed on the content information. Additionally, it means that the content information consists of relatively short lines of octets with the high-order bit set.

CMC_ET_QUOTED_PRINTABLE – Specifies that the content information has been encoded in the form of RFC 1521/MIME for largely printable characters in the ASCII character set such that the resulting octets are unlikely to be modified by mail transports.

This is a **CMC_pv_guid** type of property.

5.2.6 File directory

NAME

Content Item File Directory

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_FILE_DIRECTORY     \
```

"–//XAPIA/CMC/PROPERTY//NONSGML Content Item File Directory//EN"

DESCRIPTION

This property specifies the file directory of the content item when it was added to the message. Where the Content Type property is CMC_CT_EXTERNAL_MESSAGE, this property denotes the server name as well as the directory name on a remote server. The format of this string is implementation-defined.

This is a **CMC_pv_string** type of property.

5.2.7 File name

NAME

Content Item File Name

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_FILE_NAME          \
```

"–//XAPIA/CMC/PROPERTY//NONSGML Content Item File Name//EN"

DESCRIPTION

This property specifies the file name of the content item when it was added to the message. Where the Content Type property is CMC_CT_EXTERNAL_MESSAGE, this property denotes the file name on a remote server. The format of this string is implementation-defined.

This is a **CMC_pv_string** type of property.

5.2.8 Item number

NAME

Content Item Item Number

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_ITEM_NUMBER        \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Content Item Item Number/EN"

DESCRIPTION

This property specifies the sequence number of the item within its parent container, a message object or another content item. This is a mandatory property.

No two items within a given parent container can have the same value for Item Number.

This is a **CMC_pv_uint32** type of property.

5.2.9 Item type

NAME

Content Item Item Type

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_ITEM_TYPE          \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Content Item Item Type//EN"

DESCRIPTION

This property specifies the type of the content item. Valid values for this property include:

CMC_IT_NOTE
CMC_IT_ATTACHMENT
CMC_IT_ANNOTATION

CMC_IT_NOTE – Specifies a note type.

CMC_IT_ATTACHMENT – Specifies an attachment type.

CMC_IT_ANNOTATION – Specifies an annotation on another content item object.

This is a **CMC_pv_enum** type of property.

5.2.10 Last modified

NAME

Content Item Last Modified

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_LAST_MODIFIED      \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Content Item Last Modified//EN"

DESCRIPTION

This property specifies the date and time that the file from which the content item was derived was last modified.

This is a **CMC_pv_iso_date_time** type of property.

5.2.11 Object class

NAME

Content Item Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Object Class //EN"

DESCRIPTION

This property defines the class of the object as a content item. This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_CONTENT_ITEM which specifies that the object's class is a content item.

This is a **CMC_pv_enum** type of property.

5.2.12 Render position

NAME

Content Item Render Position

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_RENDER_POSITION   \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Content Item Render Position//EN"

DESCRIPTION

This property specifies the position of the content item within its container.

This is a **CMC_pv_uint32** type of property.

5.2.13 Size

NAME

Content Item Size

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_SIZE           \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Content Item Size//EN"

DESCRIPTION

This property specifies the size of the content item.

This is a **CMC_pv_uint32** type of property.

5.2.14 Title

NAME

Content Item Title

C DECLARATION

```
#define CMC_PT_CONTENT_ITEM_TITLE          \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Content Item Title//EN"

DESCRIPTION

This property specifies the full description of the content item. For example, "Quarterly Financial Report" could be a content item title.

This is a **CMC_pv_string** type of property.

5.3 Distribution list object properties

Distribution lists identify groups of users. A distribution list contains recipient objects. The following subclauses define, declare, and describe distribution list properties.

5.3.1 Address

NAME

Distribution List Address

C DECLARATION

```
#define CMC_PT_DISTRIBUTION_LIST_ADDRESS              \
       "–//XAPIA/CMC/PROPERTY//NONSGML Distribution List Address//EN"
```

DESCRIPTION

This property provides the address for a distribution list to be used in mailing to the list. This address is often the same value as the name of the distribution list. It is an optional property of a distribution list.

This property is generated by the messaging system.

This is a **CMC_pv_string** type of property.

5.3.2 Comment

NAME

Distribution List Comment

C DECLARATION

```
#define CMC_PT_DISTRIBUTION_LIST_COMMENT              \
       "–//XAPIA/CMC/PROPERTY//NONSGML Distribution List Comment//EN"
```

DESCRIPTION

A descriptive comment about the distribution list.

This is a **CMC_pv_string** type of property.

5.3.3 Last modification time

NAME

Distribution List Last Modification Time

C DECLARATION

```
#define CMC_PT_DISTRIBUTION_LIST_LAST_MODIFICATION_TIME              \
       "–//XAPIA/CMC/PROPERTY//NONSGML Distribution List Last Modification Time//EN"
```

DESCRIPTION

This property specifies the date and time of the last update to the distribution list.

This is a **CMC_pv_iso_date_time** type of property.

5.3.4 Name

NAME

Distribution List Name

C DECLARATION

```
#define CMC_PT_DISTRIBUTION_LIST_NAME      \
       "–//XAPIA/CMC/PROPERTY//NONSGML Distribution List Name//EN"
```

DESCRIPTION

This property specifies the name of the distribution list. It is a mandatory property for distribution list objects.

The string may be generated by the messaging system from a directory or by the user.

This is a **CMC_pv_string** type of property.

5.3.5 Object class

NAME

Distribution List Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
    "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as a distribution list. This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_DISTRIBUTION_LIST which specifies that the object's class is a distribution list.

This is a **CMC_pv_enum** type of property.

5.3.6 Parent

NAME

Distribution List Parent

C DECLARATION

```
#define CMC_PT_DISTRIBUTION_LIST_PARENT       \
    "–//XAPIA/CMC/PROPERTY//NONSGML Distribution List Parent//EN"
```

DESCRIPTION

This property specifies the parent of the distribution list. If the implementation supports the nesting of distribution lists, this property specifies the parent distribution list. If the distribution list is the top level, this property is not present. Otherwise, it is mandatory. This property is null for the parent.

This is a **CMC_pv_object_handle** type of property.

5.3.7 Shared

NAME

Distribution List Shared

C DECLARATION

```
#define CMC_PT_DISTRIBUTION_LIST_SHARED       \
    "–//XAPIA/CMC/PROPERTY//NONSGML Distribution List Shared//EN"
```

DESCRIPTION

This property indicates whether more than one user has access to this distribution list.

The default value for this property is CMC_FALSE if supported.

This is a **CMC_pv_boolean** type of property.

5.4 Message object properties

The message object is a collection of message specific object properties. The following subclauses define, declare, and describe message object properties.

5.4.1 Application Id

NAME

Message Application Id

C DECLARATION

```
#define CMC_PT_MESSAGE_APPLICATION_ID      \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Application Id//EN"

DESCRIPTION

This property specifies a globally unique identifier for the message. This property is set by the application.

This is a **CMC_pv_string** type of property.

5.4.2 Application message status

NAME

Message Application Message Status

C DECLARATION

```
#define CMC_PT_MESSAGE_APPLICATION_MSG_STATUS           \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Application Msg Status//EN"

DESCRIPTION

This property specifies the caller specified status for the message. This property can be used by the caller to mark whether a message is a draft or completed message. There are no implied semantics to the messaging service; however, the value of the property is persistent across sessions.

The valid values for this property include:

CMC_MESSAGE_STATUS_DRAFT

CMC_MESSAGE_STATUS_DRAFT – Specifies that the message is in draft mode.

This is a **CMC_pv_flags** type of property.

5.4.3 Auto-Action

NAME

Message Auto-Action

C DECLARATION

```
#define CMC_PT_MESSAGE_AUTO_ACTION                \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Auto Action//EN"

DESCRIPTION

This property specifies the automatic action or disposition of the message after it is sent. Support for this property is optional for implementations conforming to this Recommendation. In the case of newly created messages, the property is created by a call to **cmc_add_properties()**. In the case of newly created messages, the property can also be modified by a call to **cmc_add_properties()** or deleted by a call to **cmc_delete_properties()**. This property at the message object will override the preference set in the profile container object.

The valid value for this property is:

CMC_AA_DELETE

Set:	The specified message is to be deleted by the underlying messaging system after it has been successfully submitted for transfer.
Clear:	The specified message is placed in the sent folder if it exists. If not, the message is deleted.

This is a **CMC_pv_flags** type of property.

5.4.4 Deferred delivery time

NAME

Message Deferred Delivery Time

C DECLARATION

```
#define CMC_PT_MESSAGE_DEFERRED_DELIVERY_TIME                \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Deferred Delivery Time//EN"

DESCRIPTION

This property specifies the UTC (Coordinate Universal Time) date and time before which the message should not be delivered to recipients.

This is a **CMC_pv_iso_date_time** type of property.

5.4.5 Id

NAME

Message Id

C DECLARATION

```
#define CMC_PT_MESSAGE_ID         \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Id//EN"

DESCRIPTION

This property specifies a globally unique identifier for the message. This property is set by **cmc_send_message_object()**, is defined by the messaging service (established at submission), and is unique within the domain.

In gateway applications, the Message Id may be added or updated by the caller.

This is a **CMC_pv_string** type of property.

5.4.6 In message status

NAME

Message In Message Status

C DECLARATION

```
#define CMC_PT_MESSAGE_IN_MSG_STATUS        \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message In Msg Status//EN"

DESCRIPTION

This property specifies the messaging service specified, input or receipt status for the message. This property is used by the underlying messaging service to record modal status information about the receipt and processing of the message. For example, the fact that a message has just been received in an inbox, has been read, or has been changed from its original receipt can be specified by the messaging service.

Support for this property is optional for implementations conforming to this Recommendation. The property is read-only; it is created and modified by the underlying messaging service. The property cannot be deleted by the user.

The valid values for this property include:

CMC_MESSAGE_STATUS_NEW
CMC_MESSAGE_STATUS_READ
CMC_MESSAGE_STATUS_CHANGED

CMC_MESSAGE_STATUS_NEW – Specifies that the message has just been received by the underlying messaging service. The flag will be reset when the session is closed, the message object is accessed, or the message container is closed.

CMC_MESSAGE_STATUS_READ – Specifies that the message has been read. This status flag is set when a property of one of the subordinate content item objects for the message has been read by a call to **cmc_read_properties()**.

CMC_MESSAGE_STATUS_CHANGED – Specifies whether the contents of a message has changed from the form it was in when it was originally received. This status flag is set when any property contained within the message object is added to or modified by a call to the **cmc_add_properties()** function or deleted by a call to the **cmc_delete_properties()** function.

This is a **CMC_pv_flags** type of property.

5.4.7 In reply to

NAME

Message In Reply To

C DECLARATION

```
#define CMC_PT_MESSAGE_IN_REPLY_TO          \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message In Reply To//EN"

DESCRIPTION

This property specifies the previous correspondence which this message answers.

The property value may be a textual reference or may be a textual approximation of the message identifier of the previous correspondence.

This is a **CMC_pv_string** type of property.

5.4.8 Item count

NAME

Message Item Count

C DECLARATION

```
#define CMC_PT_MESSAGE_ITEM_COUNT    \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Item Count//EN"

DESCRIPTION

This property specifies the number of top-level content items contained in a message. This count does not include content items nested in other content items, messages, or reports. This property is set by the implementation.

This is a **CMC_pv_uint32** type of property.

5.4.9 NRN diagnostic

NAME

Message NRN Diagnostic

C DECLARATION

```
#define CMC_PROP_TYPE_MESSAGE_NRN_DIAGNOSTIC                          \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message NRN Diagnostic//EN"

DESCRIPTION

The property specifies the diagnostic details of the reason for the non-receipt notification. These are additional details for the non-receipt reason. This property only pertains to messages of type CMC_MT_RECEIPT.

This is a **CMC_pv_string** type of property.

5.4.10 NRN reason

NAME

Message NRN Reason

C DECLARATION

```
#define CMC_PROP_TYPE_MESSAGE_NRN_REASON                              \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message NRN Reason//EN"

DESCRIPTION

This property explains why the message was not received. This property only pertains to messages of type CMC_MT_RECEIPT with the receipt type property of CMC_RECEIPT_NRN.

This is a **CMC_pv_string** type of property.

5.4.11 Object class

NAME

Message Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
     "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as a message.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_MESSAGE which specifies that the object's class is a message.

This is a **CMC_pv_enum** type of property.

5.4.12 Out message status

NAME

Message Out Message Status

C DECLARATION

```
#define CMC_PT_MESSAGE_OUT_MSG_STATUS                         \
     "–//XAPIA/CMC/PROPERTY//NONSGML Message Out Msg Status//EN"
```

DESCRIPTION

This property specifies the messaging service specified, output or disposition status for the message. This property is used by the underlying messaging service to record modal status information about the disposition of the message. For example, the fact that a message has been marked for delete, has been submitted for transfer, or is in the process of being sent can be specified by the messaging service.

Support for this property is optional for implementations conforming to this Recommendation. The property is read-only; it is created and modified by the underlying messaging service. The property cannot be deleted by the user.

The valid values for this property include:

CMC_MESSAGE_STATUS_DELETED
CMC_MESSAGE_STATUS_SUBMITTED
CMC_MESSAGE_STATUS_SENT

CMC_MESSAGE_STATUS_DELETED – Specifies that the message is in transition to being deleted. In some implementation environments (e.g. disconnected user), a deletion operation on a message may not be able to be acted on immediately. This flag indicates that even though the message appears in a message container, it has been marked for delete.

CMC_MESSAGE_STATUS_SUBMITTED – Specifies that the message has been submitted for transfer by the underlying messaging service by either a call to the **cmc_send_message_object()** function or a call to the **cmc_commit_object()** function for committing a message object to an outbox type of message container. In some implementation environments (e.g. disconnected user), a send operation on a message may not be able to be acted on immediately. This flag indicates that even though the message appears in a message container, it has been marked for submission to the underlying messaging service.

CMC_MESSAGE_STATUS_SENT – Specifies that the message is in transition to being sent by the underlying messaging service. In some implementation environments, the transfer of a message by the underlying messaging service may not be immediate. In such cases, the message may appear in a message container even though it has been sent by the application. This flag indicates such a state.

This is a **CMC_pv_flags** type of property.

5.4.13 Priority

NAME

Message Priority

C DECLARATION

```
#define CMC_PT_MESSAGE_PRIORITY      \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Priority//EN"
```

DESCRIPTION

This property specifies the priority of the message. The property is defaultable. It can be set when the message is created.

The valid values for this property include:

CMC_PRIORITY_URGENT
CMC_PRIORITY_NORMAL
CMC_PRIORITY_LOW

CMC_PRIORITY_URGENT – Specifies that the message is of an urgent priority.

CMC_PRIORITY_NORMAL – Specifies that the message is of a nominal priority. This is the default value.

CMC_PRIORITY_LOW – Specifies that the message is of a low priority.

This is a **CMC_pv_enum** type of property.

5.4.14 Receipt requested

NAME

Message Receipt Requested

C DECLARATION

```
#define CMC_PT_MESSAGE_RECEIPT_REQUESTED             \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Receipt Requested//EN"
```

DESCRIPTION

This property indicates whether a receipt for the message sent was requested.

The valid values for this property are:

CMC_RECEIPT_RN
CMC_RECEIPT_NRN
CMC_RECEIPT_BOTH
CMC_RECEIPT_NONE

CMC_RECEIPT_RN – Requests that a receipt notification is returned only when the recipient has received the subject message.

CMC_RECEIPT_NRN – Requests that a non-receipt notification is returned only when the recipient has failed to receive the subject message.

CMC_RECEIPT_BOTH – Requests that either a receipt notification or a non-receipt notification is returned depending on whether the recipient has received or failed to receive the subject message.

CMC_RECEIPT_NONE – Requests that no receipt should be returned regardless of whether the recipient has received or failed to receive the subject message.

This is a **CMC_pv_enum** type of property.

5.4.15 Receipt type

NAME

Message Receipt Type

C DECLARATION

```
#define CMC_PT_MESSAGE_RECEIPT_TYPE                                   \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Receipt Type//EN"

DESCRIPTION

The type of the receipt returned for the subject message. This is used to indicate whether the subject message has been received or not received by the intended recipient.

The valid values for this property include:

CMC_RECEIPT_RN
CMC_RECEIPT_NRN

CMC_RECEIPT_RN – Specifies that this is a receipt notification.

CMC_RECEIPT_NRN – Specifies that this is a non-receipt notification.

This property pertains only if the message type property has the value of CMC_MESSAGE_TYPE_RECEIPT.

This is a **CMC_pv_enum** type of property.

5.4.16 Report requested

NAME

Message Report Requested

C DECLARATION

```
#define CMC_PT_MESSAGE_REPORT_REQUESTED                               \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Report Requested//EN"

DESCRIPTION

The type of the report to be returned for the subject message. This is used to indicate whether the subject message has been delivered or not delivered by the underlying messaging transport systems.

The valid values for this property include:

CMC_REPORT_DR
CMC_REPORT_NDR
CMC_REPORT_BOTH
CMC_REPORT_NONE

CMC_REPORT_DR – Specifies that a delivery report is requested.

CMC_REPORT_NDR – Specifies that a non-delivery report is requested.

CMC_REPORT_BOTH – Specifies that either delivery report, or non-delivery report is requested, whichever applicable.

CMC_REPORT_NONE – Specifies that neither delivery report nor non-delivery report is requested.

This is a **CMC_pv_enum** type of property.

5.4.17 Role

NAME

Message Role

C DECLARATION

```
#define CMC_PT_MESSAGE_ROLE                                           \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Role//EN"

DESCRIPTION

The role of this message.

The valid values for this property include:

CMC_MESSAGE_ROLE_ORIGINAL
CMC_MESSAGE_ROLE_RETURNED
CMC_MESSAGE_ROLE_FORWARDED
CMC_MESSAGE_ROLE_REPLIED
CMC_MESSAGE_ROLE_OBSOLETED
CMC_MESSAGE_ROLE_RESENT

CMC_MESSAGE_ROLE_ORIGINAL – Specifies that this is the original message.

CMC_MESSAGE_ROLE_RETURNED – Specifies that this is a returned message, content of another message.

CMC_MESSAGE_ROLE_FORWARDED – Specifies that this is a forwarded message, content of another message.

CMC_MESSAGE_ROLE_REPLIED – Specifies that this is a reply message to another message.

CMC_MESSAGE_ROLE_OBSOLETED – Specifies that this is an obsolete message.

CMC_MESSAGE_ROLE_RESENT – Specifies that this is a resent copy of another message, the original.

This is a **CMC_pv_enum** type of property.

5.4.18 Sensitivity

NAME

Message Sensitivity

C DECLARATION

```
#define CMC_PT_MESSAGE_SENSITIVITY          \
    "–//XAPIA/CMC/PROPERTY//NONSGML Message Sensitivity//EN"
```

DESCRIPTION

This property specifies the sensitivity of the message.

The valid values for this property include:

CMC_MESSAGE_SENSITIVITY_PERSONAL
CMC_MESSAGE_SENSITIVITY_PRIVATE
CMC_MESSAGE_SENSITIVITY_CONFIDENTIAL
CMC_MESSAGE_SENSITIVITY_NONE

CMC_MESSAGE_SENSITIVITY_PERSONAL – Specifies that the message is personal.

CMC_MESSAGE_SENSITIVITY_PRIVATE – Specifies that the message is private.

CMC_MESSAGE_SENSITIVITY_CONFIDENTIAL – Specifies that the message is confidential.

CMC_MESSAGE_SENSITIVITY_NONE – Specifies that the message is non-sensitive.

This is a **CMC_pv_enum** type of property.

5.4.19 Size

NAME

Message Size

C DECLARATION

```
#define CMC_PT_MESSAGE_SIZE          \
    "–//XAPIA/CMC/PROPERTY//NONSGML Message Size//EN"
```

DESCRIPTION

This property specifies the size of the message.

This is a **CMC_pv_uint32** type of property.

5.4.20 Subject

NAME

Message Subject

C DECLARATION

```
#define CMC_PT_MESSAGE_SUBJECT        \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Subject//EN"
```

DESCRIPTION

This property states the subject of the message. This property is defaultable to a null string.

This is a **CMC_pv_string** type of property.

5.4.21 Time received

NAME

Message Time Received

C DECLARATION

```
#define CMC_PT_MESSAGE_TIME_RECEIVED        \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Time Received//EN"
```

DESCRIPTION

This property specifies the date and time that the message was received.

This is a **CMC_pv_iso_date_time** type of property.

5.4.22 Time sent

NAME

Message Time Sent

C DECLARATION

```
#define CMC_PT_MESSAGE_TIME_SENT     \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Time Sent//EN"
```

DESCRIPTION

This property specifies the date and time that the message was sent.

This property is set by the service in **cmc_send_message_object()**.

This is a **CMC_pv_iso_date_time** type of property.

5.4.23 Type

NAME

Message Type

C DECLARATION

```
#define CMC_PT_MESSAGE_TYPE       \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Type//EN"
```

DESCRIPTION

This property specifies the type of the message. Support for this property is optional for implementations conforming to this Recommendation. In the case of received or existing messages, the property can be created by the messaging system. In the case of newly created messages, the property is created by a call to **cmc_add_properties()**. In the case of newly created messages, the property can also be modified by a call to **cmc_add_properties()** or deleted by a call to **cmc_delete_properties()**.

The valid values for this property are:

CMC_MT_IPM
CMC_MT_RECEIPT
CMC_MT_EDI

CMC_MT_DIRECTORY
CMC_MT_DOCMGMT
CMC_MT_WORKFLOW
CMC_MT_CALSCHED

CMC_MT_IPM – Electronic mail or interpersonal message, in the parlance of Recommendation X.400.

CMC_MT_RECEIPT – A messaging receipt. This message type is used for Receipt Notification and Non-Receipt Notification. The message type may also be useful for other message receipts also.

The following message types are reserved for the purposes specified. The values represent work-in-progress by the XAPIA and other industry groups. These message types may be modified in future versions of this Recommendation to reflect completion of this work.

CMC_MT_EDI – Electronic data interchange type message. The form and format of the EDI messages are not specified by this Recommendation.

CMC_MT_DIRECTORY – Directory services type message. This message type provides for the use of the messaging services as a transport for directory inquiry functions. The form and format of the directory messages are not specified by this Recommendation.

CMC_MT_DOCMGMT – Document management type message. This message type provides for the access and search of library services using the messaging service as a transport for the document management inquiry functions. The form and format of the document management messages are not specified by this Recommendation.

CMC_MT_WORKFLOW – Workflow management type message. This message type facilitates the automated handling of business processes by using the messaging service as a transport for the workflow functions. The form and format of the workflow management messages are not specified by this Recommendation.

CMC_MT_CALSCHED – Calendaring and Scheduling type message. This message type provides the use of the messaging service as a transport for calendaring and scheduling functions. The form and format of the calendaring and scheduling messages are not specified by this Recommendation. There are other XAPIA specifications that provide for the definition of calendaring and scheduling interoperability specification.

This is a **CMC_pv_enum** type of property.

5.5 Message container object properties

A message container object is a collection of container properties, message objects, and, quite possibly, other message containers. The following subclauses define, declare, and describe message container object properties.

5.5.1 Child allowed

NAME

Message Container Child Allowed

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_CHILD_ALLOWED                      \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Container Child Allowed//EN"

DESCRIPTION

This property permits or denies the existence of a child of the message container.

This is a **CMC_pv_boolean** type of property.

5.5.2 Comment

NAME

Message Container Comment

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_COMMENT              \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Message Container Comment//EN"

DESCRIPTION

A descriptive comment about the message container.

This is a **CMC_pv_string** type of property.

5.5.3 Location

NAME

Message Container Location

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_LOCATION             \
    "–//XAPIA/CMC/PROPERTY//NONSGML Message Container Location//EN"
```

DESCRIPTION

The location of the message container.

The valid values for this property include:

CMC_MESSAGE_CONTAINER_LOCATION_LOCAL
CMC_MESSAGE_CONTAINER_LOCATION_SERVER
CMC_MESSAGE_CONTAINER_LOCATION_UNKNOWN

CMC_MESSAGE_CONTAINER_LOCATION_LOCAL – Specifies that the location of the message container is local and not on the messaging server.

CMC_MESSAGE_CONTAINER_LOCATION_SERVER – Specifies that the location of the message container is on the messaging server.

CMC_MESSAGE_CONTAINER_LOCATION_UNKNOWN – Specifies that the location of the message container is unknown.

This is a **CMC_pv_enum** type of property.

5.5.4 Name

NAME

Message Container Name

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_NAME       \
    "–//XAPIA/CMC/PROPERTY//NONSGML Message Container Name//EN"
```

DESCRIPTION

The name of the message container.

This is a **CMC_pv_string** type of property.

5.5.5 Object class

NAME

Message Container Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS           \
    "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as a message container.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_MESSAGE_CONTAINER which specifies that the object's class is a message container.

This is a **CMC_pv_enum** type of property.

5.5.6 Parent

NAME

Message Container Parent

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_PARENT                 \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Container Parent//EN"
```

DESCRIPTION

The parent of the message container. If the implementation supports the nesting of message containers, this property specifies the parent message container. If the message container is the top level, this property is not present. Otherwise, it is mandatory.

This is a **CMC_pv_object_handle** type of property.

5.5.7 Server name

NAME

Message Container Server Name

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_SERVER_NAME            \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Container Server Name//EN"
```

DESCRIPTION

This property specifies the name of the server on which the message container is located.

This is a **CMC_pv_string** type of property.

5.5.8 Shared

NAME

Message Container Shared

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_SHARED     \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Container Shared//EN"
```

DESCRIPTION

This property specifies whether more than one user has access to this message container.

This is a **CMC_pv_boolean** type of property.

5.5.9 Type

NAME

Message Container Type

C DECLARATION

```
#define CMC_PT_MESSAGE_CONTAINER_TYPE       \
        "–//XAPIA/CMC/PROPERTY//NONSGML Message Container Type//EN"
```

DESCRIPTION

This property specifies the type of the container of the message.

The valid values for this property include:

CMC_MCT_DELETED
CMC_MCT_DRAFTS
CMC_MCT_FILED
CMC_MCT_INBOX
CMC_MCT_OUTBOX
CMC_MCT_SENT

CMC_MCT_DELETED – Specifies that the message container is for deleted messages.

CMC_MCT_DRAFTS – Specifies that the message container is for draft messages.

CMC_MCT_FILED – Specifies that the message container is for filed messages.

CMC_MCT_INBOX – Specifies that the message container is the inbox. An implementation may have more than one inbox.

CMC_MCT_OUTBOX – Specifies that the message container is for outgoing messages. An implementation must have at least one outbox and it is mandatory. Objects committed to the outbox are not modifiable. Committed objects can only be deleted or copied.

CMC_MCT_SENT – Specifies that the message container is for messages that have been sent. The sent box is optional. An implementation will have, at the most, one sent box (0-1 sent box).

This is a **CMC_pv_enum** type of property.

5.6 Per recipient information object properties

Per Recipient Information objects are components of a report that is generated to reflect the delivery or non-delivery status of a message. The following subclauses define, declare, and describe the Per Recipient Information object properties.

5.6.1 Comment

NAME

Per Recipient Information Comment

C DECLARATION

```
#define CMC_PT_PRI_COMMENT                               \
    "–//XAPIA/CMC/PROPERTY//NONSGML PRI Comment//EN"
```

DESCRIPTION

This property provides supplementary information about the status of the message.

This is a **CMC_pv_string** type of property.

5.6.2 Delivery time

NAME

Per Recipient Information Delivery Time

C DECLARATION

```
#define CMC_PT_PRI_DELIVERY_TIME                               \
    "–//XAPIA/CMC/PROPERTY//NONSGML PRI Delivery Time//EN"
```

DESCRIPTION

This property specifies the date and time that the subject message was delivered.

This property is set by the service in **cmc_send_message_object()**.

This property is mandatory if the per recipient information type is CMC_PRI_DR.

This is a **CMC_pv_iso_date_time** type of property.

5.6.3 Diagnostic

NAME

Per Recipient Information Diagnostic

C DECLARATION

```
#define CMC_PT_PRI DIAGNOSTIC                                  \
    "–//XAPIA/CMC/PROPERTY//NONSGML PRI Diagnostic//EN"
```

DESCRIPTION

This property specifies the detailed diagnostic information indicating why the subject message was not delivered.

This is a **CMC_pv_string** type of property.

5.6.4 Object class

NAME

Per Recipient Information Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS                                    \
        "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as a Per Recipient Information.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OC_PER_RECIPIENT_INFORMATION which specifies that the object's class is a Per Recipient Information.

This is a **CMC_pv_enum** type of property.

5.6.5 Reason

NAME

Per Recipient Information Reason

C DECLARATION

```
#define CMC_PT_PRI_REASON                                      \
        "–//XAPIA/CMC/PROPERTY//NONSGML PRI Reason//EN"
```

DESCRIPTION

This property specifies the reason indicating why the per recipient information was generated.

This property is mandatory if the per recipient information type is CMC_PRI_NDR.

This is a **CMC_pv_string** type of property.

5.6.6 Recipient address

NAME

Per Recipient Information Recipient Address

C DECLARATION

```
#define CMC_PT_PRI_RECIPIENT_ADDRESS                           \
        "–//XAPIA/CMC/PROPERTY//NONSGML PRI Recipient Address//EN"
```

DESCRIPTION

This property specifies the address of the recipient intended for the subject message, who either received or was not able to receive the message, as indicated by the per recipient information type. This is not the originator of the subject message who would usually be the recipient of this report. The report recipient cannot reply to the report.

This is a **CMC_pv_string** type of property.

5.6.7 Recipient name

NAME

Per Recipient Information Recipient Name

C DECLARATION

```
   #define CMC_PT_PRI_RECIPIENT_NAME                              \
        "–//XAPIA/CMC/PROPERTY//NONSGML PRI Recipient Name//EN"
```

DESCRIPTION

This property specifies the name of the recipient intended for the subject message, who either received or was not able to receive the message, as indicated by the per recipient information type. This is not the originator of the subject message who would usually be the recipient of this report. The report recipient cannot reply to the report.

This is a **CMC_pv_string** type of property.

5.6.8 Type

NAME

Per Recipient Information Type

C DECLARATION

```
   #define CMC_PT_PRI_TYPE                                        \
        "–//XAPIA/CMC/PROPERTY//NONSGML PRI Type//EN"
```

DESCRIPTION

This property specifies the type of the per recipient information.

The valid values for this property include:

CMC_PRI_DR
CMC_PRI_NDR
CMC_PRI_UNKNOWN

CMC_PRI_DR – Specifies a delivery notice type of per recipient information.

CMC_PRI_NDR – Specifies a non-delivery notice type of per recipient information.

CMC_PRI_UNKNOWN – Specifies that the per recipient information type was not specified or not applicable.

This is a **CMC_pv_enum** type of property.

5.7 Profile container object properties

The profile container object identifies session context and configuration specific information. The following subclauses define, declare, and describe profile container properties.

5.7.1 Auto-Action

NAME

Profile Container Auto-Action

C DECLARATION

```
   #define CMC_PT_PROFILE_CONTAINER_AUTO_ACTION      \
        "–//NONSGML Profile Container Auto Action//EN"
```

DESCRIPTION

This property specifies the automatic action or disposition of the message after it is sent. Support for this property is optional for implementations conforming to this Recommendation. In the case of newly created messages, the property is created by a call to **cmc_add_properties()**. In the case of newly created messages, the property can also be modified by a call to **cmc_add_properties()** or deleted by a call to **cmc_delete_properties()**. This value can be overridden by the CMC_PT_MESSAGE_AUTO_ACTION property at the message object on a per message basis.

The valid value for this property is:

CMC_AA_DELETE

Set:	The specified message is to be deleted by the underlying messaging system after it has been successfully submitted for transfer.
Clear:	The specified message is placed in the sent folder if it exists. If not, the message is deleted.

This is a **CMC_pv_flags** type of property.

5.7.2 Character Set

NAME

Profile Container Character Set

C DECLARATION

```
#define CMC_PT_PROFILE_CHARACTER_SET           \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Profile Character Set//EN"

DESCRIPTION

The character set to be used for conveying string data between the user and CMC. The property value is an array of character set object identifiers associated with the implementation. The array is a counted array. The first character set identifier in the array is the default character set used if the caller does not specify one explicitly in the **cmc_logon()** function. Refer to the platform specific clause in B.2.4 for object identifiers for common character sets.

This is a **CMC_pv_array_of_guid** type of property.

5.7.3 Conformance

NAME

Profile Container Conformance

C DECLARATION

```
#define CMC_PT_PROFILE_CONF         \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Profile Conf//EN"

DESCRIPTION

The conformance level of the implementation. The property value will be either CMC_CONF_SIMPLE_CMC if the implementation supports only the Simple CMC, and CMC_CONF_FULL_CMC if the implementation supports a stand-alone version of the Full CMC. A value of CMC_CONF_FULL_CMC implies that the implementation also supports the Simple CMC interface as required by the conformance clause.

This is a **CMC_pv_enum** type of property.

5.7.4 Default Service

NAME

Profile Container Default Service

C DECLARATION

```
#define CMC_PT_PROFILE_DEFAULT_SERVICE         \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Profile Default Service//EN"

DESCRIPTION

The default service name. A pointer value of NULL will be written if no default service name is available. This property, along with the CMC_PT_PROFILE_DEFAULT_USER, can be used as defaults for the service name and user name for **cmc_logon()**. This will be returned in the implementation default character set.

This is a **CMC_pv_string** type of property.

5.7.5 Default User

NAME

Profile Container Default User

C DECLARATION

```
#define CMC_PT_PROFILE_DEFAULT_USER        \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Profile Default User//EN"

DESCRIPTION

The default CMC user name. A pointer value of NULL will be written if no default user name is available. This property, along with the CMC_PROFILE_DEFAULT_SERVICE, can be used as defaults for the provider name and user name for **cmc_logon()**. This will be returned in the implementation default character set.

This is a **CMC_pv_string** type of property.

5.7.6 Line Terminator

NAME

Profile Container Line Term

C DECLARATION

```
#define CMC_PT_PROFILE_LINE_TERM     \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Profile Line Term//EN"

DESCRIPTION

The line terminator characters to be used to terminate lines of strings. The values for the property are CMC_LINE_TERM_CRLF if the line delimiter is a carriage return followed by a line feed, CMC_LINE_TERM_LF if the line delimiter is a line feed, or CMC_LINE_TERM_CR if the line delimiter is a carriage return.

This is a **CMC_pv_enum** type of property.

5.7.7 Object Class

NAME

Profile Container Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS                              \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"

DESCRIPTION

This property defines the class of the object as address book.

The only valid value for this property is CMC_PT_OBJECT_CLASS_PROFILE, which specifies that the object's class is a profile container object.

This is a **CMC_pv_enum** type of property.

5.7.8 Object Extensions Supported

NAME

Profile Container Object Extensions Supported

C DECLARATION

```
#define CMC_PT_PROFILE_OBJECT_EXT    \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Profile Object Ext//EN"

DESCRIPTION

The object class extensions supported by the implementation. The property values are an array of the object class global identifiers for the object class extensions supported by the implementation. There is not an implicit order to the object GUIDs in the array.

This is a **CMC_pv_array_guid** type of property.

5.7.9 Objects Supported

NAME

Profile Container Objects Supported

C DECLARATION

```
#define CMC_PT_PROFILE_OBJECT_SUP    \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Object Sup//EN"
```

DESCRIPTION

The object classes supported by the implementation. The property values are an array of the object class global identifiers for the object classes supported by the implementation. There is not an implicit order to the object GUIDs in the array.

This is a **CMC_pv_array_guid** type of property.

5.7.10 Properties Supported

NAME

Profile Container Properties Supported

C DECLARATION

```
#define CMC_PT_PROFILE_PROP_SUP      \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Prop Sup//EN"
```

DESCRIPTION

The properties supported by the implementation. The property values are an array of the property global identifiers for the object properties supported by the implementation. There is not an implicit order to the property GUIDs in the array.

This is a **CMC_pv_array_guid** type of property.

5.7.11 Property Extensions Supported

NAME

Profile Container Properties Supported

C DECLARATION

```
#define CMC_PT_PROFILE_PROP_EXT      \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Prop Ext//EN"
```

DESCRIPTION

The property extensions supported by the implementation. The property values are an array of the property global identifiers for the object property extensions supported by the implementation. There is not an implicit order to the property GUIDs in the array.

This is a **CMC_pv_array_guid** type of property.

5.7.12 Required Password

NAME

Profile Container Required Password

C DECLARATION

```
#define CMC_PT_PROFILE_REQ_PASSWORD          \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Req Password//EN"
```

DESCRIPTION

Whether a password is required for log on to the service. The values of the property are CMC_REQUIRED_NO if the password is not required to log on, CMC_REQUIRED_OPT if the password is optional to log on, or CMC_REQUIRED_YES if the password is required to log on.

This is a **CMC_pv_enum** type of property.

5.7.13 Required Service

NAME

Profile Container Required Service

C DECLARATION

```
#define CMC_PT_PROFILE_REQ_SERVICE          \
        "–//XAPIA/CMC/PROPERTY//NONSGML Profile Req Service//EN"
```

DESCRIPTION

Whether a service name is required to log on to the service. The values of the property CMC_REQUIRED_NO if the service name is not required to log on, CMC_REQUIRED_OPT if the service name is optional to log on, or CMC_REQUIRED_YES if the service name is required to log on.

This is a **CMC_pv_enum** type of property.

5.7.14 Required User

NAME

Profile Container Required User

C DECLARATION

```
#define CMC_PT_PROFILE_REQ_USER      \
        "–//XAPIA/CMC/PROPERTY//NONSGML Profile Req User//EN"
```

DESCRIPTION

Whether a user name is required to log on to the service. The values of the property CMC_REQUIRED_NO if the user name is not required to log on, CMC_REQUIRED_OPT if the user name is optional to log on, or CMC_REQUIRED_YES if the user name is required to log on.

This is a **CMC_pv_enum** type of property.

5.7.15 Support Counted Strings

NAME

Profile Container Support Counted Strings

C DECLARATION

```
#define CMC_PT_PROFILE_SUP_COUNTED_STR       \
        "–//XAPIA/CMC/PROPERTY//NONSGML Profile Sup Counted Str//EN"
```

DESCRIPTION

Whether the service supports counted strings. The property value will be set to a true value if the CMC_COUNTED_STRING_TYPE flag is supported during log on.

This is a **CMC_pv_boolean** type of property.

5.7.16 Support No Mark As Read

NAME

Profile Container Support No Mark As Read

C DECLARATION

```
#define CMC_PT_PROFILE_SUP_NOMKMSGREAD       \
        "–//XAPIA/CMC/PROPERTY//NONSGML Profile Sup NoMkMsgRead//EN"
```

DESCRIPTION

Whether the service supports the **cmc_read()** CMC_DO_NOT_MARK_AS_READ operation. The property value will be set to a true value if the CMC_DO_NOT_MARK_AS_READ flag is supported by **cmc_read()**.

This is a **CMC_pv_boolean** type of property.

5.7.17 User Interface Available

NAME

Profile Container User Interface Available

C DECLARATION

```
#define CMC_PT_PROFILE_UI_AVAIL      \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile UI Avail//EN"
```

DESCRIPTION

Whether a user interface is available for parameter entry and resolution. The property value will be set to a true value if there is UI provided by the CMC implementation.

This is a **CMC_pv_boolean** type of property.

5.7.18 Users

NAME

Profile Container Users

C DECLARATION

```
#define CMC_PT_PROFILE_USERS         \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Users//EN"
```

DESCRIPTION

The users currently logged on to the root container. The property values are the recipient names of the users logged on to the root container. Support for the property is optional for implementations conforming to this Recommendation.

There is no implicit order to the subsequent recipient names in the array.

This is a **CMC_pv_array_string** type of property.

5.7.19 Version of the Implementation

NAME

Profile Container Version of the Implementation

C DECLARATION

```
#define CMC_PT_PROFILE_VER_IMPLEM    \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Ver Implem//EN"
```

DESCRIPTION

The version of the implementation. The property value will be set to the version number of the implementation, multiplied by 100. For example, version 1.01 will return 101.

This is a **CMC_pv_uint16** type of property.

5.7.20 Version of the Specification

NAME

Profile Container Version of the Specification

C DECLARATION

```
#define CMC_PT_PROFILE_VER_SPEC      \
    "–//XAPIA/CMC/PROPERTY//NONSGML Profile Ver Spec//EN"
```

DESCRIPTION

The version of the CMC specification supported by the implementation. The property value will be set to the version of the CMC specification supported by the implementation, multiplied by 100. For example, version 1.00 will return 100.

This is a **CMC_pv_uint16** type of property.

5.8 Recipient object properties

Recipient objects identify individual users within a messaging service. A recipient object is not a container object. The following subclauses define, declare, and describe recipient object properties.

5.8.1 Address

NAME

Recipient Address

C DECLARATION

```
#define CMC_PT_RECIPIENT_ADDRESS     \
        "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Address//EN"
```

DESCRIPTION

This property specifies the address of the recipient. The format of the string is implementation-dependent.

In gateway applications, the Address of the Recipient Object whose role is Originator may be added by the gateway.

This is a **CMC_pv_string** type of property.

5.8.2 Content Return Requested

NAME

Recipient Content Return Requested

C DECLARATION

```
#define CMC_PT_RECIPIENT_CONTENT_RETURN_REQUESTED                  \
        "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Content Return Requested//EN"
```

DESCRIPTION

This property specifies whether the subject message should be returned with the non-delivery report in case of unsuccessful delivery. If the report is not requested, the message will not be returned regardless of the indication of this property.

This is a **CMC_pv_boolean** type of property.

5.8.3 Name

NAME

Recipient Name

C DECLARATION

```
#define CMC_PT_RECIPIENT_NAME        \
        "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Name//EN"
```

DESCRIPTION

This property specifies the display name of the recipient. The format of the string is implementation-dependent.

This is a **CMC_pv_string** type of property.

5.8.4 Object Class

NAME

Recipient Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
        "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as a recipient.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_RECIPIENT, which specifies that the object's class is a recipient.

This is a **CMC_pv_enum** type of property.

5.8.5 Receipt Requested

NAME

Recipient Receipt Requested

C DECLARATION

```
#define CMC_PT_RECIPIENT_RECEIPT_REQUESTED                        \
    "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Receipt Requested//EN"
```

DESCRIPTION

The type of the receipt to be returned for the message.

If both the message object and the recipient object specify the receipt requested property, the value specified at the recipient object will override the value specified at the message object. The implementation is not required to support this property at the recipient object level.

The valid values for this property include:

CMC_RECEIPT_RN
CMC_RECEIPT_NRN
CMC_RECEIPT_BOTH
CMC_RECEIPT_NONE

CMC_RECEIPT_RN – Requests that a receipt notification is returned only when the recipient has received the subject message.

CMC_RECEIPT_NRN – Requests that a non-receipt notification is returned only when the recipient has failed to receive the subject message.

CMC_RECEIPT_BOTH – Requests that either a receipt notification or a non-receipt notification is returned when the recipient has received or fails to receive the subject message.

CMC_RECEIPT_NONE – Requests that no receipt should be returned regardless of whether the recipient has received or fails to receive the subject message.

This is a **CMC_pv_enum** type of property.

5.8.6 Report Requested

NAME

Recipient Report Requested

C DECLARATION

```
#define CMC_PT_RECIPIENT_REPORT_REQUESTED                    \
    "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Report Requested//EN"
```

DESCRIPTION

The type of the report to be returned for the subject message. This is used to indicate whether the subject message has been delivered or not-delivered by the underlying messaging transport systems.

The valid values for this property include:

CMC_REPORT_DR
CMC_REPORT_NDR
CMC_REPORT_BOTH
CMC_REPORT_NONE

CMC_REPORT_DR – Specifies that a delivery report is requested.

CMC_REPORT_NDR – Specifies that a non-delivery report is requested.

CMC_REPORT_BOTH – Specifies that either delivery report, or non-delivery report is requested, whichever applicable.

CMC_REPORT_NONE – Specifies that neither delivery report nor non-delivery report is requested.

This is a **CMC_pv_enum** type of property.

5.8.7 Responsibility Flag

NAME

Recipient Responsibility Flag

C DECLARATION

```
#define CMC_PT_RECIPIENT_RESPONSIBILITY_FLAG         \
    "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Responsibility Flag//EN"
```

DESCRIPTION

This property specifies an indicator of whether this recipient should receive a copy of the message. It is useful in gateways and situations where multiple versions of CMC may be accessed by an application.

The default for this property is CMC_TRUE.

This is a **CMC_pv_boolean** type of property.

5.8.8 Role

NAME

Recipient Role

C DECLARATION

```
#define CMC_PT_RECIPIENT_ROLE       \
    "–//XAPIA/CMC/PROPERTY//NONSGML Recipient Role//EN"
```

DESCRIPTION

This property specifies the role of the recipient.

The valid values for this property include:

CMC_RECIPIENT_ROLE_TO
CMC_RECIPIENT_ROLE_CC
CMC_RECIPIENT_ROLE_BCC
CMC_RECIPIENT_ROLE_ORIGINATOR
CMC_RECIPIENT_ROLE_AUTHORIZING_USER
CMC_RECIPIENT_ROLE_REPLY_TO
CMC_RECIPIENT_ROLE_FORWARDED
CMC_RECIPIENT_ROLE_ACTUAL
CMC_RECIPIENT_ROLE_INTENDED

CMC_RECIPIENT_ROLE_TO – Specifies the primary recipient.

CMC_RECIPIENT_ROLE_CC – Specifies the carbon copy recipient.

CMC_RECIPIENT_ROLE_BCC – Specifies the blind carbon copy recipient.

CMC_RECIPIENT_ROLE_ORIGINATOR – Specifies the originator.

CMC_RECIPIENT_ROLE_AUTHORIZING_USER – Specifies the authorizing user.

CMC_RECIPIENT_ROLE_REPLY_TO – Specifies recipient to which the reply should be directed.

CMC_RECIPIENT_ROLE_FORWARDED – Specifies the forwarded recipient.

CMC_RECIPIENT_ROLE_ACTUAL – Specifies the actual recipient.

CMC_RECIPIENT_ROLE_INTENDED – Specifies the intended recipient.

This is a **CMC_pv_enum** type of property.

5.8.9 Type

NAME

Recipient Type

C DECLARATION

```
#define CMC_PT_RECIPIENT_TYPE        \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Recipient Type//EN"

DESCRIPTION

This property specifies the type of the recipient.

The valid values for this property include:

CMC_RCT_UNKNOWN (=0)
CMC_RCT_INDIVIDUAL
CMC_RCT_GROUP
CMC_RCT_REPORT_RECIPIENT

CMC_RCT_UNKNOWN – Specifies an unknown recipient type.

CMC_RCT_INDIVIDUAL – Specifies the recipient as an individual.

CMC_RCT_GROUP – Specifies that the recipient is a distribution list.

CMC_RCT_REPORT_RECIPIENT – Specifies that the recipient is the recipient of a report message.

This is a **CMC_pv_enum** type of property.

5.9 Report object properties

The report object is a collection of report specific object properties. The following subclauses define, declare, and describe report object properties.

5.9.1 Application Id

NAME

Report Application Id

C DECLARATION

```
#define CMC_PT_REPORT_APPLICATION_ID \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Application Id//EN"

DESCRIPTION

This property specifies a globally unique identifier for the report. This property is set by the application.

This is a **CMC_pv_string** type of property.

5.9.2 Id

NAME

Report Id

C DECLARATION

```
#define CMC_PT_REPORT_ID     \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Id//EN"

DESCRIPTION

This property specifies a globally unique identifier for the report. This property is set by **cmc_send_message_object()**, is defined by the messaging service (established at submission), and is unique within the domain.

In gateway applications, the Message Id may be added or updated by the caller.

This is a **CMC_pv_guid** type of property.

5.9.3 Item Count

NAME

Report Item Count

C DECLARATION

```
#define CMC_PT_REPORT_ITEM_COUNT                                \
    "–//XAPIA/CMC/PROPERTY//NONSGML Report Item Count//EN"
```

DESCRIPTION

This property specifies the number of top-level content items contained in a report. This count does not include content items nested in other content items, or messages. This property is set by the implementation.

This is a **CMC_pv_uint32** type of property.

5.9.4 Messaging System Id

NAME

Report Messaging System Id

C DECLARATION

```
#define CMC_PT_REPORT_MESSAGING_SYSTEM_ID                         \
    "–//XAPIA/CMC/PROPERTY//NONSGML Report Messaging System Id//EN"
```

DESCRIPTION

This property specifies the underlying message transport system identifier or the gateway identifier that created this report.

This is a **CMC_pv_enum** type of property.

5.9.5 Object Class

NAME

Report Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
    "–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"
```

DESCRIPTION

This property defines the class of the object as a report.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_REPORT which specifies that the object's class is a report.

This is a **CMC_pv_enum** type of property.

5.9.6 Read

NAME

Report Read

C DECLARATION

```
#define CMC_PT_REPORT_READ                                               \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Read//EN"

DESCRIPTION

This property specifies whether the report has been read.

This is a **CMC_pv_boolean** type of property.

5.9.7 Size

NAME

Report Size

C DECLARATION

```
#define CMC_PT_REPORT_SIZE                                              \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Size//EN"

DESCRIPTION

This property specifies the size of the report.

This is a **CMC_pv_uint32** type of property.

5.9.8 Subject

NAME

Report Subject

C DECLARATION

```
#define CMC_PT_REPORT_SUBJECT                                           \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Subject//EN"

DESCRIPTION

This property states the subject of the report. This property is defaultable to a NULL string.

This is a **CMC_pv_string** type of property.

5.9.9 Subject Message Id

NAME

Report Subject Message Id

C DECLARATION

```
#define CMC_PT_REPORT_SUBJECT_MESSAGE_ID                                    \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Subject Message Id//EN"

DESCRIPTION

This property identifies the user message that caused this report to be generated.

The property value may be a textual reference or may be a textual approximation of the message identifier of the previous correspondence.

This is a **CMC_pv_string** type of property.

5.9.10 Time Received

NAME

Report Time Received

C DECLARATION

```
#define CMC_PT_REPORT_TIME_RECEIVED                                         \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Report Time Received//EN"

DESCRIPTION

This property specifies the date and time that the report was received.

This is a **CMC_pv_iso_date_time** type of property.

5.9.11 Time Sent

NAME

Report Time Sent

C DECLARATION

```
#define CMC_PT_REPORT_TIME_SENT                                \
        "–//XAPIA/CMC/PROPERTY//NONSGML Report Time Sent//EN"
```

DESCRIPTION

This property specifies the date and time that the report was sent.

This property is set by the service in **cmc_send_message_object**.

This is a **CMC_pv_iso_date_time** type of property.

5.9.12 Unsent

NAME

Report Unsent

C DECLARATION

```
#define CMC_PT_REPORT_UNSENT                                   \
        "–//XAPIA/CMC/PROPERTY//NONSGML Report Unsent//EN"
```

DESCRIPTION

This property specifies that the report has not been sent.

This is a **CMC_pv_boolean** type of property.

5.10 Root container object properties

The root is the essential core container object composed of various properties and other container objects. The root container is composed of address books (containing recipients and other address books), a profile container, and message containers. The recipient object within the root container cannot be modified.

The following subclauses define, declare, and describe root container object properties.

5.10.1 Child Allowed

NAME

Root Container Child Allowed

C DECLARATION

```
#define CMC_PT_ROOT_CONTAINER_CHILD_ALLOWED          \
        "–//XAPIA/CMC/PROPERTY//NONSGML Root Container Child Allowed//EN"
```

DESCRIPTION

This property permits or denies the existence of a child of the root container.

This is a **CMC_pv_boolean** type of property.

5.10.2 Comment

NAME

Root Container Comment

C DECLARATION

```
#define CMC_PT_ROOT_CONTAINER_COMMENT      \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Root Container Comment//EN"

DESCRIPTION

This property provides a descriptive comment about the root container.

This is a **CMC_pv_string** type of property.

5.10.3 Location

NAME

Root Container Location

C DECLARATION

```
#define CMC_PT_ROOT_CONTAINER_LOCATION     \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Root Container Location//EN"

DESCRIPTION

This property specifies the location of the root container.

The valid values for this property include:

CMC_ROOT_CONTAINER_LOCATION_LOCAL
CMC_ROOT_CONTAINER_LOCATION_SERVER
CMC_ROOT_CONTAINER_LOCATION_UNKNOWN

CMC_ROOT_CONTAINER_LOCATION_LOCAL – Specifies that the location of the root container is local and not on the messaging server.

CMC_ROOT_CONTAINER_LOCATION_SERVER – Specifies that the location of the root container is on the messaging server.

CMC_ROOT_CONTAINER_LOCATION_UNKNOWN – Specifies that the location of the root container is unknown.

This is a **CMC_pv_enum** type of property.

5.10.4 Name

NAME

Root Container Name

C DECLARATION

```
#define CMC_PT_ROOT_CONTAINER_NAME         \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Root Container Name//EN"

DESCRIPTION

This property specifies the name of the root container.

This is a **CMC_pv_string** type of property.

5.10.5 Object Class

NAME

Root Container Object Class

C DECLARATION

```
#define CMC_PT_OBJECT_CLASS          \
```
"–//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"

DESCRIPTION

This property defines the class of the object as the root.

This property is created by **cmc_open_object_handle()**.

The only valid value for this property is CMC_PT_OBJECT_CLASS_ROOT, which specifies that the object's class is the root.

This is a **CMC_pv_enum** type of property.

5.10.6 Shared

NAME

Root Container Shared

C DECLARATION

```
#define CMC_PT_ROOT_CONTAINER_SHARED       \
    "-//XAPIA/CMC/PROPERTY//NONSGML Root Container Shared//EN"
```

DESCRIPTION

This property specifies whether the root container is shared with another entity.

This is a **CMC_pv_boolean** type of property.

6 Functional interface

This clause defines the functions of the Common Messaging Call interface. The functions of both the generic and C interfaces are specified. Those of the C interface are repeated in Annex A, "C declaration summary".

6.1 Simple CMC functions

Simple CMC offers a basic set of functions that are intended to provide messaging-aware capabilities for messaging-enabled applications. These functions were previously published as CMC Version 1.0. Table 14 lists the functions of the Simple CMC interface.

Table 14/X.446 – Simple CMC Interface Functions

Function	Description
Sending messages	
Send	Send a mail message
Send documents	String-based function to send mail
Receiving messages	
Act on	Perform an action on a specified message
List	List summary information about messages meeting specified criteria
Read	Read and return a specified message
Looking up names	
Look up	Looks up addressing information
Administration	
Free	Free memory allocated by the messaging service
Log off	Terminate a session with the messaging service
Log on	Establish a session with the messaging service
Query configuration	Determine information about the installed CMC service

The manual pages for these functions are given in subsequent pages.

6.1.1 Sending messages

6.1.1.1 Send

NAME

Send – Send a mail message.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_send(
   CMC_session_id        session,
   CMC_message           *message,
   CMC_flags             send_flags,
   CMC_ui_id             ui_id,
   CMC_extension         *send_extensions
);
```

DESCRIPTION

This function sends a mail message.

The caller can optionally provide a list of recipients, subject text, attachments and/or note text. If one or more recipients are provided, the function can send the message.

The successful return of this function does not necessarily imply the validation of recipients.

ARGUMENTS

Session (Session Id)

Opaque session id which represents a session with the messaging service.

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

If the session id is invalid, and a valid session is not created through UI, then the error CMC_E_INVALID_SESSION_ID is returned.

Message (Message)

Message structure containing the contents of the message to be sent. If the extension argument flag CMC_X_SEND_UI_REQUESTED is not set or supported, there must be at least one recipient of type TO, CC, or BCC.

All other fields are optional. The time_sent and message_reference fields are ignored.

The following conditions on the message structure fields apply:

Recipients – The number of recipients per message may be limited in some services. If the limit is exceeded, the error CMC_E_TOO_MANY_RECIPIENTS is returned. If zero recipients are specified, a pointer value of NULL should be assigned to *recipients*.

The recipient descriptor can include either the recipient's name, an address, or name/address pair. If just a name is specified, the name is resolved to an address using implementation-defined name resolution rules. If just an address is specified, then this address is used for delivery and for the recipient display name. If both an address and a name are specified, a resolution of the name should not be performed. If an implementation cannot support both names and addresses, then the name is ignored. The address is in an implementation-defined format and is assumed to have been obtained from the implementation using some other means. A recipient of type originator is not required for send; if present, its action is defined by the CMC implementation.

Attachments – The number of attachments per message may be limited in some services. If the limit is exceeded, the CMC_E_TOO_MANY_FILES is returned. A pointer value of NULL indicates no attachments. The attachment files are read before the **cmc_send()** function returns, so that the files may be freely changed or deleted without affecting the message.

Subject **–** A pointer value of NULL indicates no subject text. Some implementations may truncate subject lines which are too long or contain carriage returns/line feeds/form feeds.

Note Text **–** A pointer value of NULL indicates no text. Implementations may place limits on the size of the text. If the note text exceeds the limit of the service, it may demote the body text to an attachment or generate the error CMC_E_TEXT_TOO_LARGE.

Message Type – Pointer to a string which is the message type. The type specifies the type of message being sent (see description of Message data structure for details). To specify an interpersonal message, the string "CMC: IPM" is used. If a pointer value of NULL or a pointer to an empty string is given, the value "CMC: IPM" is assumed.

Flags – The following flag may be used when sending a message:

CMC_MSG_TEXT_NOTE_AS_FILE.

All other flags will be ignored. For more information on these flags, see the description of the message structure.

Send Flags (Flags)

Bit mask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_LOGON_UI_ALLOWED

CMC_SEND_UI_REQUESTED

CMC_ERROR_UI_ALLOWED

CMC_COUNTED_STRING_TYPE

CMC_LOGON_UI_ALLOWED – Set if the function should display a dialogue box to prompt for logon if required. If not set, the function will not display a dialogue box and will return the error CMC_E_INVALID_SESSION_ID if the user is not logged on.

CMC_SEND_UI_REQUESTED – Set if the function should display a dialogue box to prompt for recipients, the message fields, and other sending options. If not set, the function will not display a dialogue box, but at least one recipient must be specified.

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

CMC_COUNTED_STRING_TYPE – Set if the string type used in the message is counted string. If not set, the strings are assumed to be null terminated. If the session parameter is valid, this flag is ignored.

UI Identifier (UI Id)

User Interface handle (e.g. dialogue window) for use in resolving any questions which arise when the service performs the function, in prompting the user for additional information, or in verifying or acknowledging information which has been provided.

Ignored if UI is not supported by the CMC implementation.

Send Extensions (Extensions)

A pointer to an array of CMC_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

Send 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)

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

CMC_E_ATTACHMENT_NOT_FOUND
CMC_E_ATTACHMENT_OPEN_FAILURE
CMC_E_ATTACHMENT_READ_FAILURE
CMC_E_ATTACHMENT_WRITE_FAILURE
CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_FLAG
CMC_E_INVALID_MESSAGE_PARAMETER
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_SESSION_ID
CMC_E_INVALID_UI_ID
CMC_E_LOGON_FAILURE
CMC_E_RECIPIENT_NOT_FOUND
CMC_E_TEXT_TOO_LARGE
CMC_E_TOO_MANY_FILES
CMC_E_TOO_MANY_RECIPIENTS
CMC_E_UNSUPPORTED_DATA_EXT
CMC_E_UNSUPPORTED_FLAG
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_USER_CANCEL
CMC_E_USER_NOT_LOGGED_ON

6.1.1.2 Send Documents

NAME

Send Documents – String-based function to send mail.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_send_documents(
   CMC_string          recipient_addresses,
   CMC_string          subject,
   CMC_string          text_note,
   CMC_flags           send_doc_flags,
   CMC_string          file_paths,
   CMC_string          attach_titles,
   CMC_string          delimiter,
   CMC_ui_id           ui_id
);
```

DESCRIPTION

This function sends a mail message. This function is primarily intended for calling from a "scripting" language (e.g. spreadsheet macro) that cannot handle data structures.

This function will try to establish a session without logon UI. If this is not possible, it will prompt for logon information to establish a session. The session is always closed on completion.

ARGUMENTS

Recipient Addresses (String)

Pointer to a string containing the recipient addresses for the message. When multiple recipients are specified, they should be separated by the Delimiter character. Recipients are assumed to be primary recipients unless prefixed by "cc:" or "bcc:" for copy recipients and blind copy recipients. The prefix "to:" may also be used for consistency. A pointer value of NULL indicates that recipients should be prompted for in a dialogue.

Subject (String)

Pointer to a string containing the subject of a message. A pointer value of NULL indicates no subject text.

Text Note (String)

Pointer to a string containing the note text to be carried with the message. A pointer value of NULL indicates no note text.

Send Doc Flags (Flags)

Bit mask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_COUNTED_STRING_TYPE

CMC_FIRST_ATTACH_AS_TEXT_NOTE

CMC_COUNTED_STRING_TYPE – Set if the string type used in the message is counted string. If not set, the strings are assumed to be null terminated.

CMC_FIRST_ATTACH_AS_TEXT_NOTE – Set if the first attachment should be sent as the message text note. If not set, the text note is contained in the text note field.

File Paths (String)

Pointer to a string containing the actual path names for the attachment files. When multiple path names are specified, they should be separated by the Delimiter character.

Attach Titles (String)

Pointer to a string containing the attachment titles to be seen by the recipient. When multiple names are specified, they should be separated by the Delimiter character.

Delimiter (String)

Pointer to a character that is used to delimit the names in the File Paths, File Names, and Recipient Addresses strings. This character should be chosen to be one not used in operating system file names or recipient names. This parameter cannot be NULL.

UI Identifier (UI Id)

Pointer to an identifier for a User Interface (e.g. dialogue window) for use in resolving any questions which might otherwise result in an error and queries the user for additional information as required.

Ignored if UI is not supported by the CMC implementation.

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

CMC_E_ATTACHMENT_NOT_FOUND
CMC_E_ATTACHMENT_OPEN_FAILURE
CMC_E_ATTACHMENT_READ_FAILURE
CMC_E_ATTACHMENT_WRITE_FAILURE
CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_FLAG
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_UI_ID
CMC_E_LOGON_FAILURE
CMC_E_RECIPIENT_NOT_FOUND
CMC_E_TEXT_TOO_LARGE
CMC_E_TOO_MANY_FILES
CMC_E_TOO_MANY_RECIPIENTS
CMC_E_UNSUPPORTED_FLAG
CMC_E_USER_CANCEL
CMC_E_USER_NOT_LOGGED_ON

6.1.2 Receiving messages

6.1.2.1 Act On

NAME

Act On – Perform an action on a specified message.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_act_on(
   CMC_session_id            session,
   CMC_message_reference     *message_reference,
   CMC_enum                  operation,
   CMC_flags                 act_on_flags,
   CMC_ui_id                 ui_id,
   CMC_extension             *act_on_extensions
);
```

DESCRIPTION

This function performs the action specified on the message indicated by the message_reference.

ARGUMENTS

Session (Session Id)

Opaque session id which represents a session with the messaging service.

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

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Message Reference (Message Reference)

Specifies the message reference of the message to be acted upon.

If the message reference is invalid (or no longer valid, such as after it has been deleted), then the error CMC_E_INVALID_MESSAGE_REFERENCE is returned. NULL message reference pointers and message references of length zero are considered invalid for operations that require this parameter.

Operation (Enum)

The operation to perform on the message. Valid operations include:

CMC_ACT_ON_EXTENDED	(= 0)
CMC_ACT_ON_DELETE	

CMC_ACT_ON_EXTENDED – Look in the list of extensions for the action to carry out.

CMC_ACT_ON_DELETE – Action requested is to delete the specified message from mailbox. This operation requires a valid message reference parameter.

Act On Flags (Flags)

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

CMC_ERROR_UI_ALLOWED

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

UI Id (UI Id)

User Interface handle (e.g. dialogue window) for use in resolving any questions which might otherwise result in an error.

Ignored if UI is not supported by the CMC implementation.

Act On Extensions (Extension)

A pointer to an array of CMC_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

Act On 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

CMC_E_FAILURE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_INVALID_ENUM

CMC_E_INVALID_FLAG

CMC_E_INVALID_MESSAGE_REFERENCE

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_SESSION_ID

CMC_E_INVALID_UI_ID

CMC_E_MESSAGE_IN_USE

CMC_E_UNSUPPORTED_ACTION

CMC_E_UNSUPPORTED_FLAG

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.1.2.2 List

NAME

List – List summary information about messages which meet a specified criteria.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_list(
   CMC_session_id             session,
   CMC_string                 message_type,
   CMC_flags                  list_flags,
   CMC_message_reference      *seed,
   CMC_uint32                 *count,
   CMC_ui_id                  ui_id,
   CMC_message_summary        **result,
   CMC_extension              *list_extensions
);
```

DESCRIPTION

This function lists summary information, including a message reference, about messages which meet the specified criteria. Using the returned message reference(s), the message(s) may be further processed using **cmc_read()** and **cmc_act_on()**.

Optional criteria include:

– the message is of a specified message type; and

– the message is as yet unread.

The search begins after a specified "seed" message reference, or at the beginning of the mailbox. A maximum number of messages to list can be specified. The function returns the actual number of messages returned.

Optionally, each message summary returned in "result" can include only the message reference.

ARGUMENTS

Session (Session Id)

Opaque session id which represents a session with the messaging service.

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

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Message Type (String)

Information is returned only for messages of the specified type. If the type is not recognized, the error CMC_E_UNRECOGNIZED_MESSAGE_TYPE will be returned. The format of the Message Type string is given in 5.4.23.

A NULL indicates that information should be returned for all available messages.

List Flags (Flags)

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

CMC_ERROR_UI_ALLOWED
CMC_LIST_UNREAD_ONLY
CMC_LIST_MSG_REFS_ONLY
CMC_LIST_COUNT_ONLY

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

CMC_LIST_UNREAD_ONLY – If set, list should include only unread messages. If not set, list may include both read and unread messages.

CMC_LIST_MSG_REFS_ONLY – If set, only Message Reference is populated in the result structure. Values of other fields are undefined, and should be ignored. If not set, all information in the result structure is returned.

CMC_LIST_COUNT_ONLY – If set, the function should not return any summary structures, only the count of messages meeting the specified criteria. If not set, summary structures will be returned.

Seed (Message Reference)

Specifies the message reference of the message after which the search should begin. If the message reference is invalid (or no longer valid, such as after it has been deleted), then the error CMC_E_INVALID_MESSAGE_REFERENCE is returned.

A NULL message reference seed pointer indicates that the search should start with the first message in the mailbox.

Count (Uint32)

Specifies the maximum number of messages to return. A value of zero specifies no maximum.

UI Id (UI Id)

User Interface handle (e.g. dialogue window) for use in resolving any questions which might otherwise result in an error.

Ignored if UI is not supported by the CMC implementation.

List Extensions (Extension)

A pointer to an array of CMC_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

Count (Uint32)

Specifies the number of messages actually returned. If no messages match the criteria, or if the mailbox is empty, a value of zero is returned.

Result (Message Summary)

The "result" field is the address at which an array of CMC_message_summary structures is to be returned. This array of structures is allocated by the service, and the entire array should be freed with a single call to **cmc_free()**.

The message reference field contained in each CMC_message_summary may be used to identify messages in subsequent calls to **cmc_read()** and **cmc_act_on()**.

NOTE – The message reference field may need to be copied prior to invoking **cmc_free()** on this structure.

If the CMC_LIST_MSG_REFS_ONLY flag has been set, the CMC_message_summary structures will return only message references. Values of other fields are undefined, and should be ignored.

List 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

CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_FLAG
CMC_E_INVALID_MESSAGE_REFERENCE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_SESSION_ID
CMC_E_INVALID_UI_ID
CMC_E_UNRECOGNIZED_MESSAGE_TYPE
CMC_E_UNSUPPORTED_FLAG
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.1.2.3 Read

NAME

Read – Read and return a specified message.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_read(
   CMC_session_id             session,
   CMC_message_reference      *message_reference,
   CMC_flags                  read_flags,
   CMC_message                **message,
   CMC_ui_id                  ui_id,
   CMC_extension              *read_extensions
);
```

DESCRIPTION

This function returns a message structure containing the data from the message indicated by the specified message reference. Optionally, the message structure returned can include only the message and attachment headers.

If the flag CMC_MSG_TEXT_NOTE_AS_FILE is set in the returned message structure, then the text note field is contained in the file referred to by the first attachment.

For systems that can mark messages as read, a message will have the state "READ" after this function successfully executes, unless the flag CMC_DO_NOT_MARK_AS_READ is set.

ARGUMENTS

Session (Session Id)

Opaque session id which represents a session with the messaging service.

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

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Message Reference (Message Reference)

Specifies the message reference of the message to be read and returned. If the message reference is invalid (or no longer valid, such as after it has been deleted), then the error CMC_E_INVALID_MESSAGE_REFERENCE is returned.

A NULL message reference pointer indicates that the first message in the mailbox should be read and returned.

Read Flags (Flags)

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

CMC_ERROR_UI_ALLOWED

CMC_MSG_AND_ATT_HDRS_ONLY

CMC_DO_NOT_MARK_AS_READ

CMC_READ_FIRST_UNREAD_MESSAGE

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

CMC_MSG_AND_ATT_HDRS_ONLY – If set, the attachments[n].attach_filename fields will be undefined when **cmc_read()** returns, and should be ignored. This may be useful to reduce the amount of data transferred. If clear, the attachment_filename fields will be returned normally.

NOTE – If CMC_MSG_TEXT_NOTE_AS_FILE is set in the message to indicate that the text note is stored in the first attachment, the attachment_filename field will be returned for that attachment regardless of the setting of this flag.

CMC_DO_NOT_MARK_AS_READ – If set, the state of the message is not changed to read when the function is returned. This will also suppress sending of a Receipt Report. The implementation can be queried to see if it supports this feature with the CMC_CONFIG_SUP_NOMKMSGREAD in **cmc_query_config()**.

CMC_READ_FIRST_UNREAD_MESSAGE – This is only available when passing a NULL message reference to receive the first message in the mailbox. If set, the first message not marked as read should be returned. If not set, the first message in the mailbox should be returned, whether it is marked as read or not.

UI Id (UI Id)

User Interface handle (e.g. dialogue window) for use in resolving any questions which arise when the service performs the function.

Ignored if UI is not supported by the CMC implementation.

Read Extensions (Extension)

A pointer to an array of CMC_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

Message (Message)

The "message" field is the address at which a pointer to a CMC_message structure is to be returned. This structure is allocated by the service, and should be freed with **cmc_free()**.

Attachment data will be returned in files, and the CMC_message structure will indicate the names of those files.

If the CMC_MSG_AND_ATT_HDRS_ONLY flag has been set (see "flags"), the CMC_message structure will not return the attachment files as described above.

Read 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

CMC_E_ATTACHMENT_OPEN_FAILURE
CMC_E_ATTACHMENT_READ_FAILURE
CMC_E_ATTACHMENT_WRITE_FAILURE
CMC_E_DISK_FULL
CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_FLAG
CMC_E_INVALID_MESSAGE_REFERENCE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_SESSION_ID
CMC_E_INVALID_UI_ID
CMC_E_TOO_MANY_FILES
CMC_E_UNABLE_TO_NOT_MARK_READ
CMC_E_UNSUPPORTED_FLAG
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.1.3 Looking up names

6.1.3.1 Look Up

NAME

Look Up – Look up addressing information in the directory.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_look_up(
   CMC_session_id          session,
   CMC_recipient           *recipient_in,
   CMC_flags               look_up_flags,
   CMC_ui_id               ui_id,
   CMC_uint32              *count,
   CMC_recipient           **recipient_out,
   CMC_extension           *look_up_extensions
);
```

DESCRIPTION

This function looks up addressing information in the directory provided by the CMC messaging service. It primarily is used to resolve a friendly name to an address.

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

ARGUMENTS

Session (Session Id)

Opaque session id which represents a session with the messaging service.

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

If the session id is invalid and a valid session is not created through UI, then the error CMC_E_INVALID_SESSION_ID is returned.

Recipient In (Recipient)

For name resolution, the name field in the structure contains the name to be resolved. The name type can be set to provide information on desired resolution of the name. See the recipient structure documentation for possible types.

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

For displaying UI to create addressing lists, this will point to an array of recipients that is terminated with the CMC_RECIP_LAST_ELEMENT flag. The list of recipients will be used as the defaults for displaying in the address list UI.

For both name resolution and displaying recipient details, all recipient 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:

CMC_LOGON_UI_ALLOWED

CMC_ERROR_UI_ALLOWED

CMC_COUNTED_STRING_TYPE

CMC_LOOKUP_RESOLVE_PREFIX_SEARCH

CMC_LOOKUP_RESOLVE_IDENTITY

CMC_LOOKUP_RESOLVE_UI

CMC_LOOKUP_DETAILS_UI

CMC_LOOKUP_ADDRESSING_UI

CMC_LOGON_UI_ALLOWED – Set if the function should display a dialogue box to prompt for logon if required. If not set, the function will not display a dialogue box and will return the error CMC_E_INVALID_SESSION_ID if the user is not logged on.

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

CMC_COUNTED_STRING_TYPE – Set if the string type used in the call parameters is counted. If this is not set, the strings are assumed to be null terminated. If the session parameter is valid, this flag is ignored.

CMC_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. CMC implementations are required to support simple prefix searching. The availability of wild-card or substring searches is optional.

CMC_LOOKUP_RESOLVE_IDENTITY – If set, the function will return a recipient record for the identity of the user in the mail 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.

CMC_LOOKUP_RESOLVE_UI – Set if the CMC implementation should attempt to disambiguate names by presenting a name resolution dialogue to the user. If this flag is not set, resolutions which do not result in a single name will return the error CMC_E_AMBIGUOUS_RECIPIENT 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.

CMC_LOOKUP_DETAILS_UI – If set, the function will display details UI for the recipient pointed to in recipient_in. This will only act on the first recipient in the list. If the name resolves to more than one address, this will not be carried out and the error CMC_E_AMBIGUOUS_RECIPIENT will be returned.

CMC_LOOKUP_ADDRESSING_UI – If set, the function will display UI to allow creation of a recipient list for addressing a message and general directory browsing. The recipient list passed to the function will be the original recipient list for the UI. The function will return the list of recipients selected by the user. This flag is optional for implementations to support.

UI Id (UI Id)

User Interface handle (e.g. dialogue window) for use in resolving any questions which arise when the service performs the function.

Ignored if UI is not supported by the CMC implementation.

Count (Uint32)

Specifies the maximum number of names to return. A value of 0 specifies no maximum. The value will be returned in the location pointed to by this parameter. A valid pointer to a location for the returned count information is required.

Look Up Extensions (Extension)

A pointer to an array of CMC_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

Recipient Out (Recipient)

Pointer to an array of one or more recipient structures allocated by **cmc_look_up()**. The structure may then be used in calls to **cmc_send()**. The returned pointer is passed to **cmc_free()** to free all the recipient structures.

Count (Uint32)

Specifies the number of names actually returned. If no names match the criteria, a value of 0 is returned, and the error CMC_E_RECIPIENT_NOT_FOUND is returned.

If fewer names are returned than are known to be available, the CMC_RECIP_LIST_TRUNCATED flag will be set in the last recipient structure of the array along with the CMC_RECIP_LAST_ELEMENT flag.

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

CMC_E_AMBIGUOUS_RECIPIENT
CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_FLAG
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_SESSION_ID
CMC_E_INVALID_UI_ID
CMC_E_LOGON_FAILURE
CMC_E_NOT_SUPPORTED
CMC_E_RECIPIENT_NOT_FOUND
CMC_E_UNSUPPORTED_DATA_EXT
CMC_E_UNSUPPORTED_FLAG
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_USER_CANCEL
CMC_E_USER_NOT_LOGGED_ON

6.1.4 Administration

Administrative functions defined within this Recommendation include free, logoff, logon, and query configuration.

6.1.4.1 Free

NAME

Free – Free memory allocated by the messaging service.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_free(
   CMC_buffer       memory
);
```

DESCRIPTION

This function frees memory allocated by the messaging service. After the call, the pointer *memory* will be invalid and should not be referenced again. When any CMC 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 CMC 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 CMC function. The CMC functions which return structures of this form are:

cmc_copy_object()
cmc_commit_object()
cmc_copy_object_handle()
cmc_identifier_to_name()
cmc_list()
cmc_list_objects()
cmc_list_properties()
cmc_look_up()
cmc_name_to_identifier()
cmc_open_cursor()
cmc_open_stream()
cmc_query_configuration()
cmc_read()
cmc_read_stream()
cmc_read_property_costs()
cmc_read_properties()
cmc_read_cursor()

cmc_free()'s behavior is undefined when called with a pointer to a memory block not allocated by the messaging service, a pointer to a memory block that has already been freed, or a pointer contained in a structure returned by the CMC implementation.

In some situations, the extensions specified for a function may be a combination of input and output extensions. In this case, the output extensions must be freed one at a time using **cmc_free()**. An example of this is shown in Annex C, "Programming examples".

ARGUMENTS

Memory (Buffer)

A pointer to memory allocated by the messaging service. A value of NULL will be ignored.

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

CMC_E_FAILURE
CMC_E_INVALID_MEMORY

6.1.4.2 Logoff

NAME

Logoff – Log off the CMC service.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_logoff(
   CMC_session_id         session,
   CMC_ui_id              ui_id,
   CMC_flags              logoff_flags,
   CMC_extension          *logoff_extensions
);
```

DESCRIPTION

This function allows the calling application to log off the CMC service. The users of the CMC service should call **cmc_free()** for all memory pointers allocated by the service during this session prior to calling **cmc_logoff()**. Failure to do so may result in memory leaking or undefined behavior of further access to these pointers once the session is terminated.

NOTE – Some implementations of the CMC service may choose to free all the pointers that it created for this session when **cmc_logoff()** is called. However, the support of end-of-session cleanup is optional for the CMC service.

ARGUMENTS

Session (Session Id)

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

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

UI Id (UI Id)

An identifier for a User Interface (e.g. the parent-window handle for the calling application) for use in resolving any questions which might otherwise result in an error.

Ignored if UI is not supported by the CMC implementation.

Logoff Flags (Flags)

Bit mask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_ERROR_UI_ALLOWED
CMC_LOGOFF_UI_ALLOWED

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

CMC_LOGOFF_UI_ALLOWED – Set if the function may display UI other than for errors while logging the user off from the session.

Logoff Extensions (Extension)

A pointer to an array of CMC_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

CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_FLAG
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_SESSION_ID
CMC_E_INVALID_UI_ID
CMC_E_UNSUPPORTED_FLAG
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_USER_NOT_LOGGED_ON

6.1.4.3 Logon

NAME

Logon – Log on to the CMC service.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_logon(
   CMC_string                service,
   CMC_string                user,
   CMC_string                password,
   CMC_object_identifier     character_set,
   CMC_ui_id                 ui_id,
   CMC_uint16                caller_cmc_version,
   CMC_flags                 logon_flags,
   CMC_session_id            *session,
   CMC_extension             *logon_extensions
);
```

DESCRIPTION

This function allows the calling application to log on to the CMC service.

The function returns a session id which the application may use in subsequent CMC calls.

ARGUMENTS

Service (String)

A string indicating the location of the underlying messaging service, e.g. the path to a message store or a remote server node name. This value may be NULL if the underlying messaging service does not require a service name. This may be necessary on some implementations and ignored on others.

The messaging service underlying a CMC implementation, or installation of an implementation, may optionally support multiple messaging protocols simultaneously. If multiple protocols are supported by an implementation, the particular protocol is chosen by the service, based on criteria such as:

– configuration of protocol support;
– dynamic availability of protocol support;

– capabilities of recipient (if known);

– analysis of address format/notation used;

– other system-specific criteria.

These criteria may be applied on a per-message or a per-recipient granularity.

User (String)

A string that identifies the CMC user, e.g. a messaging service user name. This value may be NULL if the underlying messaging service does not require a user name or if UI is allowed.

Password (String)

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

Character Set (Object Identifier)

An object identifier identifying the character set of strings used by the CMC caller. The possible values available to the client are returned by the CMC implementation from **cmc_query_configuration()**. The client may pass NULL in which case the character set used is determined by the CMC service.

UI Id (UI Id)

An identifier for a User Interface (e.g. the parent-window handle for the calling application) for use in resolving any questions which might otherwise result in an error, or for use in prompting for logon if allowed and required.

Ignored if UI is not supported by the CMC implementation.

Caller CMC Version (Uint16)

The calling application's CMC version number, multiplied by 100. For example, version 1.01 is specified as the integer 101. The version of this Recommendation is 2.00 and is represented as the value 200.

Logon Flags (Flags)

Bit mask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_LOGON_UI_ALLOWED

CMC_ERROR_UI_ALLOWED

CMC_COUNTED_STRING_TYPE

CMC_FULL_CMC

CMC_LOGON_UI_ALLOWED – Set if the function should display a dialogue box to prompt for logon if required. If not set, the function will not display a dialogue box and will return an error if not enough information has been supplied.

CMC_ERROR_UI_ALLOWED – Set if the function may display a dialogue box on encountering recoverable errors. If not set, the function may not display a dialogue box and will simply return an error code.

CMC_COUNTED_STRING_TYPE – The CMC caller sets this if the string type that the caller uses for CMC interactions is length first. If not set, null-terminated strings will be assumed.

CMC_FULL_CMC – Set if the application is requesting Full CMC functionality. If this flag is not set, then the application is accessing Simple CMC. Full CMC is only available if the caller specifies a caller_cmc_version of greater than or equal to 200.

Logon Extensions (Extensions)

A pointer to an array of CMC_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. The extension to do this is CMC_X_COM_SUPPORT_EXT. Any CMC implementation that supports extensions must support this extension. For more information on this extension, see the common extensions in B.2.

RESULTS

Session (Session Id)

Opaque session id that represents a session with the CMC service.

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

CMC_E_COUNTED_STRING_UNSUPPORTED
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_CONFIGURATION
CMC_E_INVALID_ENUM
CMC_E_INVALID_FLAG
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_UI_ID
CMC_E_LOGON_FAILURE
CMC_E_PASSWORD_REQUIRED
CMC_E_SERVICE_UNAVAILABLE
CMC_E_UNSUPPORTED_CHARACTER_SET
CMC_E_UNSUPPORTED_FLAG
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_UNSUPPORTED_VERSION

6.1.4.4 Query Configuration

NAME

Query Configuration – Determine information about the installed CMC configuration.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_query_configuration(
   CMC_session_id        session,
   CMC_enum              item,
   CMC_buffer            reference,
   CMC_extension         *config_extensions
);
```

DESCRIPTION

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

NOTE – The configuration may not be changed through CMC, and that any underlying configuration file format is implementation-dependent.

ARGUMENTS

Session (Session Id)

Opaque session id which represents a session with the messaging service.

Session ids 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 Id, session-dependent configuration information will be returned.

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Item (Enum)

This argument indicates which configuration information should be returned. The possible values include:

CMC_CONFIG_CHARACTER_SET

CMC_CONFIG_LINE_TERM

CMC_CONFIG_DEFAULT_SERVICE

CMC_CONFIG_DEFAULT_USER

CMC_CONFIG_REQ_PASSWORD

CMC_CONFIG_REQ_SERVICE

CMC_CONFIG_REQ_USER

CMC_CONFIG_UI_AVAIL

CMC_CONFIG_SUP_NOMKMSGREAD

CMC_CONFIG_SUP_COUNTED_STR

CMC_CONFIG_VER_IMPLEM

CMC_CONFIG_VER_SPEC

CMC_CONFIG_CHARACTER_SET – The reference argument should be a pointer to a CMC_object_identifier array. A pointer to the array of character set object identifier strings for the implementation will be returned here. The array will be terminated with a NULL CMC_Object_Identifier. The first character set Object ID in the array is the default character set used if the caller does not specify one explicitly. The platform specific clause B.2.4 contains the Object ID values defined for common character sets. This pointer to the array should be freed using **cmc_free()**. This Object ID is used by the caller at logon to specify to the implementation to use a different character set than the default.

CMC_CONFIG_LINE_TERM – The reference argument should be a pointer to a CMC_enum variable, which will be set to a value of CMC_LINE_TERM_CRLF if the line delimiter is a carriage return followed by a line feed, CMC_LINE_TERM_LF if the line delimiter is a line feed, or CMC_LINE_TERM_CR if the line delimiter is a carriage return.

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

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

CMC_CONFIG_REQ_PASSWORD – The reference argument should be a pointer to a CMC_enum variable, which will be set to a value of CMC_REQUIRED_NO if the password is not required to log on, CMC_REQUIRED_OPT if the password is optional to log on, or CMC_REQUIRED_YES if the password is required to log on.

CMC_CONFIG_REQ_SERVICE – The reference argument should be a pointer to a CMC_enum variable, which will be set to a value of CMC_REQUIRED_NO if the service name is not required to log on, CMC_REQUIRED_OPT if the service name is optional to log on, or CMC_REQUIRED_YES if the service name is required to log on.

CMC_CONFIG_REQ_USER – The reference argument should be a pointer to a CMC_enum variable, which will be set to a value of CMC_REQUIRED_NO if the user name is not required to log on, CMC_REQUIRED_OPT if the user name is optional to log on, or CMC_REQUIRED_YES if the user name is required to log on.

CMC_CONFIG_UI_AVAIL – The reference argument should be a pointer to a CMC_boolean variable, which will be set to a true value if there is UI provided by the CMC implementation.

CMC_CONFIG_SUP_NOMKMSGREAD – The reference argument should be a pointer to a CMC_boolean variable, which will be set to a true value if the CMC_DO_NOT_MARK_AS_READ flag is supported by **cmc_read()**.

CMC_CONFIG_SUP_COUNTED_STR – The reference argument should be a pointer to a CMC_boolean variable, which will be set to a true value if the CMC_COUNTED_STRING_TYPE flag is supported during logon.

CMC_CONFIG_VER_IMPLEM – The reference argument should be a pointer to a CMC_uint16 variable, which will be set to the version number for the implementation, multiplied by 100. For example, version 1.01 will return 101.

CMC_CONFIG_VER_SPEC – The reference argument should be a pointer to a CMC_uint16 variable, which will be set to the CMC specification version number for the implementation, multiplied by 100. For example, version 1.00 will return 100.

Config Extensions (Extension)

A pointer to an array of CMC_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 CMC_X_COM_SUPPORT_EXT. Any CMC implementation that supports extensions must support this extension. For more information on this extension, see the common extensions in B.2.

RESULTS

Reference (Buffer)

This argument points to the buffer in which to receive the configuration information. The number of bytes implied by the item parameter value must be owned by the caller and modifiable. The type of the variable or buffer depends on the item argument.

Config 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

CMC_E_FAILURE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_INVALID_ENUM

CMC_E_INVALID_PARAMETER

CMC_E_NOT_SUPPORTED

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2 Full CMC functions

The Full CMC is an enhanced set of functions that are intended to provide message-reliant capabilities for messaging-enabled applications. Table 15 lists the functions of the Full CMC interface.

Table 15/X.446 – Full CMC interface functions

Function	Description
Administrative functions	
Free	Refer to Simple CMC for description
Logoff	Refer to Simple CMC for description
Logon	Refer to Simple CMC for description
Bind functions	
Bind Implementation	Create and return a dispatch table
Unbind Implementation	Frees any data associated with a call to **cmc_bind_implementation()** on a specific CMC implementation
Composition functions	
Copy Object	Copies a source object to a container object
Add Properties	Add or modify a set of properties in an object
Commit Object	Commits an object to the persistent store within a container object
Copy Object Handle	Copies an object handle
Delete Objects	Deletes the specified objects from a container
Delete Properties	Deletes the specified properties within an object
Open Object Handle	Open an object handle
Restore Object	Restores object data from a file
Save Object	Saves object data to a file
Enumeration functions	
Get Last Error	Returns a localized text error message for the last error that occurred on the object
Get Root Handle	Returns a handle to the container that is the root of the object model hierarchy for the session
List Contained Properties	Lists the properties within a container object

Table 15/X.446 – Full CMC interface functions *(concluded)*

Function	Description
Enumeration functions *(cont.)*	
List Number Matched	Lists the number of objects within a container that match a criteria
List Objects	Lists the objects within a container object
List Properties	Lists the properties within an object
Open Cursor	Open a cursor for a container object
Read Cursor	Read the current fractional position of a cursor
Read Properties	Read the content information of a set of properties
Read Property Costs	Read the relative cost associated with reading a set of properties
Update Cursor Position	Updates the current fractional position of a cursor
Update Cursor Position with Seed	Updates the current position of a cursor relative to an object in the container
Event notification functions	
Check Event	Checks for a messaging service event
Register Event	Registers events in which the caller is interested in checking
Unregister Event	Unregisters events in which the caller is no longer interested
Call Callbacks	Call the callback function(s) which are registered if the event has occurred
Messaging functions	
Create Derived Message Object	Creates a message for forwarding or replying to a given message
Send Message Object	Submits a message object to the MTA for sending
Name handling functions	
Identifier to Name	Converts a property identifier to a property name
Name To Identifier	Converts a property name to a property identifier
Stream functions	
Export Stream	Exports stream data to a file
Import File to Stream	Imports data from a file to a stream
Open Stream	Open a property for stream read or write operations
Read Stream	Read a stream of content information
Seek Stream	Go to a location in a stream of content information
Write Stream	Write a stream of content information

The manual pages for these functions are given in subsequent pages.

6.2.1 Bind functions

Bind functions enable an implementation to create and return a dispatch table and to subsequently free any data associated with the bind implementation function.

6.2.1.1 Bind Implementation

NAME

Bind Implementation – Creates and returns a dispatch table.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_bind_implementation(
   CMC_guid                   implementation_name,
   CMC_dispatch_table         **dispatch_table,
   CMC_extension              *cmc_bind_implementation_extensions
);
```

DESCRIPTION

This function creates and populates a dispatch table of CMC function addresses for the caller. The function must be supported by the CMC Manager and the CMC implementation. Local administrative tasks may be done with the CMC Manager and/or CMC implementation at this time.

ARGUMENTS

Implementation Name (GUID)

A globally unique identifier which represents a specific CMC implementation. Different versions of the same CMC implementation should be distinguished within this GUID so that the different versions may coexist.

Bind Implementation Extensions (Extension)

A pointer to an array of CMC_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

Dispatch Table (Dispatch Table)

The address of the CMC implementation's dispatch table. This table is allocated by the CMC implementation and should be freed with a call to **cmc_free()**.

Bind Implementation 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

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNRECOGNIZED_IDENTIFIER

CMC_E_BIND_FAILURE

CMC_E_ID_NOT_FOUND

6.2.1.2 Unbind Implementation

NAME

Unbind Implementation – Frees any data associated with a call to **cmc_bind_implementation()** on a specific CMC implementation.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_unbind_implementation(
   CMC_guid                 implementation_name,
   CMC_extension            *cmc_unbind_implementation_extensions
);
```

DESCRIPTION

This function frees and disassociates any data associated with a binding to a specific CMC implementation. Local administrative tasks may be done with the CMC Manager and/or CMC implementation at this time.

ARGUMENTS

Implementation Name (GUID)

A globally unique identifier which represents a specific CMC implementation being unbound from the application or CMC Manager.

Unbind Implementation Extensions (Extension)

A pointer to an array of CMC_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

Unbind Implementation 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

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNRECOGNIZED_IDENTIFIER

CMC_E_UNBIND_FAILURE

CMC_E_ID_NOT_FOUND

6.2.2 Composition functions

The composition functions provide the ability to create and manipulate the CMC objects and object properties.

6.2.2.1 Add Properties

NAME

Add Properties – Adds a set of properties to an object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_add_properties(
   CMC_object_handle          object,
   CMC_uint32                 number_properties,
   CMC_property               *properties,
   CMC_extension              *add_properties_extensions
);
```

DESCRIPTION

This function will add a set of properties to an object.

If the property already exists in the object, then the property will be replaced. If it does not exist, the property will be added. There is no CMC-defined order of properties within an object. It is implementation-specific in what order a new property will be added to an object (e.g. it may not be appended to the end of the object).

Properties added to an object may not be committed until after a call to **cmc_commit_object()**.

ARGUMENTS

Object (Object Handle)

An opaque handle to an object.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Number Properties (Uint32)

The number of properties in the **properties** argument.

Properties (Property)

A pointer to an array of property structures that are to be added to the object.

Add Properties Extensions (Extension)

A pointer to an array of CMC_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 Properties 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.2.2 Commit Object

NAME

Commit Object – Commits an object to the persistent store within a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_commit_object(
   CMC_object_handle         source_object,
   CMC_extension             *commit_object_extensions
);
```

DESCRIPTION

This function will commit an object to the persistent store within a container object.

If the object is being committed to the outbox message container, the action makes the object non-modifiable. The object can only be deleted or copied.

When a message is committed to the outbox, it becomes a candidate for submission at any time. The implementation can send the message at the implementation's convenience. **cmc_send_message_object()** can be used to expedite the immediate sending of a message.

All of the current properties for the source object will be committed to the persistent store within a container object The object must have been added to a container with a previous call to **cmc_copy_object()**.

Any cursor for the container remains valid after the object is committed to the container. Any objects committed to the container, after the cursor was opened, may not be listed in a call to **cmc_list_objects()** for the container. If the container associated with the object does not support commitment of objects, then the error code CMC_E_UNSUPPORTED_ACTION is returned.

ARGUMENTS

Source Object (Object Handle)

An opaque handle for the source object to be committed to the persistent store of the container.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Commit Object Extensions (Extension)

A pointer to an array of CMC_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

Commit Object 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

CMC_E_UNSUPPORTED_ACTION
CMC_E_INVALID_OBJECT_HANDLE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_DISK_FULL
CMC_E_ACCESS_DENIED
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.2.3 Copy Object

NAME

Copy Object – Copies a source object to a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_copy_object(
   CMC_object_handle         container,
   CMC_object_handle         source_object,
   CMC_object_handle         *new_object,
   CMC_extension             *copy_object_extensions
);
```

DESCRIPTION

This function will copy a source object to the specified container object. If the source object is a container object, copy object performs a deep copy function in which all the properties and the contained object are copied recursively.

All of the current properties for the object will be saved with the object in the specified container object. The function adds a new object to the specified container object that contains all of the properties of the source object. A handle to the new object within the container is returned. The source object and its contents are left unchanged. The new object must be committed to the container object with a call to **cmc_commit_object()** before it becomes persistent within the container object.

The container cursors remain valid after objects are added to the container associated with the cursor. Any objects added to the container, after the cursor was opened, may not be listed in a call to **cmc_list_object()** for the container. If the specified container is accessible only in a read-only fashion, then the error code CMC_E_UNSUPPORTED_ACTION is returned.

ARGUMENTS

Container (Object Handle)

An opaque handle to a container object.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Source Object (Object Handle)

An opaque handle for the source object.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Copy Object Extensions (Extension)

A pointer to an array of CMC_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 Object (Object Handle)

An opaque handle for the new object.

Copy Object 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

CMC_E_UNSUPPORTED_ACTION
CMC_E_INVALID_SOURCE_OBJECT
CMC_E_INVALID_CONTAINER_OBJECT
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER

6.2.2.4 Copy Object Handle

NAME

Copy Object Handle – Copies an object handle.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_copy_object_handle(
   CMC_object_handle       source_handle,
   CMC_object_handle       *new_handle,
   CMC_extension           *copy_object_handle_extensions
);
```

DESCRIPTION

This function will copy an object handle. A copy of the object is not created. Instead, the new object handle effectively refers to the original content information that the source object handle referred to. Cursor handles cannot be copied.

This function provides a straightforward method of copying an object handle from an array of object handles returned from another CMC call. A call to **cmc_free()** with the source object handle will not free up the content information referred to by the new object handle. The implementation will only free the related memory when the last reference to it is removed by a call to **cmc_free()** with the last object handle referencing the content information.

ARGUMENTS

Source Handle (Object Handle)

An opaque handle to the source object that is to be copied.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Copy Object Handle Extensions (Extension)

A pointer to an array of CMC_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 Handle (Object Handle)

A new opaque handle for the object.

Copy Object Handle 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.2.5 Delete Objects

NAME

Delete Objects – Deletes the specified objects.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_delete_objects(
   CMC_uint32                number_objects,
   CMC_object_handle         *object,
   CMC_extension             *delete_objects_extensions
);
```

DESCRIPTION

This function deletes the specified objects. Delete Objects performs a deep delete function in which all specified objects and any contained objects are deleted. The object handles are invalid upon return from the call.

ARGUMENTS

Number Objects (Object Handle)

The number of objects in the **objects** parameter.

Objects (Object Handle)

A pointer to an array of opaque handles to objects to be deleted.

If any of the object handles is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Delete Objects Extensions (Extension)

A pointer to an array of CMC_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 Objects 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_ACCESS_DENIED

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.2.6 Delete Properties

NAME

Delete Properties – Deletes the specified set of properties from the object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_delete_properties(
   CMC_object_handle       object,
   CMC_uint32              number_properties,
   CMC_id                  *property_ids,
   CMC_extension           *delete_properties_extensions
);
```

DESCRIPTION

This function deletes the specified properties from the object.

ARGUMENTS

Object (Object Handle)

The opaque handle of the object.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Number Properties (Uint32)

The number of properties in the **properties** argument.

Property Ids (Property Id)

A pointer to an array of the unique ids for the properties to be deleted from the object.

Delete Properties Extensions (Extension)

A pointer to an array of CMC_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 Properties 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.2.7 Open Object Handle

NAME

Open Object Handle – Creates a new object handle.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_open_object_handle(
   CMC_session_id          session,
   CMC_id                  object_class,
   CMC_object_handle       *new_object,
   CMC_extension           *open_object_handle_extensions
);
```

DESCRIPTION

This function will create a new object handle. The service allocates the necessary resources for a new object and returns the handle associated with this object. This object does not exist within any container object until it is added with a call to **cmc_copy_object()**. The content information for this object does not exist until properties are added to the object with a call to **cmc_add_properties()**.

ARGUMENTS

Session (Session id)

The opaque handle which represents a session with the messaging service.

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Object Class (Identifier)

Identifier of the class of the object.

Open Object Handle Extensions (Extension)

A pointer to an array of CMC_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 Object (Object Handle)

An opaque handle for the new object. The session id is encapsulated in the object handle. The object handle is sufficient to reference the proper object within an individual session.

Open Object Handle 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

CMC_E_INVALID_SESSION_ID
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_UNRECOGNIZED_IDENTIFIER

6.2.2.8 Restore Object

NAME

Restore Object – Restores object data from the file system.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_restore_object(
   CMC_object_handle   container,
   CMC_string          file_specification,
   CMC_flags           restore_flags,
   CMC_object_handle   *restored_object,
   CMC_extension       *restore_object_extensions
);
```

DESCRIPTION

This function restores an object from a file. For instance, this function provides a simple method for attaching a file to a message. In this case, *restored object* represents a newly created content item object which will later be associated with a message object under composition. Alternatively, this function provides a method for retrieving a message stored in the file system by an earlier call to **cmc_save_object()**. The on-disk representation for objects stored in the file system is not defined since it may vary from one messaging system to another. As such, applications should not, in general, rely on the ability to import objects which have been exported using other messaging systems. However, this restriction does not apply in the case of a message attachment object.

For a message content item object, this function has the side effect of initializing values for the following properties:

- Filename;
- Creation Date;
- Last Modification Date.

Other properties must be set by calling **cmc_add_properties()**.

NOTE – The file content item must still be associated with a message under composition in order to complete the content item process.

ARGUMENTS

Container (Object Handle)

A handle to the container object which will contain the restored object.

File Specification (String)

A complete file system specification for the file which contains the object data.

Restore Flags (Flags)

Bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_RESTORE_OBJECT_OVERWRITE

CMC_RESTORE_OBJECT_OVERWRITE – Set if the function should overwrite an existing object.

Restore Object Extensions (Extension)

A pointer to an array of CMC_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

Restored Object (Object Handle)

A handle to the restored object.

Restore Object 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

CMC_E_UNSUPPORTED_ACTION

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INVALID_CONTAINER_OBJECT

CMC_E_ACCESS_DENIED

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_FLAG

CMC_E_INVALID_FILE_SPECIFICATION

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.2.9 Save Object

NAME

Save Object – Saves object data to the file system.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_save_object(
   CMC_object_handle   object,
   CMC_string   file_specification,
   CMC_flags    save_flags,
   CMC_extension*save_object_extensions
);
```

DESCRIPTION

This function saves object data to a file. For instance, this function provides a simple method for detaching the data from an attachment to the file system. In this case, *object* represents an attachment object which will later be associated with a message. Alternatively, this function provides a method for storing a message to the file system. The message data can be restored from the file by a subsequent call to **cmc_restore_object()**. The on-disk representation for objects stored in the file system is not defined since it may vary from one messaging system to another. As such, applications should not, in general, rely on the ability to import objects which have been exported using other messaging systems.

ARGUMENTS

Object (Object Handle)

A handle to the object (e.g. message or attachment object) for which data is to be exported.

File Specification (String)

A complete file system specification for the file which will contain the object data.

Save Flags (Flags)

Bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_SAVE_OBJECT_OVERWRITE
CMC_SAVE_OBJECT_NOCREATE

CMC_SAVE_OBJECT_OVERWRITE – Set if the function should overwrite an existing file matching file_specification.

CMC_SAVE_OBJECT_NOCREATE – Set if the function should not create a file matching file_specification if the file does not already exist.

Save Object Extensions (Extension)

A pointer to an array of CMC_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 Object 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_ACCESS_DENIED

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_FLAG

CMC_E_INVALID_FILE_SPECIFICATION

CMC_E_DISK_FULL

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3 Enumeration functions

The enumeration functions provide the ability to list, read, and update the CMC objects and object properties.

6.2.3.1 Get Last Error

NAME

Get Last Error – Returns a localized text error message for the last error that occurred on the object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_get_last_error(
   CMC_session_id   session,
   CMC_object_handle   object,
   CMC_string   *error_buffer,
   CMC_extension*get_last_error_extensions
);
```

DESCRIPTION

The cmc_get_last_error function is used by client applications to retrieve a localized string to display to the user which corresponds to the last error returned from a function call made on this object. The implementation allocates storage for the returned buffer and the client is responsible for freeing it. If the function returns an error (non-zero), the calling application should not call cmc_get_last_error again for additional diagnostics. Even if the function returns zero, it is still possible that no string is available. The return code must be (zero) for the application to make use of the descriptive string. Implementations of cmc_get_last_error should localize error messages to the language of the system, which requires the user to set the appropriate character set in the cmc_logon call.

If both the session and object parameters are NULL, this indicates a get last error request from cmc_logon, where the returned error string would be in the default code page for the system. If the session id is valid and the object value is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned. If the session id parameter is invalid and the object parameter is valid, then the error CMC_E_INVALID_SESSION_ID is returned.

ARGUMENTS

Session (Session Id)

Session id which represents the session with the CMC service during which the error occurred.

Object (Object Handle)

A handle to the object (e.g. message or attachment object) for which data is to be returned from.

Get Last Error Extensions (Extension)

A pointer to an array of CMC_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

Error Buffer (String)

The address of the buffer where the implementation stores the descriptive error string. This buffer is allocated by the service and should be freed with a call to **cmc_free()**.

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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INVALID_SESSION_ID

CMC_E_INSUFFICIENT_MEMORY

CMC_E_INVALID_PARAMETER

CMC_E_FAILURE

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.2 Get Root Handle

NAME

Get Root Handle – Returns a handle to the container that is the root of the object model hierarchy.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_get_root_handle(
   CMC_session  session,
   CMC_object_handle   *root_object_handle,
   CMC_extensions   *get_root_handle_extensions
);
```

DESCRIPTION

This function returns a handle to the container that is the root of the object model hierarchy for the session. Multiple calls to this function will return the same object handle during the lifetime of the session.

ARGUMENTS

Session (Session ID)

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

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

Get Root Handle Extensions (Extension)

A pointer to an array of CMC_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

Root Object Handle (Object Handle)

A handle to the container that is the root of the object model hierarchy.

Get Root Handle 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

CMC_E_INVALID_SESSION_ID
CMC_E_ACCESS_DENIED
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.3 List Contained Properties

NAME

List Contained Properties – Lists the properties of objects within a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_list_contained_properties(
   CMC_cursor_handle   cursor,
   CMC_sint32   *number_objects,
   CMC_uint32   *number_properties,
   CMC_id       *property_ids,
   CMC_property ***properties,
   CMC_extension*list_contained_properties_extensions
);
```

DESCRIPTION

This function lists the properties of objects within a container object. One of the purposes of this function is to retrieve summary information about the objects in the container (e.g. compose an inbox message summary).

ARGUMENTS

Cursor (Cursor Handle)

The opaque handle for the cursor to the specified container object.

Number Objects (Sint32)

A pointer to the maximum number of object handles to return. A value of zero specifies no maximum. A negative value specifies that the handles of the specified number of objects that precede the current position of the cursor should be returned in the same sort of order as specified by cursor. For example, if the current position of the cursor is on the eighth object in the container, then a value of –3 will list the handles of the fifth, sixth, and seventh objects and the cursor is updated to the fifth object. A value of 4 will list the handles of the eighth, ninth, tenth, eleventh objects and the cursor will be updated to the twelfth object.

Number Properties (Uint32)

A pointer to the number of properties in the **Property Ids** argument.

Property Ids (Property Id)

A pointer to an array of property identifiers corresponding to the properties that are to be listed.

List Contained Properties Extensions (Extension)

A pointer to an array of CMC_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 Objects (Sint32)

The actual number of objects for which properties are returned.

Number Properties (Uint32)

The actual number of properties returned for each object.

Properties (Property)

The address of an array of arrays of property structures within the container object that are listed. Each array is the set of properties associated with a single object. The number of elements in the array are given in the **Number Properties** result. The number of arrays are given in the **Number Objects** result. This array of arrays is allocated by the service and should be freed with a call to **cmc_free()**.

List Contained Properties 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

CMC_E_INVALID_CURSOR_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_PROPERTY_ID

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.4 List Number Matched

NAME

List Number Matched – Lists the number of elements within a container object that match the restrictions specified by a cursor.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_list_number_matched(
   CMC_cursor_handle   cursor,
   CMC_uint32   *number_matches,
   CMC_extension *list_number_matched_extensions
);
```

DESCRIPTION

This function returns the number of elements within a container that match the restrictions specified by a cursor. This value can be used with the current fractional position of the cursor to display a "thumb" on a scroll bar.

ARGUMENTS

Cursor (Cursor Handle)

The opaque handle to a cursor.

If the cursor handle is invalid, then the error CMC_E_INVALID_CURSOR_HANDLE is returned.

List Number Matched Extensions (Extension)

A pointer to an array of CMC_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 Matches (Uint32)

The number of elements within the container that match the restrictions specified by the cursor. If zero, no elements match the restrictions specified by the cursor.

List Number Matched 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

CMC_E_INVALID_CURSOR_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.5 List Objects

NAME

List Objects – Lists the elements within a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_list_objects(
   CMC_cursor_handle   cursor,
   CMC_sint32   *number_objects,
   CMC_object_handle   **objects,
   CMC_extension *list_objects_extensions
);
```

DESCRIPTION

This function returns a pointer to an array of object handles that correspond to the elements within a container object. The container object is referenced by a cursor that has been opened by a call to the **cmc_open_cursor()** function. The cursor is updated by the service so that subsequent calls to this function will return object handles to additional elements of the container based on the updated position of the cursor.

ARGUMENTS

Cursor (Cursor Handle)

The opaque handle to a cursor.

If the cursor handle is invalid, then the error CMC_E_INVALID_CURSOR_HANDLE is returned.

Number Objects (Sint32)

A pointer to the maximum number of object handles to return. A value of zero specifies no maximum. A negative value specifies that the handles of the specified number of objects that precede the current position of the cursor should be returned in the same sort of order as specified by cursor. For example, if the current position of the cursor is on the eighth element in the container, then a value of –3 will list the handles of the fifth, sixth, and seventh elements and the cursor is updated to the eighth element. A value of 4 will list the handles of the eighth, ninth, tenth, eleventh elements and the cursor will be updated to the twelfth element.

List Objects Extensions (Extension)

A pointer to an array of CMC_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

Cursor (Cursor Handle)

The opaque handle for the cursor to the specified container object. This handle may have been updated by the service.

Number Objects (Sint32)

The actual number of object handles returned. If no elements matched the cursor restrictions, or if the container object was empty, a value of zero is returned.

Objects (Object Handle)

The address of an array of object handles corresponding to the elements in the container object. This array is allocated by the service and should be freed with a call to **cmc_free()**.

NOTE – The individual object handles within this array become invalid when the array is freed. The application can retain handles to the objects prior to invoking **cmc_free()** on the array. Using a freed handle will result in an undefined behaviour.

List Objects 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

CMC_E_INVALID_CURSOR_HANDLE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.6 List Properties

NAME

List Properties – Lists the properties in an object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_list_properties(
   CMC_object_handle   object,
   CMC_uint32   *number_properties,
   CMC_id       **property_ids,
   CMC_extension*list_properties_extensions
);
```

DESCRIPTION

This function returns the unique ids of the properties within an object. A subsequent call to **cmc_read_properties()** will return the property content information for the object.

ARGUMENTS

Object (Object Handle)

The opaque handle of the object to be listed.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Number Properties (Uint32)

A pointer to the maximum number of property ids to return. A value of zero specifies all of the properties should be listed.

List Properties Extensions (Extension)

A pointer to an array of CMC_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 Properties (Uint32)

The actual number of property ids returned.

Property Ids (Identifier)

The address of an array of unique property ids corresponding to the properties in the object. This array is allocated by the service and should be freed with a call to **cmc_free()**.

List Properties 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_PROPERTY_NAME

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.7 Open Cursor

NAME

Open Cursor – Opens a cursor for a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_open_cursor(
   CMC_object_handle   object,
   CMC_cursor_restriction *restriction,
   CMC_uint32   number_sort_keys,
   CMC_cursor_sort_key *sort_keys,
   CMC_cursor_handle   *cursor,
   CMC_extension *open_cursor_extensions
);
```

DESCRIPTION

This function returns an opaque handle of a cursor to the specified container object. The cursor can be defined to operate on the container with specified sort-rules.

ARGUMENTS

Object (Object Handle)

The opaque handle to a container object.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Restriction (Cursor Restriction)

A pointer to a cursor restriction structure to be used in the enumeration of the elements in the container. Implementations may not support all types of restrictions.

Number Sort Keys (Uint32)

The number of elements in the sort_keys argument. If zero, the sort rules for the container are undefined.

Sort Keys (Cursor Sort Key)

A pointer to an array of cursor sort keys for sorting the container. The first element is the first sort key, the second element is the second sort key, etc. Allowing more than one sort key may not be supported by all implementations. Objects that do not have the property listed by the sort key are placed last.

Open Cursor Extensions (Extension)

A pointer to an array of CMC_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

Cursor (Cursor Handle)

An opaque handle for the cursor to the specified container object. This handle is allocated by the service and should be freed with a call to **cmc_free()**.

Open Cursor 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

CMC_E_INVALID_OBJECT_HANDLE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_RESTRICTION
CMC_E_UNSUPPORTED_KEYS
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.8 Read Cursor

NAME

Read Cursor – Read, the current fractional position of the specified cursor within a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_read_cursor(
   CMC_cursor_handle   cursor,
   CMC_uint32   *position_numerator,
   CMC_uint32   *position_denominator,
   CMC_extension*read_cursor_extensions
);
```

DESCRIPTION

This function returns the current fractional position of the specified cursor. The values returned in position_numerator and position_denominator are suitable for determining and drawing a "thumb" on a scroll bar. The scroll bar maximum could be determined by a specific container object property.

ARGUMENTS

Cursor (Cursor Handle)

An opaque handle to a cursor.

If the cursor handle is invalid, then the error CMC_E_INVALID_CURSOR_HANDLE is returned.

Read Cursor Extensions (Extension)

A pointer to an array of CMC_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

Position Numerator (Uint32)

The numerator component of the current cursor position fraction. The ratio of the position_numerator/position_denominator provides an approximate fractional position of the cursor through the elements of the container object. This storage is allocated by the caller.

Position Denominator (Uint32)

The denominator component of the current position of the cursor. The ratio of the position_numerator/position_denominator provides an approximate fractional position of the cursor through the elements of the container object. This storage is allocated by the caller.

Read Cursor 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

CMC_E_INVALID_CURSOR_HANDLE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_PROPERTY_NAME
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.9 Read Properties

NAME

Read Properties – Reads the content information associated with a set of properties in an object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_read_properties(
   CMC_object_handle   object,
   CMC_uint32   *number_properties,
   CMC_id       *property_ids,
   CMC_property **properties,
   CMC_extension*read_properties_extensions
);
```

DESCRIPTION

This function returns the content information of the specified properties within an object.

If a specified property is not in the object, then the property type CMC_pv_return_code will be returned in the position of that property in the **properties** argument with the property value of the return code CMC_E_PROPERTY_ID_NOT_FOUND. The property identifier for this property is undefined by this Recommendation.

ARGUMENTS

Object (Object Handle)

The opaque handle of the object to be listed.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Number Properties (Uint32)

A pointer to the number of property ids in the **property_ids** argument.

Property Ids (Identifier)

A pointer to an array of unique property ids corresponding to the properties that are to be read.

Read Properties Extensions (Extension)

A pointer to an array of CMC_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 Properties (Uint32)

The actual number of properties returned. If none of the specified properties were in the object, a value of zero is returned.

Properties (Property)

A pointer to an array of property structures that contain the content information for the properties that were read. This array is allocated by the service and should be freed with a call to **cmc_free()**.

Read Properties 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

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_PROPERTY_NAME

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.10 Read Property Costs

NAME

Read Property Costs – Reads the relative cost associated with reading individual properties in an object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_read_property_costs(
   CMC_object_handle   object,
   CMC_uint32    *number_properties,
   CMC_id        *property_ids,
   CMC_enum      *costs,
   CMC_extension *read_property_costs_extensions
);
```

DESCRIPTION

This function returns the relative cost associated with reading individual properties within an object.

Support for this function is not mandatory for conformance to this Recommendation. Implementations that do not support this function shall return the error CMC_E_NOT_SUPPORTED.

The basis for determining the cost of reading the property is implementation-specific.

ARGUMENTS

Object (Object Handle)

The opaque handle of the object to be listed.

If the object handle is invalid, then the error CMC_E_INVALID_OBJECT_HANDLE is returned.

Number Properties (Uint32)

A pointer to the number of property ids in the **property_ids** argument.

Property Ids (Identifier)

A pointer to an array of unique property ids corresponding to the properties that are to be read.

Read Property Costs Extensions (Extension)

A pointer to an array of CMC_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 Properties (Uint32)

The actual number of property costs returned. If none of the specified property costs were read, a value of zero is returned.

Costs (Enum)

A pointer to an array of relative property costs. The individual costs correspond one-for-one to the specified property names. The valid relative cost values include the following:

CMC_COST_UNDETERMINED
CMC_COST_NONE
CMC_COST_MINOR
CMC_COST_MAJOR

CMC_COST_UNDETERMINED – The cost of reading the property cannot be determined.

CMC_COST_NONE – There is no relative cost associated with reading the property.

CMC_COST_MINOR – There is only a relatively low cost associated with reading the property.

CMC_COST_MAJOR – There is a relatively high cost associated with reading the property.

This array is allocated by the service and should be freed with a call to **cmc_free()**.

Read Property Costs 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

CMC_E_INVALID_OBJECT_HANDLE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_PROPERTY_NAME
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_NOT_SUPPORTED

6.2.3.11 Update Cursor Position

NAME

Update Cursor Position – Updates the current fractional position of the specified cursor within a container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_update_cursor_position(
   CMC_cursor_handle   cursor,
   CMC_uint32   position_numerator,
   CMC_uint32   position_denominator,
   CMC_extension *update_cursor_position_extensions
);
```

DESCRIPTION

This function updates the cursor to a specified position within the elements of a container object. The position is determined by the ratio of the position_numerator to the position_denominator.

ARGUMENTS

Cursor (Cursor Handle)

The opaque handle to a cursor. If the cursor handle is invalid, then the error CMC_E_INVALID_CURSOR_HANDLE is returned.

Position Numerator (Uint32)

The numerator of the desired cursor position fraction. The ratio of the position_numerator to position_denominator provides the fractional position of the cursor through the elements of the container object.

Position Denominator (Uint32)

The denominator of the current position of the cursor. The ratio of the position_numerator to position_denominator provides the fractional position of the cursor through the elements of the container object.

Update Cursor Position Extensions (Extension)

A pointer to an array of CMC_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 Cursor Position 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

CMC_E_INVALID_CURSOR_HANDLE

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.3.12 Update Cursor Position With Seed

NAME

Update Cursor Position With Seed – Updates the current position of the specified cursor relative to a specific seed object within the container object.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_update_cursor_position_with_seed(
   CMC_cursor_handle   cursor,
   CMC_object_handle   seed,
   CMC_extension*update_cursor_position_with_seed_extensions
);
```

DESCRIPTION

This function updates the cursor to a specified position within the elements of a container object. The position is determined by the relative position of the seed object within the container.

ARGUMENTS

Cursor (Cursor Handle)

The opaque handle to a cursor. If the cursor handle is invalid, then the error CMC_E_INVALID_CURSOR_HANDLE is returned.

Seed (Object Handle)

The opaque handle of the object within the container relative to which cursor position should be updated.

Update Cursor Position with Seed Extensions (Extension)

A pointer to an array of CMC_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 Cursor Position with Seed 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

CMC_E_INVALID_CURSOR_HANDLE

CMC_E_INVALID_CURSOR_HANDLE

CMC_E_INVALID_OBJECT_HANDLE

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.4 Event notification functions

Event notification functions enable an implementation to check for events, register and unregister events, and call callbacks.

6.2.4.1 Check Event

NAME

Check Event – Checks for a messaging service event.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_check_event(
   CMC_session_id   session,
   CMC_event       event_type,
   CMC_uint32      minimum_timeout,
   CMC_buffer      check_event_data,
   CMC_buffer      *callback_data,
   CMC_extension   *check_event_extensions
);
```

DESCRIPTION

This function checks for an event associated with the messaging system. It provides an alternative to registering callbacks with the CMC implementation for those applications which prefer to poll synchronously for events or to provide event notification from implementations which do not support callbacks.

For each event, there is a flag associated with the event. There may also be input and output parameters associated with an event. These event data structures are given in the Callback data type.

If an event has not occurred and the minimum time-out is non-zero, the implementation waits for the event the specified time-out before returning to the calling program. If the event occurs before that time-out is reached, the function returns immediately. If the error does not occur before the time-out is reached, the function returns CMC_E_NO_EVENT.

Under implementation-defined circumstances, which are not considered as actual errors, this function may terminate prematurely, before any event was detected and before the specified time-out is reached. In this case, the function returns the code CMC_E_FUNCTION_INTERRUPTED.

NOTE – Other errors can also arise that cause this function to return prematurely. In this case, CMC_E_FUNCTION_INTERRUPTED is not used. Instead, the appropriate CMC error code is returned.

ARGUMENTS

Session (Session id)

The opaque handle which represents a session with the messaging service.

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Event Type (Event)

A bitmask of events for which the caller is interested in checking. Unspecified events should always be passed as 0. Undocumented events are reserved. The definition of CMC events is given in the Event data type description.

Minimum Time-out (Uint32)

The time, in seconds, after which the function returns even if the event has not occurred.

A value of zero causes the function to simply check for the event and return immediately thereafter.

The value CMC_NO_TIMEOUT indicates that the function should wait for the event without any time limit.

If a value other than CMC_NO_TIMEOUT is used, the actual minimum time spent in this function is implementation-dependent.

Check Event Data (Buffer)

A pointer to a check data structure associated with this event. See the Callback data type for the specific structure of check data. Whether the implementation or application allocates the buffer is event-specific and detailed in the data type description.

Check Event Extensions (Extension)

A pointer to an array of CMC_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

Callback Data (Buffer)

The address of the callback data structure associated with this event. For this call, the structure is returned directly to the application rather than being directed at a callback function. See the Callback data type for the event description and specific structure of the callback data. Whether the implementation or application allocates the buffer is event-specific and detailed in the data type description.

Check Event Extensions (Extensions)

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

CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_EVENT
CMC_E_INVALID_FUNCTION_EXT
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_FUNCTION_INTERRUPTED
CMC_E_INVALID_SESSION_ID
CMC_E_NO_EVENT

6.2.4.2 Register Event

NAME

Register Event – Registers events in which the caller is interested.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_register_event(
   CMC_session_id   session,
   CMC_event        event_type,
   CMC_callback     callback,
   CMC_buffer       register_data,
   CMC_extension    *register_event_extensions
);
```

DESCRIPTION

This function specifies the events within the messaging system of which the caller is interested in being alerted.

The caller can be notified by an event either through a callback function or by using the Check Event function call to synchronously poll for events for which it has registered. CMC implementations are not required to support callbacks.

There may also be input and output parameters associated with an event. These parameters are contained in the Client Data. The structure of client data for events is given in the Callback data type.

ARGUMENTS

Session (Session id)

The opaque handle which represents a session with the messaging service.

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Event Type (Event)

A bitmask of events for which the caller is interested in checking. Unspecified events should always be passed as 0. Undocumented events are reserved. The definition of CMC events is given in the Event data type description.

Callback (Callback)

The client procedure that should be called by the service to handle the callback activity. A NULL value indicates that no callback function is given and that the event should be signalled through the Check Event function. If callbacks are not supported by an implementation, the error code CMC_E_CALLBACK_NOT_SUPPORTED is returned.

Register Data (Buffer)

A pointer to a register data structure associated with this event. See the Callback data type for the specific structure of register data. Whether the implementation or application allocates the buffer is event-specific and detailed in the event description.

Register Event Extensions (Extension)

A pointer to an array of CMC_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 Event Extensions (Extensions)

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

CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_EVENT
CMC_E_INVALID_FUNCTION_EXT
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_CALLBACK_NOT_SUPPORTED
CMC_E_INVALID_SESSION_ID

6.2.4.3 Unregister Event

NAME

Unregister Event – Unregisters events for which the caller is no longer interested.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_unregister_event(
   CMC_session_id  session,
   CMC_flags      event_type,
   CMC_callback   callback,
   CMC_buffer     unregister_data,
   CMC_extension  *unregister_event_extensions
);
```

DESCRIPTION

This function specifies events within the messaging system for which the caller is interested in discontinuing notification.

ARGUMENTS

Session (Session id)

The opaque handle which represents a session with the messaging service.

If the session id is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Event Type (Flags)

A bitmask of events for which the caller is no longer interested in checking. Unspecified events should always be passed as 0. Undocumented events are reserved. The definition of CMC events is given in the Event data type description.

Callback (Callback)

The client procedure that was registered to handle the callback activity. A NULL value indicates that no callback function was designated. If callbacks are not supported by an implementation, the error code CMC_E_CALLBACK_NOT_SUPPORTED is returned.

Unregister Data (Buffer)

A pointer to an unregister data structure that can be used to pass event data that will be needed by the callback function to provide a context for discontinuing registration. The structure of Unregister Data is given in the Callback data type description.

Unregister Event Extensions (Extension)

A pointer to an array of CMC_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 Event Extensions (Extensions)

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

CMC_E_FAILURE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_INVALID_EVENT

CMC_E_INVALID_FUNCTION_EXT

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

CMC_E_NOT_SUPPORTED

CMC_E_INVALID_SESSION_ID

6.2.4.4 Call Callbacks

NAME

Call Callbacks – Calls the callback function(s) which are registered if the event has occurred.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_call_callbacks(
   CMC_session_id   session,
   CMC_event     event_type,
   CMC_extension *call_callbacks_extensions
);
```

DESCRIPTION

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

This function is useful in environments where an implementation can only call callbacks when the implementation's code is executing. That is, this function is useful for implementations where callbacks can only be called as a side effect of calling any CMC function in that implementation.

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

ARGUMENTS

Session (Session Id)

Opaque session handle which represents a session with the messaging service. If a valid session handle is specified, the callback functions registered with that session are invoked. If the session handle is invalid, then the error CMC_E_INVALID_SESSION_ID is returned.

Event Type (Event)

A bitmask of events. Unspecified events should always be passed as 0. Undocumented events are reserved. The definition of CMC events is given in the Event data type description.

Call Callbacks Extensions (Extension)

A pointer to an array of CMC_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 (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

CMC_E_FAILURE

CMC_E_INSUFFICIENT_MEMORY

CMC_E_INVALID_EVENT

CMC_E_INVALID_FUNCTION_EXT

CMC_E_INVALID_PARAMETER

CMC_E_INVALID_SESSION_ID

CMC_E_NOT_SUPPORTED

CMC_E_SERVICE_UNAVAILABLE

CMC_E_UNSUPPORTED_FLAG

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.5 Messaging functions

The messaging functions provide the ability to create derived messages, send messages, and wait for new messages.

6.2.5.1 Create Derived Message Object

NAME

Create Derived Message Object – Creates a message object that is suitable for forwarding or replying.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_create_derived_message_object(
   CMC_object_handle   original_message,
   CMC_enum     derived_action,
   CMC_boolean  inherit_contents,
   CMC_boolean  modified_message,
   CMC_object_handle   *derived_message,
   CMC_extensions   *create_derived_message_object_extensions
);
```

DESCRIPTION

This function is used to create a message suitable for forwarding or replying to a given message. The "object" parameter must be a message object with at least one recipient.

The derived_message may contain additional properties in addition to those in original_message. Likewise, the properties in derived_message may not have the same values as corresponding properties in original_message. For example, some implementations will alter the subject of a replied message from, say, "Quarterly Financial Results", to say, "Re: Quarterly Financial Results". The implementation must define the rules to apply to this function including which attributes get modified and what extra attributes get generated in the derived message.

An originator recipient object is needed to reply to the message.

ARGUMENTS

Original Message (Object Handle)

A handle to the message object that is to be forwarded or replied to.

Derived Action (Enum)

Indicates whether the derived message is intended to be forwarded or replied to. It can be one of the following values:

CMC_DERIVED_ACTION_FORWARD
CMC_DERIVED_ACTION_REPLY_ORIGINATOR
CMC_DERIVED_ACTION_REPLY_ALL

CMC_DERIVED_ACTION_FORWARD – The message is intended to be forwarded.

CMC_DERIVED_ACTION_REPLY_ORIGINATOR – The message is intended to be replied to the originator.

CMC_DERIVED_ACTION_REPLY_ALL – The message is intended to be replied to all the recipients of the original message.

Inherit Contents (Boolean)

All the contents of the original message are either copied and included in the derived message or are ignored. If true, the new object inherits all the contents.

Modified Message (Boolean)

Specifies whether the original message should be changed.

Create Derived Message Object Extensions (Extension)

A pointer to an array of CMC_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

Derived Message (Object Handle)

A new handle to a message object that can be forwarded or replied to.

Create Derived Object 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

CMC_E_INVALID_OBJECT_HANDLE
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_UNSUPPORTED_ACTION
CMC_E_REQUIRED_PROPS_MISSING

6.2.5.2 Send Message Object

NAME

Send Message Object – Sends a message object from the outbox.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_send_message_object(
   CMC_object_handle   object,
   CMC_extensions   *send_message_object_extensions
);
```

DESCRIPTION

This function is used to send a message from the outbox, if the outbox container is supported. The function will also attempt to transfer all other committed messages in the outbox. The "object" parameter must be a message object with at least one recipient.

ARGUMENTS

Object (Object Handle)

A handle to the message object that is to be submitted to the messaging service.

Send Message Object Extensions (Extension)

A pointer to an array of CMC_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

Send Message Object 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

CMC_E_INVALID_OBJECT_HANDLE
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT
CMC_E_REQUIRED_PROPS_MISSING

6.2.6 Name handling functions

The name handling functions provide the ability to convert a property identifier to a property name and convert a property name to a property identifier.

6.2.6.1 Identifier To Name

NAME

Identifier To Name – Converts an identifier into its associated unique name.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_identifier_to_name(
   CMC_id       identifier,
   CMC_name     *name,
   CMC_extension*identifier_to_name_extensions
);
```

DESCRIPTION

This function converts an identifier into its corresponding unique name. It may be used for object class identifiers and property identifiers.

The name is a formal public identifier, as defined by ISO 9070. The identifier is an implementation-specific, unique identifier. The identifier is used to uniquely identify the property or object class within the CMC property structure.

ARGUMENTS

Identifier (Identifier)

The identifier to be converted into a name.

Identifier To Name Extensions (Extension)

A pointer to an array of CMC_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

Name (Name)

The name of the identifier string. This string is allocated by the service and should be freed with **cmc_free()**.

Identifier To Name 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

CMC_E_INVALID_PROPERTY_ID
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_PROPERTY_NAME_NOT_FOUND
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.6.2 Name To Identifier

NAME

Name To Identifier – Converts a unique name into its corresponding identifier.

SYNOPSIS

```
#include <xcmc.h>
CMC_return_code
cmc_name_to_identifier(
   CMC_name      name,
   CMC_id        *identifier,
   CMC_extension *name_to_identifier_extensions
);
```

DESCRIPTION

This function converts a unique name into its corresponding identifier. It may be used for object class identifiers and property identifiers.

The name is a formal public identifier, as defined by ISO 9070. The identifier is an implementation specific, unique identifier. The identifier is used to uniquely identify a property or an object class.

ARGUMENTS

Name (Name)

The name to be converted into an identifier.

Name To Identifier Extensions (Extension)

A pointer to an array of CMC_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

Identifier (Identifier)

The identifier corresponding to the name.

Name To Identifier 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

CMC_E_INVALID_PROPERTY_NAME
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_PROPERTY_ID_NOT_FOUND
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.7 Stream functions

Some CMC properties may be defined in terms of large amounts of content information. These properties necessitate a group of functions to permit the access to the content information in the form of streamed input or output.

6.2.7.1 Export Stream

NAME

Export Stream – Export stream data to the file system.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_export_stream(
   CMC_stream_handle   stream,
   CMC_string   file_specification,
   CMC_uint32   count,
   CMC_flags    export_flags,
   CMC_extension *export_stream_extensions
);
```

DESCRIPTION

This function exports stream data to a file.

ARGUMENTS

Stream (Stream Handle)

A handle to the stream from which data is to be exported.

File Specification (String)

A complete file system specification for the file which will contain the stream data.

Count (Uint32)

Specifies the number of bytes to export.

Export Flags (Flags)

Bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved.

CMC_EXPORT_STREAM_OVERWRITE

CMC_EXPORT_STREAM_NOCREATE

CMC_EXPORT_STREAM_APPEND

CMC_EXPORT_STREAM_OVERWRITE – Set if the function should overwrite an existing file matching file_specification.

CMC_EXPORT_STREAM_NOCREATE – Set if the function should not create a file matching file_specification if the file does not already exist.

CMC_EXPORT_STREAM_APPEND – Set if the function should append the stream data to an existing file matching file_specification.

Export Stream Extensions (Extension)

A pointer to an array of CMC_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

Export Stream 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

CMC_E_INVALID_STREAM_HANDLE
CMC_E_ACCESS_DENIED
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_FLAG
CMC_E_INVALID_FILE_SPECIFICATION
CMC_E_DISK_FULL
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.7.2 Import File To Stream

NAME

Import File To Stream – Import data from the file system to a stream.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_import_file_to_stream(
   CMC_stream_handle   stream,
   CMC_string   file_specification,
   CMC_uint32   file_offset,
   CMC_extension *import_file_to_stream_extensions
);
```

DESCRIPTION

This function imports data from a file to a stream.

ARGUMENTS

Stream (Stream Handle)

A handle to the stream to which data are to be imported.

File Specification (String)

A complete file system specification for the file from which to import data.

File Offset (Uint32)

Specifies the offset in bytes from the beginning of the file from which to begin reading data.

Import File to Stream Extensions (Extension)

A pointer to an array of CMC_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

Import File to Stream 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

CMC_E_INVALID_STREAM_HANDLE
CMC_E_ACCESS_DENIED
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_INVALID_FLAG
CMC_E_INVALID_FILE_SPECIFICATION
CMC_E_INVALID_FILE_OFFSET
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.7.3 Open Stream

NAME

Open Stream – Open a property for stream-based read or write operations.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_open_stream(
   CMC_object_handle   object,
   CMC_property_id  property_id,
   CMC_enum     operation,
   CMC_stream_handle   *stream,
   CMC_extension *open_stream_extensions
);
```

DESCRIPTION

This function will open a stream for reading or writing large content information in a property.

ARGUMENTS

Object (Object Handle)

Opaque object handle. This handle encapsulates the session id.

Property Id (Property Id)

Property to read or write through the stream.

Operation (Enum)

The operation the stream is to be used for. Valid operations include:

CMC_OPEN_READ
CMC_OPEN_WRITE

CMC_OPEN_READ – Open the stream for read operations.

CMC_OPEN_WRITE – Open the stream for write operations.

Open Stream Extensions (Extension)

A pointer to an array of CMC_extension structures for this function. The array may contain both input extension 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

Stream (Stream Handle)

The stream handle allocated for accessing the specified property. The returned value is passed to **cmc_free()** to free the handle and any service specific information about the stream when it is no longer used.

Open Stream 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

CMC_E_INVALID_OBJECT_HANDLE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PROPERTY_ID
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.7.4 Read Stream

NAME

Read Stream – Read a stream of content information from the specified property.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_read_stream(
   CMC_stream_handle   stream,
   CMC_uint32   *count,
   CMC_buffer   content_information,
   CMC_extension*read_stream_extensions
);
```

DESCRIPTION

This function will read content information from the specified property into a user-managed buffer.

ARGUMENTS

Stream (Stream Handle)

Opaque stream handle. This handle encapsulates the session and object handles.

Count (Uint32)

Specifies the maximum number of bytes to be read. A value of zero specifies no maximum.

Read Stream Extensions (Extension)

A pointer to an array of CMC_extension structures for this function. The array may contain both input extension 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

Count (Uint32)

Specifies the number of bytes of content information actually read. If nothing was read, a value of zero is returned.

Content Information (Buffer)

A buffer which contains the content information that was read. This buffer is allocated by the service and the entire buffer should be freed with a single call to **cmc_free()**. This buffer is managed by the user of the API, not by the service.

Read Stream 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

CMC_E_INVALID_STREAM_HANDLE
CMC_E_ACCESS_DENIED
CMC_E_INSUFFICIENT_MEMORY
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.7.5 Seek Stream

NAME

Seek Stream – Move to the specified location with the content information of the specified property stream.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_seek_stream(
   CMC_stream_handle   stream,
   CMC_enum     operation,
   CMC_uint32   *location,
   CMC_extension*seek_stream_extensions
);
```

DESCRIPTION

This function will move to the specified location within a property stream. The location is specified as a byte offset from either the beginning, the end, or the current position within the content information.

ARGUMENTS

Stream (Stream Handle)

Opaque stream handle. This handle encapsulates the session and object handles.

Operation (Enum)

The seek direction. It will specify either to seek from the beginning, the end of the content information, or from the current position in the stream. Valid operations include:

CMC_SEEK_BEGINNING

CMC_SEEK_END

CMC_SEEK_CURRENT_POSITION

CMC_SEEK_BEGINNING – Seek the specified offset from the beginning of the content information.

CMC_SEEK_END – Seek the specified offset from the end of the content information.

CMC_SEEK_CURRENT_POSITION – Seek the specified offset from the current position with the content information.

Location (Uint32)

Pointer to the byte offset or location within the stream.

Seek Stream Extensions (Extension)

A pointer to an array of CMC_extension structures for this function. The array may contain both input extension 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

Location (Uint32)

The actual byte offset moved to.

Seek Stream 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

CMC_E_INVALID_STREAM_HANDLE

CMC_E_ACCESS_DENIED

CMC_E_INSUFFICIENT_MEMORY

CMC_E_FAILURE

CMC_E_INVALID_PARAMETER

CMC_E_UNSUPPORTED_FUNCTION_EXT

6.2.7.6 Write Stream

NAME

Write Stream – Write a stream of content information to the specified property.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_write_stream(
   CMC_stream_handle   stream,
   CMC_uint32   count,
   CMC_buffer   content_information,
   CMC_extension *write_stream_extensions
);
```

DESCRIPTION

This function will write content information to the specified property.

ARGUMENTS

Stream (Stream Handle)

Opaque stream handle. This handle encapsulates the session and object handles.

Count (Uint32)

Specifies the number of bytes to be written to the property.

Content Information (Buffer)

Pointer to a buffer which contains the content information to be written.

Write Stream Extensions (Extension)

A pointer to an array of CMC_extension structures for this function. The array may contain both input extension 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

Write Stream 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

CMC_E_INVALID_STREAM_HANDLE
CMC_E_ACCESS_DENIED
CMC_E_INSUFFICIENT_MEMORY
CMC_E_NO_MORE_BYTES_TO_WRITE
CMC_E_FAILURE
CMC_E_INVALID_PARAMETER
CMC_E_UNSUPPORTED_FUNCTION_EXT

7 Return codes

This clause defines the return codes of the CMC interface. The return codes of the generic interface are specified here; the return codes of the C interface are specified in Annex A, "C declaration summary". Tables 16 to 21 list the generic return codes and the functions to which the return codes pertain. Following the tables, each return code is defined.

The CMC 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.

TABLE 16/X.446 – SIMPLE CMC INTERFACE RETURN CODES

Return code	Act	Free	List	Logoff	Logon	Query	Read	Look	Send	SndDoc
CMC_E_ACCESS_DENIED	–	–	–	–	–	–	–	–	–	–
CMC_E_AMBIGUOUS_RECIPIENT	–	–	–	–	–	–	–	X	–	–
CMC_E_ATTACHMENT_NOT_FOUND	–	–	–	–	–	–	–	–	X	X
CMC_E_ATTACHMENT_OPEN_FAILURE	–	–	–	–	–	–	X	–	X	X
CMC_E_ATTACHMENT_READ_FAILURE	–	–	–	–	–	–	X	–	X	X
CMC_E_ATTACHMENT_WRITE_FAILURE	–	–	–	–	–	–	X	–	X	X
CMC_E_BIND_FAILURE	–	–	–	–	–	–	–	–	–	–
CMC_E_CALLBACK_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–	–
CMC_E_COUNTED_STRING_UNSUPPORTED	–	–	–	–	X	–	–	–	–	–
CMC_E_DISK_FULL	–	–	–	–	–	–	X	–	–	–
CMC_E_FAILURE	X	X	X	X	X	X	X	X	X	X
CMC_E_FUNCTION_INTERRUPTED	–	–	–	–	–	–	–	–	–	–
CMC_E_ID_NOT_FOUND	–	–	–	–	–	–	–	–	–	–
CMC_E_INSUFFICIENT_MEMORY	X	–	X	X	X	X	X	X	X	X
CMC_E_INVALID_CONFIGURATION	–	–	–	–	X	–	–	–	–	–
CMC_E_INVALID_CONTAINER_OBJECT	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_CURSOR_HANDLE	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_ENUM	X	–	–	–	X	X	–	–	–	–
CMC_E_INVALID_EVENT	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FILE_OFFSET	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FILE_SPECIFICATION	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FLAG	X	–	X	X	X	–	X	X	X	–
CMC_E_INVALID_FUNCTION_EXT	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MEMORY	–	X	–	–	–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_PARAMETER	–	–	–	–	–	–	–	–	X	–
CMC_E_INVALID_MESSAGE_REFERENCE	X	–	X	–	–	–	X	–	–	–
CMC_E_INVALID_OBJECT_HANDLE	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_PARAMETER	X	X	X	X	X	X	X	X	X	X
CMC_E_INVALID_PROPERTY_ID	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_PROPERTY_NAME	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_RESTRICTION	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_SESSION_ID	X	–	X	X	–	–	X	–	–	–
CMC_E_INVALID_SOURCE_OBJECT	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_STREAM_HANDLE	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_UI_ID	X	–	X	X	X	–	X	X	X	X

TABLE 16/X.446 – SIMPLE CMC INTERFACE RETURN CODES *(CONCLUDED)*

Return code	Act	Free	List	Logoff	Logon	Query	Read	Look	Send	SndDoc
CMC_E_INVALID_VALUE	–	–	–	–	–	–	–	–	–	–
CMC_E_LOGON_FAILURE	–	–	–	–	X	–	–	X	X	X
CMC_E_MESSAGE_IN_USE	X	–	–	–	–	–	–	–	–	–
CMC_E_NAME_NOT_FOUND	–	–	–	–	–	–	–	–	–	–
CMC_E_NO_EVENT	–	–	–	–	–	–	–	–	–	–
CMC_E_NO_MORE_BYTES_TO_WRITE	–	–	–	–	–	–	–	–	–	–
CMC_E_NOT_SUPPORTED	–	–	–	–	–	X	–	X	–	–
CMC_E_PASSWORD_REQUIRED	–	–	–	–	X	–	–	–	–	–
CMC_E_PROPERTY_DATA_TYPE_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_ID_NOT_FOUND	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_NAME_NOT_FOUND	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_PROBLEMS	–	–	–	–	–	–	–	–	–	–
CMC_E_RECIPIENT_NOT_FOUND	–	–	–	–	–	–	–	X	X	X
CMC_E_REQUIRED_PROPS_MISSING	–	–	–	–	–	–	–	–	–	–
CMC_E_RESTRICTION_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–	–
CMC_E_SERVICE_UNAVAILABLE	–	–	–	–	X	–	–	–	–	–
CMC_E_TEXT_TOO_LARGE	–	–	–	–	–	–	–	–	X	X
CMC_E_TOO_MANY_CONTENT_ITEMS	–	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_FILES	–	–	–	–	–	–	X	–	X	X
CMC_E_TOO_MANY_RECIPIENTS	–	–	–	–	–	–	–	–	X	X
CMC_E_UNABLE_TO_NOT_MARK_READ	–	–	–	–	–	–	X	–	–	–
CMC_E_UNBIND_FAILURE	–	–	–	–	–	–	–	–	–	–
CMC_E_UNRECOGNIZED_IDENTIFIER	–	–	–	–	–	–	–	–	–	–
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	–	–	X	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_ACTION	X	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_CHARACTER_SET	–	–	–	–	X	–	–	–	–	–
CMC_E_UNSUPPORTED_DATA_EXT	–	–	–	–	–	–	–	X	X	–
CMC_E_UNSUPPORTED_FLAG	X	–	X	X	X	–	X	X	X	–
CMC_E_UNSUPPORTED_FUNCTION_EXT	X	–	X	X	X	X	X	X	X	–
CMC_E_UNSUPPORTED_KEYS	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_VALUE	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_VERSION	–	–	–	–	X	–	–	–	–	–
CMC_E_USER_CANCEL	–	–	–	–	–	–	–	X	X	X
CMC_E_USER_NOT_LOGGED_ON	–	–	–	X	–	–	–	X	X	X

TABLE 17/X.446 – FULL CMC ADMINISTRATIVE AND BIND FUNCTION INTERFACE RETURN CODES

Return code	Free	Logoff	Logon		Bind	Unbind
CMC_E_ACCESS_DENIED	–	–	–		–	–
CMC_E_AMBIGUOUS_RECIPIENT	–	–	–		–	–
CMC_E_ATTACHMENT_NOT_FOUND	–	–	–		–	–
CMC_E_ATTACHMENT_OPEN_FAILURE	–	–	–		–	–
CMC_E_ATTACHMENT_READ_FAILURE	–	–	–		–	–
CMC_E_ATTACHMENT_WRITE_FAILURE	–	–	–		–	–
CMC_E_BIND_FAILURE	–	–	–		X	–
CMC_E_CALLBACK_NOT_SUPPORTED	–	–	–		–	–
CMC_E_COUNTED_STRING_UNSUPPORTED	–	–	X		–	–
CMC_E_DISK_FULL	–	–	–		–	–
CMC_E_FAILURE	X	X	X		X	X
CMC_E_FUNCTION_INTERRUPTED	–	–	–		–	–
CMC_E_ID_NOT_FOUND	–	–	–		X	X
CMC_E_INSUFFICIENT_MEMORY	–	X	X		X	X
CMC_E_INVALID_CONFIGURATION	–	–	X		–	–
CMC_E_INVALID_CONTAINER_OBJECT	–	–	–		–	–
CMC_E_INVALID_CURSOR_HANDLE	–	–	–		–	–
CMC_E_INVALID_ENUM	–	–	X		–	–
CMC_E_INVALID_EVENT	–	–	–		–	–
CMC_E_INVALID_FILE_OFFSET	–	–	–		–	–
CMC_E_INVALID_FILE_SPECIFICATION	–	–	–		–	–
CMC_E_INVALID_FLAG	–	X	X		–	–
CMC_E_INVALID_FUNCTION_EXT	–	–	–		–	–
CMC_E_INVALID_MEMORY	X	–	–		–	–
CMC_E_INVALID_MESSAGE_PARAMETER	–	–	–		–	–
CMC_E_INVALID_MESSAGE_REFERENCE	–	–	–		–	–
CMC_E_INVALID_OBJECT_HANDLE	–	–	–		–	–
CMC_E_INVALID_PARAMETER	X	X	X		X	X
CMC_E_INVALID_PROPERTY_ID	–	–	–		–	–
CMC_E_INVALID_PROPERTY_NAME	–	–	–		–	–
CMC_E_INVALID_RESTRICTION	–	–	–		–	–
CMC_E_INVALID_SESSION_ID	–	X	–		–	–
CMC_E_INVALID_SOURCE_OBJECT	–	–	–		–	–
CMC_E_INVALID_STREAM_HANDLE	–	–	–		–	–
CMC_E_INVALID_UI_ID	–	X	X		–	–

TABLE 17/X.446 – FULL CMC ADMINISTRATIVE AND BIND FUNCTION INTERFACE RETURN CODES *(CONCLUDED)*

Return code	Free	Logoff	Logon		Bind	Unbind
CMC_E_INVALID_VALUE	–	–	–		–	–
CMC_E_LOGON_FAILURE	–	–	X		–	–
CMC_E_MESSAGE_IN_USE	–	–	–		–	–
CMC_E_NAME_NOT_FOUND	–	–	–		–	–
CMC_E_NO_EVENT	–	–	–		–	–
CMC_E_NO_MORE_BYTES_TO_WRITE	–	–	–		–	–
CMC_E_NOT_SUPPORTED	–	–	–		–	–
CMC_E_PASSWORD_REQUIRED	–	–	X		–	–
CMC_E_PROPERTY_DATA_TYPE_NOT_SUPPORTED	–	–	–		–	–
CMC_E_PROPERTY_ID_NOT_FOUND	–	–	–		–	–
CMC_E_PROPERTY_NAME_NOT_FOUND	–	–	–		–	–
CMC_E_PROPERTY_PROBLEMS	–	–	–		–	–
CMC_E_RECIPIENT_NOT_FOUND	–	–	–		–	–
CMC_E_REQUIRED_PROPS_MISSING	–	–	–		–	–
CMC_E_RESTRICTION_NOT_SUPPORTED	–	–	–		–	–
CMC_E_SERVICE_UNAVAILABLE	–	–	X		–	–
CMC_E_TEXT_TOO_LARGE	–	–	–		–	–
CMC_E_TOO_MANY_CONTENT_ITEMS	–	–	–		–	–
CMC_E_TOO_MANY_FILES	–	–	–		–	–
CMC_E_TOO_MANY_RECIPIENTS	–	–	–		–	–
CMC_E_UNABLE_TO_NOT_MARK_READ	–	–	–		–	–
CMC_E_UNBIND_FAILURE	–	–	–		–	X
CMC_E_UNRECOGNIZED_IDENTIFIER	–	–	–		X	X
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	–	–	–		–	–
CMC_E_UNSUPPORTED_ACTION	–	–	–		–	–
CMC_E_UNSUPPORTED_CHARACTER_SET	–	–	X		–	–
CMC_E_UNSUPPORTED_DATA_EXT	–	–	–		–	–
CMC_E_UNSUPPORTED_FLAG	–	X	X		–	–
CMC_E_UNSUPPORTED_FUNCTION_EXT	–	X	X		–	–
CMC_E_UNSUPPORTED_KEYS	–	–	–		–	–
CMC_E_UNSUPPORTED_VALUE	–	–	–		–	–
CMC_E_UNSUPPORTED_VERSION	–	–	X		–	–
CMC_E_USER_CANCEL	–	–	–		–	–
CMC_E_USER_NOT_LOGGED_ON	–	X	–		–	–

TABLE 18/X.446 – FULL CMC COMPOSITION FUNCTION INTERFACE RETURN CODES

Return code	Add Props	Comt Obj	Copy Obj	Copy Obj Hdl	Del Objs	Del Props	Open Obj Hdl	Restore Obj	Save Obj
CMC_E_ACCESS_DENIED	–	X	–	–	X	–	–	X	X
CMC_E_AMBIGUOUS_RECIPIENT	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_NOT_FOUND	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_OPEN_FAILURE	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_READ_FAILURE	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_WRITE_FAILURE	–	–	–	–	–	–	–	–	–
CMC_E_BIND_FAILURE	–	–	–	–	–	–	–	–	–
CMC_E_CALLBACK_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–
CMC_E_COUNTED_STRING_UNSUPPORTED	–	–	–	–	–	–	–	–	–
CMC_E_DISK_FULL	–	X	–	–	–	–	–	–	X
CMC_E_FAILURE	X	X	X	X	X	X	X	X	X
CMC_E_FUNCTION_INTERRUPTED	–	–	–	–	–	–	–	–	–
CMC_E_ID_NOT_FOUND	–	–	–	–	–	–	–	–	–
CMC_E_INSUFFICIENT_MEMORY	X	X	X	X	X	X	X	X	X
CMC_E_INVALID_CONFIGURATION	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_CONTAINER_OBJECT	–	–	X	–	–	–	–	X	–
CMC_E_INVALID_CURSOR_HANDLE	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_ENUM	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_EVENT	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FILE_OFFSET	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FILE_SPECIFICATION	–	–	–	–	–	–	–	X	X
CMC_E_INVALID_FLAG	–	–	–	–	–	–	–	X	X
CMC_E_INVALID_FUNCTION_EXT	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MEMORY	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_PARAMETER	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_REFERENCE	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_OBJECT_HANDLE	X	X	–	X	X	X	–	X	X
CMC_E_INVALID_PARAMETER	X	X	X	X	X	X	X	X	X
CMC_E_INVALID_PROPERTY_ID	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_PROPERTY_NAME	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_RESTRICTION	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_SESSION_ID	–	–	–	–	–	–	X	–	–
CMC_E_INVALID_SOURCE_OBJECT	–	–	X	–	–	–	–	–	–
CMC_E_INVALID_STREAM_HANDLE	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_UI_ID	–	–	–	–	–	–	–	–	–

TABLE 18/X.446 – FULL CMC COMPOSITION FUNCTION INTERFACE RETURN CODES *(CONCLUDED)*

Return code	Add Props	Comt Obj	Copy Obj	Copy Obj Hdl	Del Objs	Del Props	Open Obj Hdl	Restore Obj	Save Obj
CMC_E_INVALID_VALUE	–	–	–	–	–	–	–	–	–
CMC_E_LOGON_FAILURE	–	–	–	–	–	–	–	–	–
CMC_E_MESSAGE_IN_USE	–	–	–	–	–	–	–	–	–
CMC_E_NAME_NOT_FOUND	–	–	–	–	–	–	–	–	–
CMC_E_NO_EVENT	–	–	–	–	–	–	–	–	–
CMC_E_NO_MORE_BYTES_TO_WRITE	–	–	–	–	–	–	–	–	–
CMC_E_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–
CMC_E_PASSWORD_REQUIRED	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_DATA_TYPE_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_ID_NOT_FOUND	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_NAME_NOT_FOUND	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_PROBLEMS	–	–	–	–	–	–	–	–	–
CMC_E_RECIPIENT_NOT_FOUND	–	–	–	–	–	–	–	–	–
CMC_E_REQUIRED_PROPS_MISSING	–	–	–	–	–	–	–	–	–
CMC_E_RESTRICTION_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–
CMC_E_SERVICE_UNAVAILABLE	–	–	–	–	–	–	–	–	–
CMC_E_TEXT_TOO_LARGE	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_CONTENT_ITEMS	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_FILES	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_RECIPIENTS	–	–	–	–	–	–	–	–	–
CMC_E_UNABLE_TO_NOT_MARK_READ	–	–	–	–	–	–	–	–	–
CMC_E_UNBIND_FAILURE	–	–	–	–	–	–	–	–	–
CMC_E_UNRECOGNIZED_IDENTIFIER	–	–	–	–	–	–	X	–	–
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_ACTION	–	X	X	–	–	–	–	X	–
CMC_E_UNSUPPORTED_CHARACTER_SET	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_DATA_EXT	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_FLAG	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_FUNCTION_EXT	X	X	–	X	X	X	X	X	X
CMC_E_UNSUPPORTED_KEYS	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_VALUE	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_VERSION	–	–	–	–	–	–	–	–	–
CMC_E_USER_CANCEL	–	–	–	–	–	–	–	–	–
CMC_E_USER_NOT_LOGGED_ON	–	–	–	–	–	–	–	–	–

TABLE 19/X.446 – FULL CMC ENUMERATION FUNCTION INTERFACE RETURN CODES

Return code	Get Last Err	Get Root Hdle	List Cont Props	List No Matched	List Objs	List Props	Open Cur	Read Cur	Read Props	Read Prop Costs	Upd Cur Pos	Upd Cur Pos w/ Sd
CMC_E_ACCESS_DENIED	–	X	–	–	–	–	–	–	–	–	–	–
CMC_E_AMBIGUOUS_RECIPIENT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_NOT_FOUND	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_OPEN_FAILURE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_READ_FAILURE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_ATTACHMENT_WRITE_FAILURE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_BIND_FAILURE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_CALLBACK_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_COUNTED_STRING_ UNSUPPORTED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_DISK_FULL	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_FAILURE	X	X	X	X	X	X	X	X	X	X	X	X
CMC_E_FUNCTION_INTERRUPTED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_ID_NOT_FOUND	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INSUFFICIENT_MEMORY	X	X	X	X	X	X	X	X	X	X	–	–
CMC_E_INVALID_CONFIGURATION	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_CONTAINER_OBJECT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_CURSOR_HANDLE	–	–	X	X	X	–	–	–	–	–	X	X
CMC_E_INVALID_ENUM	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_EVENT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FILE_OFFSET	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FILE_SPECIFICATION	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FLAG	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_FUNCTION_EXT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MEMORY	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_PARAMETER	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_REFERENCE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_OBJECT_HANDLE	X	–	–	–	–	X	X	X	X	X	–	X
CMC_E_INVALID_PARAMETER	X	X	X	X	X	X	X	X	X	X	X	X
CMC_E_INVALID_PROPERTY_ID	–	–	X	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_PROPERTY_NAME	–	–	–	–	–	X	–	X	X	X	–	–
CMC_E_INVALID_RESTRICTION	–	–	–	–	–	–	X	–	–	–	–	–
CMC_E_INVALID_SESSION_ID	X	X	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_SOURCE_OBJECT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_STREAM_HANDLE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_INVALID_UI_ID	–	–	–	–	–	–	–	–	–	–	–	–

TABLE 19/X.446 – FULL CMC ENUMERATION FUNCTION INTERFACE RETURN CODES *(CONCLUDED)*

Return code	**Get Last Err**	**Get Root Hdle**	**List Cont Props**	**List No Matched**	**List Objs**	**List Props**	**Open Cur**	**Read Cur**	**Read Props**	**Read Prop Costs**	**Upd Cur Pos**	**Upd Cur Pos w/ Sd**
CMC_E_INVALID_VALUE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_LOGON_FAILURE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_MESSAGE_IN_USE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_NAME_NOT_FOUND	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_NO_EVENT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_NO_MORE_BYTES_TO_WRITE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_PASSWORD_REQUIRED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_DATA_TYPE_NOT_ SUPPORTED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_ID_NOT_FOUND	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_NAME_NOT_FOUND	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_PROPERTY_PROBLEMS	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_RECIPIENT_NOT_FOUND	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_REQUIRED_PROPS_MISSING	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_RESTRICTION_NOT_SUPPORTED	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_SERVICE_UNAVAILABLE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_TEXT_TOO_LARGE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_CONTENT_ITEMS	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_FILES	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_TOO_MANY_RECIPIENTS	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNABLE_TO_NOT_MARK_READ	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNBIND_FAILURE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNRECOGNIZED_IDENTIFIER	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_ACTION	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_CHARACTER_SET	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_DATA_EXT	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_FLAG	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_FUNCTION_EXT	X	X	X	X	X	X	X	X	X	X	X	X
CMC_E_UNSUPPORTED_KEYS	–	–	–	–	–	–	X	–	–	–	–	–
CMC_E_UNSUPPORTED_VALUE	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_UNSUPPORTED_VERSION	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_USER_CANCEL	–	–	–	–	–	–	–	–	–	–	–	–
CMC_E_USER_NOT_LOGGED_ON	–	–	–	–	–	–	–	–	–	–	–	–

TABLE 20/X.446 – FULL CMC EVENT NOTIFICATION AND MESSAGING FUNCTION INTERFACE RETURN CODES

Return code	Ck Event	Reg Event	Unreg Event	Call Clbks		Cr Der Msg	Snd Msg Obj
CMC_E_ACCESS_DENIED	–	–	–	–		–	–
CMC_E_AMBIGUOUS_RECIPIENT	–	–	–	–		–	–
CMC_E_ATTACHMENT_NOT_FOUND	–	–	–	–		–	–
CMC_E_ATTACHMENT_OPEN_FAILURE	–	–	–	–		–	–
CMC_E_ATTACHMENT_READ_FAILURE	–	–	–	–		–	–
CMC_E_ATTACHMENT_WRITE_FAILURE	–	–	–	–		–	–
CMC_E_BIND_FAILURE	–	–	–	–		–	–
CMC_E_CALLBACK_NOT_SUPPORTED	–	X	–	–		–	–
CMC_E_COUNTED_STRING_UNSUPPORTED	–	–	–	–		–	–
CMC_E_DISK_FULL	–	–	–	–		–	–
CMC_E_FAILURE	X	X	X	X		X	X
CMC_E_FUNCTION_INTERRUPTED	X	–	–	–		–	–
CMC_E_ID_NOT_FOUND	–	–	–	–		–	–
CMC_E_INSUFFICIENT_MEMORY	X	X	X	X		–	–
CMC_E_INVALID_CONFIGURATION	–	–	–	–		–	–
CMC_E_INVALID_CONTAINER_OBJECT	–	–	–	–		–	–
CMC_E_INVALID_CURSOR_HANDLE	–	–	–	–		–	–
CMC_E_INVALID_ENUM	–	–	–	–		–	–
CMC_E_INVALID_EVENT	X	X	X	X			
CMC_E_INVALID_FILE_OFFSET	–	–	–	–		–	–
CMC_E_INVALID_FILE_SPECIFICATION	–	–	–	–		–	–
CMC_E_INVALID_FLAG	–	–	–	–		–	–
CMC_E_INVALID_FUNCTION_EXT	X	X	X	X		–	–
CMC_E_INVALID_MEMORY	–	–	–	–		–	–
CMC_E_INVALID_MESSAGE_PARAMETER	–	–	–	–		–	–
CMC_E_INVALID_MESSAGE_REFERENCE	–	–	–	–		–	–
CMC_E_INVALID_OBJECT_HANDLE	–	–	–	–		X	X
CMC_E_INVALID_PARAMETER	X	X	X	X		X	X
CMC_E_INVALID_PROPERTY_ID	–	–	–	–		–	–
CMC_E_INVALID_PROPERTY_NAME	–	–	–	–		–	–
CMC_E_INVALID_RESTRICTION	–	–	–	–		–	–
CMC_E_INVALID_SESSION_ID	X	X	X	X		–	–
CMC_E_INVALID_SOURCE_OBJECT	–	–	–	–		–	–
CMC_E_INVALID_STREAM_HANDLE	–	–	–	–		–	–
CMC_E_INVALID_UI_ID	–	–	–	–		–	–

TABLE 20/X.446 – FULL CMC EVENT NOTIFICATION AND MESSAGING FUNCTION INTERFACE RETURN CODES *(CONCLUDED)*

Return code	Ck Event	Reg Event	Unreg Event	Call Clbks		Cr Der Msg	Snd Msg Obj
CMC_E_INVALID_VALUE	–	–	–	–		–	–
CMC_E_LOGON_FAILURE	–	–	–	–		–	–
CMC_E_MESSAGE_IN_USE	–	–	–	–		–	–
CMC_E_NAME_NOT_FOUND	–	–	–	–		–	–
CMC_E_NO_EVENT	X	–	–	–		–	–
CMC_E_NO_MORE_BYTES_TO_WRITE	–	–	–	–		–	–
CMC_E_NOT_SUPPORTED	–	–	X	X		–	–
CMC_E_PASSWORD_REQUIRED	–	–	–	–		–	–
CMC_E_PROPERTY_DATA_TYPE_NOT_ SUPPORTED	–	–	–	–		–	–
CMC_E_PROPERTY_ID_NOT_FOUND	–	–	–	–		–	–
CMC_E_PROPERTY_NAME_NOT_FOUND	–	–	–	–		–	–
CMC_E_PROPERTY_PROBLEMS	–	–	–	–		–	–
CMC_E_RECIPIENT_NOT_FOUND	–	–	–	–		–	–
CMC_E_REQUIRED_PROPS_MISSING	–	–	–	–		X	X
CMC_E_RESTRICTION_NOT_SUPPORTED	–	–	–	–		–	–
CMC_E_SERVICE_UNAVAILABLE	–	–	–	X		–	–
CMC_E_TEXT_TOO_LARGE							
CMC_E_TOO_MANY_CONTENT_ITEMS	–	–	–	–		–	–
CMC_E_TOO_MANY_FILES	–	–	–	–		–	–
CMC_E_TOO_MANY_RECIPIENTS	–	–	–	–		–	–
CMC_E_UNABLE_TO_NOT_MARK_READ	–	–	–	–		–	–
CMC_E_UNBIND_FAILURE	–	–	–	–		–	–
CMC_E_UNRECOGNIZED_IDENTIFIER	–	–	–	–		–	–
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	–	–	–	–		–	–
CMC_E_UNSUPPORTED_ACTION	–	–	–	–		X	–
CMC_E_UNSUPPORTED_CHARACTER_SET	–	–	–	–		–	–
CMC_E_UNSUPPORTED_DATA_EXT	–	–	–	–		–	–
CMC_E_UNSUPPORTED_FLAG	–	–	–	X		–	–
CMC_E_UNSUPPORTED_FUNCTION_EXT	X	X	X	X		X	X
CMC_E_UNSUPPORTED_KEYS	–	–	–	–		–	–
CMC_E_UNSUPPORTED_VALUE	–	–	–	–		–	–
CMC_E_UNSUPPORTED_VERSION	–	–	–	–		–	–
CMC_E_USER_CANCEL	–	–	–	–		–	–
CMC_E_USER_NOT_LOGGED_ON	–	–	–	–		–	–

TABLE 21/X.446 – FULL CMC NAME HANDLING AND STREAM FUNCTION INTERFACE CODES

Return code	Id to Name	Name to Id		Exp Str	Imp Str	Open Str	Read Str	Seek Str	Wrt Str
CMC_E_ACCESS_DENIED	–	–		X	X	–	X	X	X
CMC_E_AMBIGUOUS_RECIPIENT	–	–		–	–	–	–	–	–
CMC_E_ATTACHMENT_NOT_FOUND	–	–		–	–	–	–	–	–
CMC_E_ATTACHMENT_OPEN_FAILURE	–	–		–	–	–	–	–	–
CMC_E_ATTACHMENT_READ_FAILURE	–	–		–	–	–	–	–	–
CMC_E_ATTACHMENT_WRITE_FAILURE	–	–		–	–	–	–	–	–
CMC_E_BIND_FAILURE	–	–		–	–	–	–	–	–
CMC_E_CALLBACK_NOT_SUPPORTED	–	–		–	–	–	–	–	–
CMC_E_COUNTED_STRING_UNSUPPORTED	–	–		–	–	–	–	–	–
CMC_E_DISK_FULL	–	–		X	–	–	–	–	–
CMC_E_FAILURE	X	X		X	X	X	X	X	X
CMC_E_FUNCTION_INTERRUPTED	–	–		–	–	–	–	–	–
CMC_E_ID_NOT_FOUND	–	–		–	–	–	–	–	–
CMC_E_INSUFFICIENT_MEMORY	X	X		X	X	X	X	X	X
CMC_E_INVALID_CONFIGURATION	–	–		–	–	–	–	–	–
CMC_E_INVALID_CONTAINER_OBJECT	–	–		–	–	–	–	–	–
CMC_E_INVALID_CURSOR_HANDLE	–	–		–	–	–	–	–	–
CMC_E_INVALID_ENUM	–	–		–	–	–	–	–	–
CMC_E_INVALID_EVENT	–	–		–	–	–	–	–	–
CMC_E_INVALID_FILE_OFFSET	–	–		–	X	–	–	–	–
CMC_E_INVALID_FILE_SPECIFICATION	–	–		X	X	–	–	–	–
CMC_E_INVALID_FLAG	–	–		X	X	–	–	–	–
CMC_E_INVALID_FUNCTION_EXT	–	–		–	–	–	–	–	–
CMC_E_INVALID_MEMORY	–	–		–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_PARAMETER	–	–		–	–	–	–	–	–
CMC_E_INVALID_MESSAGE_REFERENCE	–	–		–	–	–	–	–	–
CMC_E_INVALID_OBJECT_HANDLE	–	–		–	–	X	–	–	–
CMC_E_INVALID_PARAMETER	X	X		X	X	–	X	X	X
CMC_E_INVALID_PROPERTY_ID	X	–	–	–	–	X	–	–	–
CMC_E_INVALID_PROPERTY_NAME	–	X		–	–	–	–	–	–
CMC_E_INVALID_RESTRICTION	–	–		–	–	–	–	–	–
CMC_E_INVALID_SESSION_ID	–	–		–	–	–	–	–	–
CMC_E_INVALID_SOURCE_OBJECT	–	–		–	–	–	–	–	–
CMC_E_INVALID_STREAM_HANDLE	–	–		X	X	–	X	X	X
CMC_E_INVALID_UI_ID	–	–		–	–	–	–	–	–

TABLE 21/X.446 – FULL CMC NAME HANDLING AND STREAM FUNCTION INTERFACE CODES *(CONCLUDED)*

Return code	Id to Name	Name to Id		Exp Str	Imp Str	Open Str	Read Str	Seek Str	Wrt Str
CMC_E_INVALID_VALUE	–	–		–	–	–	–	–	–
CMC_E_LOGON_FAILURE	–	–		–	–	–	–	–	–
CMC_E_MESSAGE_IN_USE	–	–		–	–	–	–	–	–
CMC_E_NAME_NOT_FOUND	–	–		–	–	–	–	–	–
CMC_E_NO_EVENT	–	–		–	–	–	–	–	–
CMC_E_NO_MORE_BYTES_TO_WRITE	–	–		–	–	–	–	–	X
CMC_E_NOT_SUPPORTED	–	–		–	–	–	–	–	–
CMC_E_PASSWORD_REQUIRED	–	–		–	–	–	–	–	–
CMC_E_PROPERTY_DATA_TYPE_NOT_ SUPPORTED	–	–		–	–	–	–	–	–
CMC_E_PROPERTY_ID_NOT_FOUND	–	X		–	–	–	–	–	–
CMC_E_PROPERTY_NAME_NOT_FOUND	X	–		–	–	–	–	–	–
CMC_E_PROPERTY_PROBLEMS	–	–		–	–	–	–	–	–
CMC_E_RECIPIENT_NOT_FOUND	–	–		–	–	–	–	–	–
CMC_E_REQUIRED_PROPS_MISSING	–	–		–	–	–	–	–	–
CMC_E_RESTRICTION_NOT_SUPPORTED	–	–		–	–	–	–	–	–
CMC_E_SERVICE_UNAVAILABLE	–	–		–	–	–	–	–	–
CMC_E_TEXT_TOO_LARGE	–	–		–	–	–	–	–	–
CMC_E_TOO_MANY_CONTENT_ITEMS	–	–		–	–	–	–	–	–
CMC_E_TOO_MANY_FILES	–	–		–	–	–	–	–	–
CMC_E_TOO_MANY_RECIPIENTS	–	–		–	–	–	–	–	–
CMC_E_UNABLE_TO_NOT_MARK_READ	–	–		–	–	–	–	–	–
CMC_E_UNBIND_FAILURE	–	–		–	–	–	–	–	–
CMC_E_UNRECOGNIZED_IDENTIFIER	–	–		–	–	–	–	–	–
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_ACTION	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_CHARACTER_SET	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_DATA_EXT	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_FLAG	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_FUNCTION_EXT	X	X		X	X	X	X	X	X
CMC_E_UNSUPPORTED_KEYS	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_VALUE	–	–		–	–	–	–	–	–
CMC_E_UNSUPPORTED_VERSION	–	–		–	–	–	–	–	–
CMC_E_USER_CANCEL	–	–		–	–	–	–	–	–
CMC_E_USER_NOT_LOGGED_ON	–	–		–	–	–	–	–	–

The return codes are defined as follows:

CMC_E_ACCESS_DENIED	Access has been denied.
CMC_E_AMBIGUOUS_RECIPIENT	The recipient name is ambiguous; multiple matches have been found.
CMC_E_ATTACHMENT_NOT_FOUND	The specified attachment was not found as specified.
CMC_E_ATTACHMENT_OPEN_FAILURE	The specified attachment was found but could not be opened, or the attachment file could not be created.
CMC_E_ATTACHMENT_READ_FAILURE	The specified attachment was found and opened, but there was an error reading it.
CMC_E_ATTACHMENT_WRITE_FAILURE	The attachment file was created successfully, but there was an error writing it.
CMC_E_BIND_FAILURE	Unable to bind application to implementation.
CMC_E_CALLBACK_NOT_SUPPORTED	Specified callback not supported by implementation.
CMC_E_COUNTED_STRING_UNSUPPORTED	This implementation does not support the counted string type.
CMC_E_DISK_FULL	Insufficient disk space was available to complete the requested operation (this may refer to local or shared disk space).
CMC_E_FAILURE	There was a general failure which does not fit the description of any other error code.
CMC_E_FUNCTION_INTERRUPTED	The function has been interrupted.
CMC_E_ID_NOT_FOUND	The specified id was not found.
CMC_E_INSUFFICIENT_MEMORY	Insufficient memory was available to complete the requested operation.
CMC_E_INVALID_CONFIGURATION	The underlying messaging service's configuration is invalid, so logging on cannot be completed.
CMC_E_INVALID_CONTAINER_OBJECT	An invalid container object was specified.
CMC_E_INVALID_CURSOR_HANDLE	An invalid cursor handle was specified.
CMC_E_INVALID_ENUM	A CMC_enum value is invalid.
CMC_E_INVALID_EVENT	Invalid event specified.
CMC_E_INVALID_FILE_OFFSET	An invalid file offset was specified.
CMC_E_INVALID_FILE_SPECIFICATION	An invalid file was specified.
CMC_E_INVALID_FLAG	A flag value in the flags parameter was invalid.
CMC_E_INVALID_FUNCTION_EXT	The function extension is invalid.
CMC_E_INVALID_MEMORY	Memory pointer passed is invalid.
CMC_E_INVALID_MESSAGE_PARAMETER	One of the parameters in the message was invalid.

CMC_E_INVALID_MESSAGE_REFERENCE	The specified message reference is invalid or no longer valid (e.g. it has been deleted).
CMC_E_INVALID_OBJECT_HANDLE	An invalid object handle was specified.
CMC_E_INVALID_PARAMETER	A function parameter was invalid.
CMC_E_INVALID_PROPERTY_ID	An invalid property identifier was specified.
CMC_E_INVALID_PROPERTY_NAME	The specified property name is invalid.
CMC_E_INVALID_RESTRICTION	An invalid restriction was specified.
CMC_E_INVALID_SESSION_ID	The specified session id is invalid or no longer valid (e.g. after logging off).
CMC_E_INVALID_SOURCE_OBJECT	An invalid source object was specified.
CMC_E_INVALID_STREAM_HANDLE	An invalid stream handle was specified.
CMC_E_INVALID_UI_ID	The specified user interface id is invalid or no longer valid.
CMC_E_INVALID_VALUE	The value is not valid.
CMC_E_LOGON_FAILURE	The service, user name, and/or password specified were invalid, so logging on cannot be completed.
CMC_E_MESSAGE_IN_USE	The requested action cannot be completed at this time because the message is in use.
CMC_E_NAME_NOT_FOUND	The specified name was not found.
CMC_E_NO_EVENT	The specified event does not exist.
CMC_E_NO_MORE_BYTES_TO_WRITE	There are no more bytes to write to the stream.
CMC_E_NOT_SUPPORTED	The operation requested is not supported by this implementation.
CMC_E_PASSWORD_REQUIRED	A password is required on this messaging service.
CMC_E_PROPERTY_DATA_TYPE_NOT_ SUPPORTED	The property data type is not supported by this implementation.
CMC_E_PROPERTY_ID_NOT_FOUND	The specified property identifier was not found.
CMC_E_PROPERTY_NAME_NOT_FOUND	The specified property name was not found.
CMC_E_PROPERTY_PROBLEMS	Problems exist with the properties.
CMC_E_RECIPIENT_NOT_FOUND	One or more of the specified recipients were not found.
CMC_E_REQUIRED_PROPS_MISSING	One or more of the specified properties are missing.
CMC_E_RESTRICTION_NOT_SUPPORTED	The specified restriction is too complex and is not supported by the implementation.

CMC_E_SERVICE_UNAVAILABLE	The service requested is unavailable.
CMC_E_TEXT_TOO_LARGE	The size of the text string passed to the implementation is too large.
CMC_E_TOO_MANY_CONTENT_ITEMS	Maximum number of acceptable content items exceeded.
CMC_E_TOO_MANY_FILES	The implementation cannot support the number of files specified.
CMC_E_TOO_MANY_RECIPIENTS	The implementation cannot support the number of recipients specified.
CMC_E_UNABLE_TO_NOT_MARK_READ	CMC_E_UNABLE_TO_NOT_MARK_READ flag cannot be supported.
CMC_E_UNBIND_FAILURE	Failure encountered while attempting to unbind application from implementation.
CMC_E_UNRECOGNIZED_IDENTIFIER	The specified identifier was unrecognized.
CMC_E_UNRECOGNIZED_MESSAGE_TYPE	The specified message type is not supported by this implementation.
CMC_E_UNSUPPORTED_ACTION	The requested action is not supported by this implementation.
CMC_E_UNSUPPORTED_CHARACTER_SET	The character set requested is not supported.
CMC_E_UNSUPPORTED_DATA_EXT	The data extension requested is not supported.
CMC_E_UNSUPPORTED_FLAG	The flag requested is not supported.
CMC_E_UNSUPPORTED_FUNCTION_EXT	The function extension requested is not supported.
CMC_E_UNSUPPORTED_KEYS	The specified sort keys are not supported.
CMC_E_UNSUPPORTED_VALUE	The value is not supported.
CMC_E_UNSUPPORTED_VERSION	The version specified in the call cannot be supported by this CMC implementation.
CMC_E_USER_CANCEL	The operation was cancelled by the user.
CMC_E_USER_NOT_LOGGED_ON	The user is not logged on and the CMC_E_USER_NOT_LOGGED_ON flag is not set.

8 Conformance

In order for an implementation of the Common Messaging Call API to conform to this Recommendation it must meet the following criteria:

- All functions and data structures must be implemented as defined. Statements elsewhere in the Recommendation which describe features as optional or with exceptions take precedence over this criterion.
- The implementation must be able to transport at least the CMC IPM message type.
- Support for XAPIA's CMC 1.0 applications is recommended for Simple CMC and Full CMC implementations.
- Support for Full CMC and Simple CMC is mandatory for Full CMC implementations.

- All object classes in clause 3 must be implemented as defined. Statements elsewhere in the Recommendation which describe features as optional or with exceptions take precedence over this criterion.
- Object properties designated as mandatory in the property characteristic tables shall be supported.
- Character set support is up to the underlying implementation. Support for an implementation-defined default character set is required. Optionally, other character sets may be supported. Counted string support is not required.
- All extensions are optional. Vendors are encouraged to support the CMC-defined standard extension set specified in this Recommendation. It is further encouraged that standard extension sets are developed for any proprietary or non-proprietary messaging services for which a CMC interface is provided, to accommodate features specific to that messaging service, and that the extension set can be registered externally.
- Minimum conformance for an extension set will be defined by the creator of the extension set.
- The CMC Manager and CMC Implementation must provide an implementation of the CMC Bind Implementation() and CMC Unbind Implementation() calls, and must return a pointer to the dispatch table on the CMC Bind Implementation() call. The CMC Manager, if multiple implementations are supported, could provide a means of enumerating the known CMC implementations on a given platform (optional browsing capability), and a means of registering the CMC implementations.
- CMC implementations must be able to support calls directly to its function as well as indirectly through the dispatch table.

Annex A

C declaration summary

A.1 C declaration summary

This subclause lists the declarations that define the CMC interface for the C programming language. All of the declarations, except those for symbolic constants, also appear in clause 4, Data Structures or clause 6, Functional interface.

The declarations assembled here constitute the contents of a header file to be made accessible to application programmers. The header file is **<xcmc.h>**. The symbols the declarations define are the only symbols the service makes visible to the application.

```
/*BEGIN CMC 2.0 INTERFACE*/

#ifndef_XCMC_H
#define_XCMC_H

#ifdef_cplusplus
extern "C" {
#endif

/*BASIC DATA TYPES*/
#ifndef DIFFERENT_PLATFORM
typedef char                     CMC_sint8;
typedef short                    CMC_sint16;
typedef long int                 CMC_sint32;
typedef unsigned short int       CMC_uint16;
typedef unsigned long int        CMC_uint32;
typedef void *                   CMC_buffer;
typedef unsigned char            CMC_byte;
typedef long int                 CMC_size;
typedef float                    CMC_float32;
typedef double                   CMC_float64;

/*CHARACTER SIZE DEFINITION*/
#ifndef CMC_WCHAR
#define CMC_CHAR                 char
#else
#define CMC_CHAR                 CMC_sint16
#endif
typedef CMC_CHAR *               CMC_string;
#else
typedef CMC_CHAR                 char
typedef CMC_CHAR *               CMC_string;
#endif

typedef CMC_uint16               CMC_boolean;
typedef CMC_sint32               CMC_enum;
typedef CMC_uint32               CMC_return_code;
typedef CMC_uint32               CMC_flags;
typedef CMC_string               CMC_object_identifier;
typedef CMC_string               CMC_guid;
typedef CMC_string               CMC_date_time;

#define CMC_FALSE                ((CMC_boolean) 0)
#define CMC_TRUE                 ((CMC_boolean) 1)

/*DATA STRUCTURES*/

/*COUNTED STRING*/
typedef struct {
      CMC_uint32                 length;
      CMC_CHAR                   string[1];
} CMC_counted_string;

/*SESSION ID*/
typedef CMC_uint32               CMC_session_id;
```

```
#ifndef DIFFERENT_PLATFORM
/*CURSOR HANDLE*/
typedef CMC_uint32              CMC_cursor_handle;

/*OBJECT HANDLE*/
typedef CMC_uint32              CMC_object_handle;

/*STREAM HANDLE*/
typedef CMC_uint32              CMC_stream_handle;

/*NULLHANDLE*/
#define CMC_NULL_OBJECT_HANDLE ((CMC_object_handle) 0)
#endif

/*OPAQUE DATA*/
typedef struct CMC_TAG_OPAQUE_DATA {
      CMC_size                  size;
      CMC_byte                  *data;
} CMC_opaque_data;

/*TIME*/
/* unusedX fields needed to align struct on 4-byte boundary */
typedef struct {
      CMC_sint8                 second;
      CMC_sint8                 minute;
      CMC_sint8                 hour;
      CMC_sint8                 day;
      CMC_sint8                 month;
      CMC_sint8                 year;
      CMC_sint8                 isdst;
      CMC_sint8                 unused1;
      CMC_sint16                tmzone;
      CMC_sint16                unused2;
} CMC_time, CMC_iso_date_time;

#define CMC_NO_TIMEZONE         ((CMC_sint16) 0x8000)

/*UI ID*/
typedef CMC_uint32              CMC_ui_id;

/*EXTENSION*/
typedef struct {
      CMC_uint32                item_code;
      CMC_uint32                item_data;
      CMC_buffer                item_reference;
      CMC_flags                 extension_flags;
} CMC_extension;

/*PROPERTY ID*/
typedef CMC_uint32              CMC_id;

/*PROPERTY NAME*/
typedef CMC_string              CMC_name;

/*MULTIVALUED PROPERTY DEFINITIONS*/

typedef struct CMC_TAG_ARRAY_BOOLEAN {
      CMC_uint32                count;
      CMC_boolean               *bits;
} CMC_array_boolean;

typedef struct CMC_TAG_ARRAY_BUFFER {
      CMC_uint32                count;
      CMC_buffer                *buffer;
} CMC_array_buffer;

typedef struct CMC_TAG_ARRAY_COUNTED_STRING {
      CMC_uint32                count;
      CMC_counted_string        *string;
} CMC_array_counted_string;

typedef struct CMC_TAG_ARRAY_ENUM {
      CMC_uint32                count;
      CMC_enum                  *set;
} CMC_array_enum;
```

```
typedef struct CMC_TAG_ARRAY_EXTENSION {
      CMC_uint32               count;
      CMC_extension            *extension;
} CMC_array_extension;

typedef struct CMC_TAG_ARRAY_FLOAT32 {
      CMC_uint32               count;
      CMC_float32              *number;
} CMC_array_float32;

typedef struct CMC_TAG_ARRAY_FLOAT64 {
      CMC_uint32               count;
      CMC_float64              *number;
} CMC_array_float64;

typedef struct CMC_TAG_ARRAY_GUID {
      CMC_uint32               count;
      CMC_guid                 *guid;
} CMC_array_guid;

typedef struct CMC_TAG_ARRAY_ISO_DATE_TIME {
      CMC_uint32               count;
      CMC_date_time            *time;
} CMC_array_iso_date_time;

typedef struct CMC_TAG_ARRAY_OBJECT_HANDLE {
      CMC_uint32               count;
      CMC_object_handle        *ohandles;
} CMC_array_object_handle;

typedef struct CMC_TAG_ARRAY_OPAQUE_DATA {
      CMC_uint32               count;
      CMC_opaque_data          *data;
} CMC_array_opaque_data;

typedef struct CMC_TAG_ARRAY_RETURN_CODE {
      CMC_uint32               count;
      CMC_return_code          *code;
} CMC_array_return_code;

typedef struct CMC_TAG_ARRAY_SINT16 {
      CMC_uint32               count;
      CMC_sint16               *number;
} CMC_array_sint16;

typedef struct CMC_TAG_ARRAY_SINT32 {
      CMC_uint32               count;
      CMC_sint32               *number;
} CMC_array_sint32;

typedef struct CMC_TAG_ARRAY_STRING {
      CMC_uint32               count;
      CMC_string               *string;
} CMC_array_string;

typedef struct CMC_TAG_ARRAY_TIME {
      CMC_uint32               count;
      CMC_time                 *time;
} CMC_array_time;

typedef struct CMC_TAG_ARRAY_UINT16 {
      CMC_uint32               count;
      CMC_uint16               *number;
} CMC_array_uint16;

typedef struct CMC_TAG_ARRAY_UINT32 {
      CMC_uint32               count;
      CMC_uint32               *number;
} CMC_array_uint32;
```

```
/*PROPERTY*/
typedef struct CMC_TAG_PROPERTY {
      CMC_id                    property_id;
      CMC_enum                  type;
      union {
                  CMC_boolean                 CMC_pv_boolean;
                  CMC_byte                    CMC_pv_byte;
                  CMC_buffer                  CMC_pv_buffer;
                  CMC_counted_string          CMC_pv_counted_string;
                  CMC_enum                    CMC_pv_enumerated;
                  CMC_extension               CMC_pv_extension;
                  CMC_float32                 CMC_pv_float32;
                  CMC_float64                 CMC_pv_float64;
                  CMC_flags                   CMC_pv_flags;
                  CMC_guid                    CMC_pv_guid;
                  CMC_iso_date_time           CMC_pv_iso_date_time;
                  CMC_object_handle           CMC_pv_object_handle;
                  CMC_opaque_data             CMC_pv_opaque_data;
                  CMC_return_code             CMC_pv_return_code;
                  CMC_sint16                  CMC_pv_sint16;
                  CMC_sint32                  CMC_pv_sint32;
                  CMC_string                  CMC_pv_string;
                  CMC_time                    CMC_pv_time;
                  CMC_uint16                  CMC_pv_uint16;
                  CMC_uint32                  CMC_pv_uint32;
                  CMC_array_boolean           CMC_pv_array_boolean;
                  CMC_array_buffer            CMC_pv_array_buffer;
                  CMC_array_counted_string    CMC_pv_array_counted_string;
                  CMC_array_enum              CMC_pv_array_enum;
                  CMC_array_extension         CMC_pv_array_extension;
                  CMC_array_float32           CMC_pv_array_float32;
                  CMC_array_float64           CMC_pv_array_float64;
                  CMC_array_guid              CMC_pv_array_guid;
                  CMC_array_iso_date_time     CMC_pv_array_iso_date_time;
                  CMC_array_object_handle     CMC_pv_array_object_handle;
                  CMC_array_opaque_data       CMC_pv_array_opaque_data;
                  CMC_array_return_code       CMC_pv_array_return_code;
                  CMC_array_sint16            CMC_pv_array_sint16;
                  CMC_array_sint32            CMC_pv_array_sint32;
                  CMC_array_string            CMC_pv_array_string;
                  CMC_array_time              CMC_pv_array_time;
                  CMC_array_uint16            CMC_pv_array_uint16;
                  CMC_array_uint32            CMC_pv_array_uint32;
      } value;
} CMC_property;

/*EVENT*/
typedef CMC_uint32              CMC_event;

/* EVENT TYPES */
#define CMC_EVENT_NEW_MESSAGES            ((CMC_enum) 0)

/*CALLBACK*/

typedef struct CMC_TAG_NEW_MESSAGE_CB_DATA {
      CMC_object_handle             *available;
} CMC_new_message_callback_data;

typedef struct CMC_TAG_NEW_MESSAGE_CHECK_DATA {
      CMC_uint32                    number_containers;
      CMC_object_handle             *containers;
} CMC_new_message_check_data;

typedef CMC_new_message_check_data  CMC_new_message_register_data;

typedef CMC_new_message_check_data  CMC_new_message_unregister_data;

typedef void (*CMC_callback) (
      CMC_session_id            session,
      CMC_event                 event,
      CMC_buffer                callback_data,
      CMC_buffer                register_data,
      CMC_extension             *callback_extensions
);
/*CURSOR RESTRICTION*/
```

```
typedef struct CMC_TAG_RESTRICTION_AND {
      CMC_uint32                             count;
      struct CMC_TAG_RESTRICTION_CURSOR . . . *restriction;
} CMC_restriction_and;

typedef struct CMC_TAG_RESTRICTION_OR {
      CMC_uint32                             count;
      struct CMC_TAG_RESTRICTION_CURSOR . . . *restriction;
} CMC_restriction_or;

typedef struct CMC_TAG_RESTRICTION_NOT {
      CMC_uint32                             count;
      struct CMC_TAG_RESTRICTION_CURSOR . . . *restriction;
} CMC_restriction_not;

typedef struct CMC_TAG_RESTRICTION_STRING {
      CMC_enum                               exactness;
      CMC_id                                 property;
      CMC_string                             string_constant;
} CMC_restriction_string;

typedef struct CMC_TAG_RESTRICTION_CONTENT {
      CMC_enum                               logical;
      CMC_id                                 property;
      CMC_buffer                             property_value;
} CMC_restriction_content;

typedef struct CMC_TAG_RESTRICTION_COMPARISON {
      CMC_enum                               logical;
      CMC_id                                 property1;
      CMC_id                                 property2;
} CMC_restriction_comparison;

typedef struct CMC_TAG_RESTRICTION_BITTEST {
      CMC_uint32                             comparison;
      CMC_id                                 property;
      CMC_uint32                             bittest;
} CMC_restriction_bittest;

typedef struct CMC_TAG_RESTRICTION_SIZE {
      CMC_enum                               logical;
      CMC_id                                 property;
      CMC_uint32                             byte_size;
} CMC_restriction_size;

typedef struct CMC_TAG_RESTRICTION_EXIST {
      CMC_id                                 property;
} CMC_restriction_exist;

typedef struct CMC_TAG_RESTRICTION_CURSOR {
      CMC_enum                               type;
      union {
                  CMC_restriction_and             restriction_and;
                  CMC_restriction_or              restriction_or;
                  CMC_restriction_not             restriction_not;
                  CMC_restriction_string          restriction_string;
                  CMC_restriction_content         restriction_content;
                  CMC_restriction_comparison      restriction_comparison;
                  CMC_restriction_bittest         restriction_bittest;
                  CMC_restriction_size            restriction_size;
                  CMC_restriction_exist           restriction_exist;
      } cr;
      CMC_extension                *property_extensions;
} CMC_cursor_restriction;

/* RESTRICTION TYPES AND CONSTANTS */
#define CMC_RESTRICTION_AND                    ((CMC_enum) 0)
#define CMC_RESTRICTION_OR                     ((CMC_enum) 1)
#define CMC_RESTRICTION_NOT                    ((CMC_enum) 2)
#define CMC_RESTRICTION_STRING                 ((CMC_enum) 3)
#define CMC_RESTRICTION_CONTENT                ((CMC_enum) 4)
#define CMC_RESTRICTION_COMPARISON             ((CMC_enum) 5)
#define CMC_RESTRICTION_BITTEST                ((CMC_enum) 6)
#define CMC_RESTRICTION_SIZE                   ((CMC_enum) 7)
#define CMC_RESTRICTION_EXIST                  ((CMC_enum) 8)
```

```
#define CMC_EXACTNESS_PRECISE                  ((CMC_enum) 0)
#define CMC_EXACTNESS_STARTS_WITH              ((CMC_enum) 1)
#define CMC_EXACTNESS_MIXED_CASE               ((CMC_enum) 2)

#define CMC_LOGICAL_LT                         ((CMC_enum) 0)
#define CMC_LOGICAL_LE                         ((CMC_enum) 1)
#define CMC_LOGICAL_EQ                         ((CMC_enum) 2)
#define CMC_LOGICAL_NE                         ((CMC_enum) 3)
#define CMC_LOGICAL_GT                         ((CMC_enum) 4)
#define CMC_LOGICAL_GE                         ((CMC_enum) 5)

#define CMC_COMPARISON_OR                      ((CMC_enum) 0)
#define CMC_COMPARISON_AND                     ((CMC_enum) 1)

/*CURSOR SORT KEY*/
typedef struct TAG_CURSOR_SORT_KEY{
      CMC_id                        property;
      CMC_enum                      order;
} CMC_cursor_sort_key;

/* CURSOR SORT KEY CONSTANTS */
#define CMC_SORT_DEFAULT                       ((CMC_enum) 0)
#define CMC_SORT_ASCEND                        ((CMC_enum) 1)
#define CMC_SORT_DESCEND                       ((CMC_enum) 2)

/*ATTACHMENT*/

typedef struct {
      CMC_string                    attach_title;
      CMC_object_identifier         attach_type;
      CMC_string                    attach_filename;
      CMC_flags                     attach_flags;
      CMC_extension                 *attach_extensions;
} CMC_attachment;

/* ATTACHMENT FLAGS */
#define CMC_ATT_APP_OWNS_FILE             ((CMC_flags) 1)
#define CMC_ATT_LAST_ELEMENT              ((CMC_flags) 0x80000000)

#define CMC_ATT_OID_BINARY                "1 2 840 113658 1 1"
#define CMC_ATT_OID_TEXT                  "1 2 840 113658 1 1 0"

/*MESSAGE REFERENCE*/
typedef CMC_counted_string      CMC_message_reference;

/*RECIPIENT*/
typedef struct {
      CMC_string                name;
      CMC_enum                  name_type;
      CMC_string                address;
      CMC_enum                  role;
      CMC_flags                 recip_flags;
      CMC_extension             *recip_extensions;
} CMC_recipient;

/* NAME TYPES */
#define CMC_TYPE_UNKNOWN                       ((CMC_enum) 0)
#define CMC_TYPE_INDIVIDUAL                    ((CMC_enum) 1)
#define CMC_TYPE_GROUP                         ((CMC_enum) 2)

/* ROLES */
#define CMC_ROLE_TO                            ((CMC_enum) 0)
#define CMC_ROLE_CC                            ((CMC_enum) 1)
#define CMC_ROLE_BCC                           ((CMC_enum) 2)
#define CMC_ROLE_ORIGINATOR                    ((CMC_enum) 3)
#define CMC_ROLE_AUTHORIZING_USER              ((CMC_enum) 4)
#define CMC_ROLE_REPLY_TO                      ((CMC_enum) 5)

/* RECIPIENT FLAGS */
#define CMC_RECIP_IGNORE                       ((CMC_flags) 1)
#define CMC_RECIP_LIST_TRUNCATED               ((CMC_flags) 2)
#define CMC_RECIP_LAST_ELEMENT                 ((CMC_flags) 0x80000000)
```

```
/*MESSAGE*/
typedef struct {
      CMC_message_reference         *message_reference;
      CMC_string                    message_type;
      CMC_string                    subject;
      CMC_time                      time_sent;
      CMC_string                    text_note;
      CMC_recipient                 *recipients;
      CMC_attachment                *attachments;
      CMC_flags                     message_flags;
      CMC_extension                 *message_extensions;
} CMC_message;

/* MESSAGE FLAGS */
#define CMC_MSG_READ                        ((CMC_flags) 1)
#define CMC_MSG_TEXT_NOTE_AS_FILE           ((CMC_flags) 2)
#define CMC_MSG_UNSENT                      ((CMC_flags) 4)
#define CMC_MSG_DELETE_AFTER_SEND           ((CMC_flags) 8)
#define CMC_MSG_LAST_ELEMENT                ((CMC_flags) 0x80000000)

/* MESSAGE TYPES */
#define CMC_MESSAGE_TYPE_IPM                     "CMC:IPM"
#define CMC_MESSAGE_TYPE_IP_RN                   "CMC:IP RN"
#define CMC_MESSAGE_TYPE_IP_NRN                  "CMC:IP NRN"
#define CMC_MESSAGE_TYPE_DR                      "CMC:DR"
#define CMC_MESSAGE_TYPE_NDR                     "CMC:NDR"
#define CMC_MESSAGE_TYPE_REPORT                  "CMC:REPORT"

/*MESSAGE SUMMARY*/
typedef struct {
      CMC_message_reference         *message_reference;
      CMC_string                    message_type;
      CMC_string                    subject;
      CMC_time                      time_sent;
      CMC_uint32                    byte_length;
      CMC_recipient                 *originator;
      CMC_flags                     summary_flags;
      CMC_extension                 *message_summary_extensions;
} CMC_message_summary;

/* MESSAGE SUMMARY FLAGS */
#define CMC_SUM_READ                        ((CMC_flags) 1)
#define CMC_SUM_UNSENT                      ((CMC_flags) 2)
#define CMC_SUM_HAS_ATTACHMENTS             ((CMC_flags) 4)
#define CMC_SUM_LAST_ELEMENT                ((CMC_flags) 0x80000000)

/*REPORT*/
typedef struct {
      CMC_recipient                 *msg_recipient;
      CMC_enum                      report_type;
      CMC_time                      delivered_time;
      CMC_uint32                    reason_code;
      CMC_flags                     report_flags;
} CMC_report;

/* REPORT FLAGS */
#define CMC_REPORT_LAST_ELEMENT             ((CMC_flags) 0x00000001)

/* REPORT TYPES */
#define CMC_X400_DR                         ((CMC_enum) 0)
#define CMC_X400_NDR                        ((CMC_enum) 1)

/*CMC FUNCTIONS*/
/*CROSS FUNCTION FLAGS*/
#define CMC_ERROR_UI_ALLOWED                ((CMC_flags) 0x01000000)
#define CMC_LOGON_UI_ALLOWED                ((CMC_flags) 0x02000000)
#define CMC_COUNTED_STRING_TYPE             ((CMC_flags) 0x04000000)

/*OBJECT CLASSES*/
#define CMC_TYPE_OC_ADDRESS_BOOK                 ((CMC_enum) 1)
#define CMC_TYPE_OC_CONTENT_ITEM                 ((CMC_enum) 2)
#define CMC_TYPE_OC_MESSAGE                      ((CMC_enum) 3)
#define CMC_TYPE_OC_MESSAGE_CONTAINER            ((CMC_enum) 4)
```

```
#define CMC_TYPE_OC_DISTRIBUTION_LIST          ((CMC_enum) 5)
#define CMC_TYPE_OC_RECIPIENT                  ((CMC_enum) 6)
#define CMC_TYPE_OC_REPORT                     ((CMC_enum) 7)
#define CMC_TYPE_OC_ROOT_CONTAINER             ((CMC_enum) 8)
#define CMC_TYPE_OC_PER_RECIPIENT_INFORMATION  ((CMC_enum) 9)
#define CMC_TYPE_OC_PROFILE_CONTAINER          ((CMC_enum) 10)

#define CMC_OC_MESSAGE  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Message//EN"

#define CMC_OC_CONTENT_ITEM  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Content Item//EN"

#define CMC_OC_RECIPIENT  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Recipient//EN"

#define CMC_OC_REPORT  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Report//EN"

#define CMC_OC_MESSAGE_CONTAINER  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Message Container//EN"

#define CMC_OC_ADDRESS_BOOK  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Address Book//EN"

#define CMC_OC_DISTRIBUTION_LIST  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Distribution List//EN"

#define CMC_OC_ROOT_CONTAINER  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Root Container//EN"

#define CMC_OC_PER_RECIPIENT_INFORMATION  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Per Recipient Information//EN"

#define CMC_OC_PROFILE_CONTAINER  \
       "-//XAPIA/CMC/OBJECT CLASS//NONSGML Profile Container//EN"

/*OBJECT PROPERTIES*/

/* Object Class. Applies to all objects. */

#define CMC_PT_OBJECT_CLASS \
       "-//XAPIA/CMC/PROPERTY//NONSGML Object Class//EN"

/* Address Book */

#define CMC_PT_ADDRESS_BOOK_CHILD_ALLOWED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Child Allowed//EN"

#define CMC_PT_ADDRESS_BOOK_COMMENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Comment//EN"

#define CMC_PT_ADDRESS_BOOK_LOCATION  \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Location//EN"

#define CMC_PT_ADDRESS_BOOK_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Name//EN"

#define CMC_PT_ADDRESS_BOOK_PARENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Parent//EN"

#define CMC_PT_ADDRESS_BOOK_SERVER_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Server Name//EN"

#define CMC_PT_ADDRESS_BOOK_SHARED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Shared//EN"

#define CMC_PT_ADDRESS_BOOK_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Address Book Type//EN"

/* Content Type */

#define CMC_PT_CONTENT_ITEM_CHARACTER_SET \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Character Set//EN"

#define CMC_PT_CONTENT_ITEM_CONTENT_INFORMATION \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Content Information//EN"

#define CMC_PT_CONTENT_ITEM_CREATE_TIME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Create Time//EN"
```

```
#define CMC_PT_CONTENT_ITEM_ENCODING_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Encoding Type//EN"

#define CMC_PT_CONTENT_ITEM_FILE_DIRECTORY \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item File Directory//EN"

#define CMC_PT_CONTENT_ITEM_FILE_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item File Name//EN"

#define CMC_PT_CONTENT_ITEM_LAST_MODIFIED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Last Modified//EN"

#define CMC_PT_CONTENT_ITEM_RENDER_POSITION \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Render Position//EN"

#define CMC_PT_CONTENT_ITEM_SIZE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Size//EN"

#define CMC_PT_CONTENT_ITEM_TITLE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Title//EN"

#define CMC_PT_CONTENT_ITEM_CONTENT_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Content Type//EN"

#define CMC_PT_CONTENT_ITEM_ITEM_NUMBER \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Number//EN"

#define CMC_PT_CONTENT_ITEM_ITEM_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Content Item Item Type//EN"

/* Distribution List */

#define CMC_PT_DISTRIBUTION_LIST_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Distribution List Name//EN"

#define CMC_PT_DISTRIBUTION_LIST_ADDRESS \
       "-//XAPIA/CMC/PROPERTY//NONSGML Distribution List Address//EN"

#define CMC_PT_DISTRIBUTION_LIST_COMMENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Distribution List Comment//EN"

#define CMC_PT_DISTRIBUTION_LIST_LAST_MODIFICATION_TIME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Distribution List Last Modification
        Time//EN"

#define CMC_PT_DISTRIBUTION_LIST_PARENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Distribution List Parent//EN"

#define CMC_PT_DISTRIBUTION_LIST_SHARED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Distribution List Shared//EN"

/* Message */

#define CMC_PT_MESSAGE_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Type//EN"

#define CMC_PT_MESSAGE_PRIORITY \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Priority//EN"

#define CMC_PT_MESSAGE_SIZE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Size//EN"

#define CMC_PT_MESSAGE_SUBJECT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Subject//EN"

#define CMC_PT_MESSAGE_APPLICATION_ID \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Application Id//EN"

#define CMC_PT_MESSAGE_TIME_RECEIVED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Time Received//EN"

#define CMC_PT_MESSAGE_TIME_SENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Time Sent//EN"

#define CMC_PT_MESSAGE_DEFERRED_DELIVERY_TIME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Deferred Delivery Time//EN"

#define CMC_PT_MESSAGE_IN_REPLY_TO \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message In Reply To//EN"
```

```
#define CMC_PT_MESSAGE_ID \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Id//EN"

#define CMC_PT_MESSAGE_RECEIPT_REQUESTED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Receipt Requested//EN"

#define CMC_PT_MESSAGE_SENSITIVITY \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Sensitivity//EN"

#define CMC_PT_MESSAGE_ITEM_COUNT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Item Count//EN"

#define CMC_PT_MESSAGE_NRN_DIAGNOSTIC   \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message NRN Diagnostic//EN"

#define CMC_PT_MESSAGE_NRN_REASON \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message NRN Reason//EN"

#define CMC_PT_MESSAGE_RECEIPT_TYPE  \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Receipt Type//EN"

#define CMC_PT_MESSAGE_REPORT_REQUESTED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Report Requested//EN"

#define CMC_PT_MESSAGE_ROLE    \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Role//EN"

#define CMC_PT_MESSAGE_AUTO_ACTION  \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Auto Action//EN"

#define CMC_PT_CLIENT_MSG_STATUS                 \
       "-//XAPIA/CMC/PROPERTY//NONSGML Client Msg Status//EN"

#define CMC_PT_OUT_MSG_STATUS              \
       "-//XAPIA/CMC/PROPERTY//NONSGML Out Msg Status//EN"

#define CMC_PT_APPLICATION_MSG_STATUS                \
       "-//XAPIA/CMC/PROPERTY//NONSGML Application Msg Status//EN"

/* Message Container */

#define CMC_PT_MESSAGE_CONTAINER_CHILD_ALLOWED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Child Allowed//EN"

#define CMC_PT_MESSAGE_CONTAINER_COMMENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Comment//EN"

#define CMC_PT_MESSAGE_CONTAINER_LOCATION \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Location//EN"

#define CMC_PT_MESSAGE_CONTAINER_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Name//EN"

#define CMC_PT_MESSAGE_CONTAINER_PARENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Parent//EN"

#define CMC_PT_MESSAGE_CONTAINER_SERVER_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Server Name//EN"

#define CMC_PT_MESSAGE_CONTAINER_SHARED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Shared//EN"

#define CMC_PT_MESSAGE_CONTAINER_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Message Container Type//EN"

/* Recipient */

#define CMC_PT_RECIPIENT_ADDRESS \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Address//EN"

#define CMC_PROP_TYPE_RECIPIENT_CONTENT_RETURN_REQUESTED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Content Return Requested//EN"

#define CMC_PT_RECIPIENT_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Name//EN"

#define CMC_PT_RECIPIENT_RECEIPT_REQUESTED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Receipt Requested//EN"

#define CMC_PT_RECIPIENT_REPORT_REQUESTED   \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Report Requested//EN"
```

```
#define CMC_PT_RECIPIENT_ROLE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Role//EN"

#define CMC_PT_RECIPIENT_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Type//EN"

#define CMC_PT_RECIPIENT_RESPONSIBILITY_FLAG \
       "-//XAPIA/CMC/PROPERTY//NONSGML Recipient Responsibility Flag//EN"

/* Report */

#define CMC_PT_REPORT_READ \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Read//EN"

#define CMC_PT_REPORT_UNSENT  \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Unsent//EN"

#define CMC_PT_REPORT_SIZE \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Size//EN"

#define CMC_PT_REPORT_SUBJECT  \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Subject//EN"

#define CMC_PT_REPORT_TIME_RECEIVED   \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Time Received//EN"

#define CMC_PT_REPORT_TIME_SENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Time Sent//EN"

#define CMC_PT_REPORT_APPLICATION_ID \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Application Id//EN"

#define CMC_PT_REPORT_SUBJECT_MESSAGE_ID     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Subject Message Id//EN"

#define CMC_PT_REPORT_ITEM_COUNT   \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Item Count//EN"

#define CMC_PT_REPORT_ID     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Id//EN"

#define CMC_PT_REPORT_MESSAGING_SYSTEM_ID   \
       "-//XAPIA/CMC/PROPERTY//NONSGML Report Messaging System Id//EN"

/* Root Container */

#define CMC_PT_ROOT_CONTAINER_CHILD_ALLOWED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Root Container Child Allowed//EN"

#define CMC_PT_ROOT_CONTAINER_COMMENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML Root Container Comment//EN"

#define CMC_PT_ROOT_CONTAINER_LOCATION \
       "-//XAPIA/CMC/PROPERTY//NONSGML Root Container Location//EN"

#define CMC_PT_ROOT_CONTAINER_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML Root Container Name//EN"

#define CMC_PT_ROOT_CONTAINER_SHARED \
       "-//XAPIA/CMC/PROPERTY//NONSGML Root Container Shared//EN"

/* Per Recipient Information */

#define CMC_PT_PRI_TYPE \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Type//EN"

#define CMC_PT_PRI_DELIVERY_TIME   \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Delivery Time//EN"

#define CMC_PT_PRI_REASON   \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Reason//EN"

#define CMC_PT_PRI_DIAGNOSTIC   \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Diagnostic//EN"

#define CMC_PT_PRI_RECIPIENT_NAME \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Recipient Name//EN"

#define CMC_PT_PRI_RECIPIENT_ADDRESS \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Recipient Address//EN"
```

```
#define CMC_PT_PRI_COMMENT \
       "-//XAPIA/CMC/PROPERTY//NONSGML PRI Comment//EN"

/* Profile Container */

#define CMC_PT_PROFILE_CHARACTER_SET    \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Character Set//EN"

#define CMC_PT_PROFILE_LINE_TERM        \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Line Term//EN"

#define CMC_PT_PROFILE_DEFAULT_SERVICE     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Default Service//EN"

#define CMC_PT_PROFILE_DEFAULT_USER      \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Default User//EN"

#define CMC_PT_PROFILE_REQ_PASSWORD      \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Req Password//EN"

#define CMC_PT_PROFILE_REQ_SERVICE        \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Req Service//EN"

#define CMC_PT_PROFILE_REQ_USER     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Req User//EN"

#define CMC_PT_PROFILE_UI_AVAIL     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile UI Avail//EN"

#define CMC_PT_PROFILE_SUP_NOMKMSGREAD     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Sup NoMkMsgRead//EN"

#define CMC_PT_PROFILE_SUP_COUNTED_STR     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Sup Counted Str//EN"

#define CMC_PT_PROFILE_VER_IMPLEM      \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Ver Implem//EN"

#define CMC_PT_PROFILE_VER_SPEC     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Ver Spec//EN"

#define CMC_PT_PROFILE_USERS     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Users//EN"

#define CMC_PT_PROFILE_OBJECT_SUP      \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Object Sup//EN"

#define CMC_PT_PROFILE_PROP_SUP   \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Prop Sup//EN"

#define CMC_PT_PROFILE_CONF       \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Conf//EN"

#define CMC_PT_PROFILE_OBJECT_EXT      \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Object Ext//EN"

#define CMC_PT_PROFILE_PROP_EXT     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Prop Ext//EN"

#define CMC_PT_PROFILE_AUTO_ACTION     \
       "-//XAPIA/CMC/PROPERTY//NONSGML Profile Auto Action//EN"

/* Property Value Constants. CMC_id values. */

/* Object Class. Applies to all objects. */

#define CMC_PV_OBJECT_CLASS                              0

/* Address Book */
#define CMC_PV_ADDRESS_BOOK_CHILD_ALLOWED                1
#define CMC_PV_ADDRESS_BOOK_COMMENT                      2
#define CMC_PV_ADDRESS_BOOK_LOCATION                     3
#define CMC_PV_ADDRESS_BOOK_NAME                         4
#define CMC_PV_ADDRESS_BOOK_PARENT                       5
#define CMC_PV_ADDRESS_BOOK_SERVER_NAME                  6
#define CMC_PV_ADDRESS_BOOK_SHARED                       7
#define CMC_PV_ADDRESS_BOOK_TYPE                         8
```

```
/* Content Item */
#define CMC_PV_CONTENT_ITEM_CHARACTER_SET              9
#define CMC_PV_CONTENT_ITEM_CONTENT_INFORMATION        10
#define CMC_PV_CONTENT_ITEM_CREATE_TIME                11
#define CMC_PV_CONTENT_ITEM_ENCODING_TYPE              12
#define CMC_PV_CONTENT_ITEM_FILE_DIRECTORY             13
#define CMC_PV_CONTENT_ITEM_FILE_NAME                  14
#define CMC_PV_CONTENT_ITEM_LAST_MODIFIED              15
#define CMC_PV_CONTENT_ITEM_RENDER_POSITION            16
#define CMC_PV_CONTENT_ITEM_SIZE                       17
#define CMC_PV_CONTENT_ITEM_TITLE                      18
#define CMC_PV_CONTENT_ITEM_CONTENT_TYPE               19
#define CMC_PV_CONTENT_ITEM_ITEM_NUMBER                20
#define CMC_PV_CONTENT_ITEM_ITEM_TYPE                  21
/* Distribution List */
#define CMC_PV_DISTRIBUTION_LIST_NAME                  22
#define CMC_PV_DISTRIBUTION_LIST_ADDRESS               23
#define CMC_PV_DISTRIBUTION_LIST_COMMENT               24
#define CMC_PV_DISTRIBUTION_LIST_LAST_MODIFICATION_TIME  25
#define CMC_PV_DISTRIBUTION_LIST_PARENT                26
#define CMC_PV_DISTRIBUTION_LIST_SHARED                27
/* Message */
#define CMC_PV_MESSAGE_TYPE                            28
#define CMC_PV_MESSAGE_PRIORITY                        29
#define CMC_PV_MESSAGE_SIZE                            30
#define CMC_PV_MESSAGE_SUBJECT                         31
#define CMC_PV_MESSAGE_APPLICATION_ID                  32
#define CMC_PV_MESSAGE_TIME_RECEIVED                   33
#define CMC_PV_MESSAGE_TIME_SENT                       34
#define CMC_PV_MESSAGE_DEFERRED_DELIVERY_TIME          35
#define CMC_PV_MESSAGE_IN_REPLY_TO                     36
#define CMC_PV_MESSAGE_ID                              37
#define CMC_PV_MESSAGE_RECEIPT_REQUESTED               38
#define CMC_PV_MESSAGE_SENSITIVITY                     39
#define CMC_PV_MESSAGE_ITEM_COUNT                      40
#define CMC_PV_MESSAGE_NRN_DIAGNOSTIC                  41
#define CMC_PV_MESSAGE_NRN_REASON                      42
#define CMC_PV_MESSAGE_REPORT_REQUESTED                43
#define CMC_PV_MESSAGE_ROLE                            44
#define CMC_PV_MESSAGE_AUTO_ACTION                     45
#define CMC_PV_MESSAGE_CLIENT_MSG_STATUS               46
#define CMC_PV_MESSAGE_OUT_MSG_STATUS                  47
#define CMC_PV_MESSAGE_APPLICATION_MSG_STATUS          48
/* Message Container */
#define CMC_PV_MESSAGE_CONTAINER_CHILD_ALLOWED         49
#define CMC_PV_MESSAGE_CONTAINER_COMMENT               50
#define CMC_PV_MESSAGE_CONTAINER_LOCATION              51
#define CMC_PV_MESSAGE_CONTAINER_NAME                  52
#define CMC_PV_MESSAGE_CONTAINER_PARENT                53
#define CMC_PV_MESSAGE_CONTAINER_SERVER_NAME           54
#define CMC_PV_MESSAGE_CONTAINER_SHARED                55
#define CMC_PV_MESSAGE_CONTAINER_TYPE                  56
/* Recipient */
#define CMC_PV_RECIPIENT_ADDRESS                       57
#define CMC_PV_RECIPIENT_CONTENT_RETURN_REQUESTED      58
#define CMC_PV_RECIPIENT_NAME                          59
#define CMC_PV_RECIPIENT_RECEIPT_REQUESTED             60
#define CMC_PV_RECIPIENT_REPORT_REQUESTED              61
#define CMC_PV_RECIPIENT_ROLE                          62
#define CMC_PV_RECIPIENT_TYPE                          63
#define CMC_PV_RECIPIENT_RESPONSIBILITY_FLAG           64
/* Report */
#define CMC_PV_REPORT_READ                             65
#define CMC_PV_REPORT_UNSENT                           66
#define CMC_PV_REPORT_SIZE                             67
#define CMC_PV_REPORT_SUBJECT                          68
#define CMC_PV_REPORT_TIME_RECEIVED                    69
#define CMC_PV_REPORT_TIME_SENT                        70
#define CMC_PV_REPORT_APPLICATION_ID                   71
#define CMC_PV_REPORT_SUBJECT_MESSAGE_ID               72
#define CMC_PV_REPORT_ITEM_COUNT                       73
#define CMC_PV_REPORT_ID                               74
#define CMC_PV_REPORT_MESSAGING_SYSTEM_ID              75
```

```
/* Root Container */
#define CMC_PV_ROOT_CONTAINER_CHILD_ALLOWED             76
#define CMC_PV_ROOT_CONTAINER_COMMENT                   77
#define CMC_PV_ROOT_CONTAINER_LOCATION                  78
#define CMC_PV_ROOT_CONTAINER_NAME                      79
#define CMC_PV_ROOT_CONTAINER_SHARED                    80
/* Per Recipient Information */
#define CMC_PV_PRI_TYPE                                 81
#define CMC_PV_PRI_DELIVERY_TIME                        82
#define CMC_PV_PRI_REASON                               83
#define CMC_PV_PRI_DIAGNOSTIC                           84
#define CMC_PV_PRI_RECIPIENT_NAME                       85
#define CMC_PV_PRI_RECIPIENT_ADDRESS                    86
#define CMC_PV_PRI_COMMENT                              87
/* Profile */
#define CMC_PV_PROFILE_CHARACTER_SET                    88
#define CMC_PV_PROFILE_LINE_TERM                        89
#define CMC_PV_PROFILE_DEFAULT_SERVICE                  90
#define CMC_PV_PROFILE_DEFAULT_USER                     91
#define CMC_PV_PROFILE_REQ_PASSWORD                     92
#define CMC_PV_PROFILE_REQ_SERVICE                      93
#define CMC_PV_PROFILE_REQ_USER                         94
#define CMC_PV_PROFILE_UI_AVAIL                         95
#define CMC_PV_PROFILE_SUP_NOMKMSGREAD                  96
#define CMC_PV_PROFILE_SUP_COUNTED_STR                  97
#define CMC_PV_PROFILE_VER_IMPLEM                       98
#define CMC_PV_PROFILE_VER_SPEC                         99
#define CMC_PV_PROFILE_USERS                            100
#define CMC_PV_PROFILE_OBJECT_SUP                       101
#define CMC_PV_PROFILE_PROP_SUP                         102
#define CMC_PV_PROFILE_CONF                             103
#define CMC_PV_PROFILE_OBJECT_EXT                       104
#define CMC_PV_PROFILE_PROP_EXT                         105
#define CMC_PV_PROFILE_AUTO_ACTION                      106

/*OBJECT PROPERTY CONSTANT VALUES*/

/* Address Book */
#define CMC_ADDRESS_BOOK_LOCATION_LOCAL        ((CMC_enum) 0)
#define CMC_ADDRESS_BOOK_LOCATION_SERVER       ((CMC_enum) 1)
#define CMC_ADDRESS_BOOK_LOCATION_UNKNOWN      ((CMC_enum) 2)

#define CMC_ADDRESS_BOOK_TYPE_GLOBAL           ((CMC_enum) 0)
#define CMC_ADDRESS_BOOK_TYPE_PERSONAL         ((CMC_enum) 1)

/* Content Information */
#define CMC_CHARSET_437          "-//XAPIA/CHARSET//NONSGML IBM 437//EN"
#define CMC_CHARSET_850          "-//XAPIA/CHARSET//NONSGML IBM 850//EN"
#define CMC_CHARSET_1252         "-//XAPIA/CHARSET//NONSGML Microsoft 1252//EN"
#define CMC_CHARSET_ISTRING      "-//XAPIA/CHARSET//NONSGML Apple ISTRING//EN"
#define CMC_CHARSET_UNICODE      "-//XAPIA/CHARSET//NONSGML UNICODE//EN"
#define CMC_CHARSET_T61          "-//XAPIA/CHARSET//NONSGML TSS T61//EN"
#define CMC_CHARSET_IA5          "-//XAPIA/CHARSET//NONSGML TSS IA5//EN"
#define CMC_CHARSET_ISO_10646    "-//XAPIA/CHARSET//NONSGML ISO 10646//EN"
#define CMC_CHARSET_ISO_646      "-//XAPIA/CHARSET//NONSGML ISO 646//EN"
#define CMC_CHARSET_ISO_8859_1   "-//XAPIA/CHARSET//NONSGML ISO 8859-1//EN"

/* Encoding Type */
#define CMC_ET_7_BIT                                                 \
       "-//XAPIA/CMC/ENCODING TYPE//NONSGML 7 Bit//EN"
#define CMC_ET_BASE64                                                \
       "-//XAPIA/CMC/ENCODING TYPE//NONSGML Base64//EN"
#define CMC_ET_BINARY                                                \
       "-//XAPIA/CMC/ENCODING TYPE//NONSGML Binary//EN"
#define CMC_ET_8_BIT                                                 \
       "-//XAPIA/CMC/ENCODING TYPE//NONSGML 8 Bit//EN"
#define CMC_ET_QUOTED_PRINTABLE                                      \
       "-//XAPIA/CMC/ENCODING TYPE//NONSGML Quoted Printable//EN"

/* Content Type */
#define CMC_CT_PLAIN_TEXT                                            \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Plain Text//EN"
#define CMC_CT_GIF_IMAGE                                             \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML GIF Image//EN"
```

```
#define CMC_CT_JPEG_IMAGE                                               \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML JPEG Image//EN"
#define CMC_CT_BASIC_AUDIO                                              \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Basic Audio//EN"
#define CMC_CT_MPEG_VIDEO                                               \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML MPEG Video//EN"
#define CMC_CT_MESSAGE                                                  \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Message//EN"
#define CMC_CT_PARTIAL_MESSAGE                                          \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Partial Message//EN"
#define CMC_CT_EXTERNAL_MESSAGE                                         \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML External Message//EN"
#define CMC_CT_APPLICATION_OCTET_STREAM             \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Application Octet Stream//EN"
#define CMC_CT_APPLICATION_POSTSCRIPT               \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Application PostScript//EN"
#define CMC_CT_ALTERNATIVE_MULTIPART                \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Alternative Multipart//EN"
#define CMC_CT_DIGEST_MULTIPART                                         \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Digest Multipart//EN"
#define CMC_CT_MIXED_MULTIPART                                          \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Mixed Multipart//EN"
#define CMC_CT_OLE                        \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML OLE//EN"
#define CMC_CT_MIXED_MULTIPART                                          \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML Mixed Multipart//EN"
#define CMC_CT_X400_G3_FAX\
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 G3 Fax//EN"
#define CMC_CT_X400_G4_FAX\
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 G4 Fax//EN"
#define CMC_CT_X400_ENCRYPTED \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Encrypted//EN"
#define CMC_CT_X400_NATIONALLY_DEFINED  \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Nationally Defined//EN"
#define CMC_CT_X400_FILE_TRANSFER    \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 File Transfer//EN"
#define CMC_CT_X400_VOICE        \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Voice//EN"
#define CMC_CT_X400_VIDEOTEX     \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Videotex//EN"
#define CMC_CT_X400_MIXED_MODE      \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Mixed Mode//EN"
#define CMC_CT_X400_PRIVATELY_DEFINED_6937        \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Privately Defined 6937//EN"
#define CMC_CT_X400_EXTERNAL_TRACE      \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 External Trace//EN"
#define CMC_CT_X400_INTERNAL_TRACE      \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML X400 Internal Trace//EN"
#define CMC_CT_SMTP_SESSION_TRANSCRIPT  \
       "-//XAPIA/CMC/CONTENT TYPE//NONSGML SMTP Session Transcript//EN"

/* Content Item Type */
#define CMC_IT_NOTE                             ((CMC_enum) 0)
#define CMC_IT_ATTACHMENT                       ((CMC_enum) 1)
#define CMC_IT_ANNOTATION                       ((CMC_enum) 2)

/* Message Types and Message Constants */
#define CMC_MT_IPM                              ((CMC_enum) 0)
#define CMC_MT_RECEIPT                          ((CMC_enum) 1)
#define CMC_MT_EDI                              ((CMC_enum) 2)
#define CMC_MT_DIRECTOR                         ((CMC_enum) 3)
#define CMC_MT_DOCMGMT                          ((CMC_enum) 4)
#define CMC_MT_WORKFLOW                         ((CMC_enum) 5)
#define CMC_MT_CALSCHED                         ((CMC_enum) 6)

#define CMC_PRIORITY_NORMAL                     ((CMC_enum) 0)
#define CMC_PRIORITY_URGENT                     ((CMC_enum) 1)
#define CMC_PRIORITY_LOW                        ((CMC_enum) 2)

#define CMC_MESSAGE_SENSITIVITY_PERSONAL        ((CMC_enum) 0)
#define CMC_MESSAGE_SENSITIVITY_PRIVATE         ((CMC_enum) 1)
#define CMC_MESSAGE_SENSITIVITY_CONFIDENTIAL    ((CMC_enum) 2)
#define CMC_MESSAGE_SENSITIVITY_NONE            ((CMC_enum) 3)
```

```
#define CMC_RECEIPT_RN                           ((CMC_enum) 0)
#define CMC_RECEIPT_NRN                          ((CMC_enum) 1)
#define CMC_RECEIPT_BOTH                         ((CMC_enum) 2)
#define CMC_RECEIPT_NONE                         ((CMC_enum) 3)

#define CMC_REPORT_DR                            ((CMC_enum) 0)
#define CMC_REPORT_NDR                           ((CMC_enum) 1)
#define CMC_REPORT_BOTH                          ((CMC_enum) 2)
#define CMC_REPORT_NONE                          ((CMC_enum) 3)

#define CMC_MESSAGE_ROLE_ORIGINAL                ((CMC_enum) 0)
#define CMC_MESSAGE_ROLE_RETURNED                ((CMC_enum) 1)
#define CMC_MESSAGE_ROLE_FORWARDED               ((CMC_enum) 2)
#define CMC_MESSAGE_ROLE_REPLIED                 ((CMC_enum) 3)
#define CMC_MESSAGE_ROLE_OBSOLETED               ((CMC_enum) 4)
#define CMC_MESSAGE_ROLE_RESENT                  ((CMC_enum) 5)

#define CMC_AA_DELETE                            ((CMC_flags) 1)

/*Client Message Status*/
#define CMC_MESSAGE_STATUS_DRAFT                 ((CMC_enum) 0)

/*Out Message Status*/
#define CMC_MESSAGE_STATUS_DELETED               ((CMC_enum) 0)
#define CMC_MESSAGE_STATUS_SUBMITTED             ((CMC_enum) 1)
#define CMC_MESSAGE_STATUS_SENT                  ((CMC_enum) 2)

/*In Message Status*/
#define CMC_MESSAGE_STATUS_NEW                   ((CMC_enum) 0)
#define CMC_MESSAGE_STATUS_READ                  ((CMC_enum) 1)
#define CMC_MESSAGE_STATUS_CHANGED               ((CMC_enum) 2)

/* Message Container Types and Constants */

#define CMC_MESSAGE_CONTAINER_LOCATION_LOCAL     ((CMC_enum) 0)
#define CMC_MESSAGE_CONTAINER_LOCATION_SERVER    ((CMC_enum) 1)
#define CMC_MESSAGE_CONTAINER_LOCATION_UNKNOWN   ((CMC_enum) 2)

#define CMC_MCT_INBOX                            ((CMC_enum) 0)
#define CMC_MCT_OUTBOX                           ((CMC_enum) 1)
#define CMC_MCT_DRAFTS                           ((CMC_enum) 2)
#define CMC_MCT_DELETED                          ((CMC_enum) 3)
#define CMC_MCT_FILED                            ((CMC_enum) 4)
#define CMC_MCT_SENT                             ((CMC_enum) 5)

/* Recipient */
#define CMC_RECIPIENT_ROLE_TO                    ((CMC_enum) 0)
#define CMC_RECIPIENT_ROLE_CC                    ((CMC_enum) 1)
#define CMC_RECIPIENT_ROLE_BCC                   ((CMC_enum) 2)
#define CMC_RECIPIENT_ROLE_ORIGINATOR            ((CMC_enum) 3)
#define CMC_RECIPIENT_ROLE_AUTHORIZING_USER      ((CMC_enum) 4)
#define CMC_RECIPIENT_ROLE_IN_REPLY_TO           ((CMC_enum) 5)

#define CMC_RCT_UNKNOWN                          ((CMC_enum) 0)
#define CMC_RCT_INDIVIDUAL                       ((CMC_enum) 1)
#define CMC_RCT_GROUP                            ((CMC_enum) 2)
#define CMC_RCT_REPORT_RECIPIENT                 ((CMC_enum) 3)

/* Report */
#define CMC_RPT_RECEIPT_NOTICE                   ((CMC_enum) 0)
#define CMC_RPT_NONRECEIPT_NOTICE                ((CMC_enum) 1)
#define CMC_RPT_DELIVERY_NOTICE                  ((CMC_enum) 2)
#define CMC_RPT_NONDELIVERY_NOTICE               ((CMC_enum) 3)

/* Root Container */
#define CMC_ROOT_CONTAINER_LOCATION_LOCAL        ((CMC_enum) 0)
#define CMC_ROOT_CONTAINER_LOCATION_SERVER       ((CMC_enum) 1)
#define CMC_ROOT_CONTAINER_LOCATION_UNKNOWN      ((CMC_enum) 2)

/* Per Recipient Information */
#define CMC_PRI_DR                               ((CMC_enum) 0)
#define CMC_PRI_NDR                              ((CMC_enum) 1)
#define CMC_PRI_UNKNOWN                          ((CMC_enum) 2)

/* Profile */
#define CMC_CONF_SIMPLE_CMC                      ((CMC_enum) 0)
#define CMC_CONF_FULL_CMC                        ((CMC_enum) 1)
```

```
/* EXTENSION FLAGS */
#define CMC_EXT_REQUIRED                       ((CMC_flags) 0x00010000)
#define CMC_EXT_OUTPUT                         ((CMC_flags) 0x00020000)
#define CMC_EXT_LAST_ELEMENT                   ((CMC_flags) 0x80000000)
#define CMC_EXT_RSV_FLAG_MASK                  ((CMC_flags) 0xFFFF0000)
#define CMC_EXT_ITEM_FLAG_MASK                 ((CMC_flags) 0x0000FFFF)

#ifndef CMC_WCHAR

/* SEND */
CMC_return_code
cmc_send(
      CMC_session_id            session,
      CMC_message               *message,
      CMC_flags                 send_flags,
      CMC_ui_id                 ui_id,
      CMC_extension             *send_extensions
);

/* SEND DOCUMENT */
CMC_return_code
cmc_send_documents(
      CMC_string                recipient_addresses,
      CMC_string                subject,
      CMC_string                text_note,
      CMC_flags                 send_doc_flags,
      CMC_string                file_paths,
      CMC_string                file_names,
      CMC_string                delimiter,
      CMC_ui_id                 ui_id
);

/* ACT ON */
CMC_return_code
cmc_act_on(
      CMC_session_id            session,
      CMC_message_reference     *message_reference,
      CMC_enum                  operation,
      CMC_flags                 act_on_flags,
      CMC_ui_id                 ui_id,
      CMC_extension             *act_on_extensions
);

/* LIST */
CMC_return_code
cmc_list(
      CMC_session_id            session,
      CMC_string                message_type,
      CMC_flags                 list_flags,
      CMC_message_reference     *seed,
      CMC_uint32                *count,
      CMC_ui_id                 ui_id,
      CMC_message_summary       **result,
      CMC_extension             *list_extensions
);

/* READ */
CMC_return_code
cmc_read(
      CMC_session_id            session,
      CMC_message_reference     *message_reference,
      CMC_flags                 read_flags,
      CMC_message               **message,
      CMC_ui_id                 ui_id,
      CMC_extension             *read_extensions
);
```

```
/* LOOK UP */
CMC_return_code
cmc_look_up(
      CMC_session_id            session,
      CMC_recipient             *recipient_in,
      CMC_flags                 look_up_flags,
      CMC_ui_id                 ui_id,
      CMC_uint32                *count,
      CMC_recipient             **recipient_out,
      CMC_extension             *look_up_extensions
);

/* FREE */
CMC_return_code
cmc_free(
      CMC_buffer                memory
);

/* LOGOFF */
CMC_return_code
cmc_logoff(
      CMC_session_id            session,
      CMC_ui_id                 ui_id,
      CMC_flags                 logoff_flags,
      CMC_extension             *logoff_extensions
);

/* LOGON */
CMC_return_code
cmc_logon(
      CMC_string                service,
      CMC_string                user,
      CMC_string                password,
      CMC_object_identifier     character_set,
      CMC_ui_id                 ui_id,
      CMC_uint16                caller_cmc_version,
      CMC_flags                 logon_flags,
      CMC_session_id            *session,
      CMC_extension             *logon_extensions
);

/* QUERY CONFIGURATION */
CMC_return_code
cmc_query_configuration(
      CMC_session_id            session,
      CMC_enum                  item,
      CMC_buffer                reference,
      CMC_extension             *config_extensions
);

/* FULL CMC */

/* COPY OBJECT */
CMC_return_code
cmc_copy_object(
      CMC_object_handle         container,
      CMC_object_handle         source_object,
      CMC_object_handle         *new_object,
      CMC_extension             *copy_object_extensions
);

/* ADD PROPERTIES */
CMC_return_code
cmc_add_properties(
      CMC_object_handle         object,
      CMC_uint32                number_properties,
      CMC_property              *properties,
      CMC_extension             *add_properties_extensions
);
```

```
/* COMMIT OBJECT */
CMC_return_code
cmc_commit_object(
      CMC_object_handle         source_object,
      CMC_extension             *commit_object_extensions
);

/* COPY OBJECT HANDLE */
CMC_return_code
cmc_copy_object_handle(
      CMC_object_handle         source_handle,
      CMC_object_handle         *new_handle,
      CMC_extension             *copy_object_handle_extensions
);

/* CREATE DERIVED MESSAGE OBJECT */
CMC_return_code
cmc_create_derived_message_object(
      CMC_object_handle         original_message,
      CMC_enum                  derived_action,
      CMC_boolean               inherit_contents,
      CMC_object_handle         *derived_message,
      CMC_boolean               modified_message,
      CMC_extension             *create_derived_object_extensions
);

/* DELETE OBJECTS */
CMC_return_code
cmc_delete_objects(
      CMC_uint32                number_objects,
      CMC_object_handle         *object,
      CMC_extension             *delete_objects_extensions
);

/* DELETE PROPERTIES */
CMC_return_code
cmc_delete_properties(
      CMC_object_handle         object,
      CMC_uint32                number_properties,
      CMC_id                    *property_ids,
      CMC_extension             *delete_properties_extensions
);

/* GET ROOT HANDLE */
CMC_return_code
cmc_get_root_handle(
      CMC_session_id            session,
      CMC_object_handle         *root_object_handle,
      CMC_extension             *get_root_handle_extensions
);

/* IDENTIFIER TO NAME */
CMC_return_code
cmc_identifier_to_name(
      CMC_id                    identifier,
      CMC_name                  *name,
      CMC_extension             *identifier_to_name_extensions
);

/* LIST CONTAINED PROPERTIES */
CMC_return_code
cmc_list_contained_properties(
      CMC_cursor_handle         cursor,
      CMC_sint32                *number_object,
      CMC_sint32                *number_properties,
      CMC_id                    *property_ids,
      CMC_property              ***properties,
      CMC_extension             *list_contained_properties_extensions
);
```

```
/* LIST NUMBER MATCHED */
CMC_return_code
cmc_list_number_matched(
      CMC_cursor_handle         *cursor,
      CMC_uint32                *number_matches,
      CMC_extension             *list_number_matched_extensions
);

/* LIST OBJECTS */
CMC_return_code
cmc_list_objects(
      CMC_cursor_handle         *cursor,
      CMC_sint32                *number_objects,
      CMC_object_handle         **objects,
      CMC_extension             *list_objects_extensions
);

/* LIST PROPERTIES */
CMC_return_code
cmc_list_properties(
      CMC_object_handle         *object,
      CMC_uint32                *number_properties,
      CMC_id                    **property_ids,
      CMC_extension             *list_properties_extensions
);

/* NAME TO IDENTIFIER */
CMC_return_code
cmc_name_to_identifier(
      CMC_name                  name,
      CMC_id                    *identifier,
      CMC_extension             *name_to_identifier_extensions
);

/* OPEN CURSOR */
CMC_return_code
cmc_open_cursor(
      CMC_object_handle         object,
      CMC_cursor_restriction    *restrictions,
      CMC_uint32                number_sort_orders,
      CMC_cursor_sort_key       *sort_keys,
      CMC_cursor_handle         *cursor,
      CMC_extension             *open_cursor_extensions
);

/* OPEN OBJECT HANDLE */
CMC_return_code
cmc_open_object_handle(
      CMC_session_id            session,
      CMC_id                    object_class,
      CMC_object_handle         *new_object,
      CMC_extension             *open_object_handle_extensions
);

/* READ CURSOR */
CMC_return_code
cmc_read_cursor(
      CMC_cursor_handle         *cursor,
      CMC_uint32                *position_numerator,
      CMC_uint32                *position_denominator,
      CMC_extension             *read_cursor_extensions
);

/* READ PROPERTIES */
CMC_return_code
cmc_read_properties(
      CMC_object_handle         object,
      CMC_uint32                *number_properties,
      CMC_id                    *property_ids,
      CMC_property              **properties,
      CMC_extension             *read_properties_extensions
);
```

```
/* READ PROPERTY COSTS */
CMC_return_code
cmc_read_property_costs(
      CMC_object_handle         object,
      CMC_uint32                *number_properties,
      CMC_id                    *property_ids,
      CMC_enum                  **costs,
      CMC_extension             *read_property_costs_extensions
);

/* RESTORE OBJECT */
CMC_return_code
cmc_restore_object(
      CMC_object_handle         container,
      CMC_string                file_specification,
      CMC_object_handle         *restored_object,
      CMC_flags                 restore_flags,
      CMC_extension             *restore_object_extensions
);

/* SAVE OBJECT */
CMC_return_code
cmc_save_object(
      CMC_object_handle         object,
      CMC_string                file_specification,
      CMC_flags                 save_flags,
      CMC_extension             *save_object_extensions
);

/* SEND MESSAGE OBJECT */
CMC_return_code
cmc_send_message_object(
      CMC_object_handle         message_to_send,
      CMC_extension             *send_message_object_extensions
);

/* UPDATE CURSOR POSITION */
CMC_return_code
cmc_update_cursor_position(
      CMC_cursor_handle         *cursor,
      CMC_uint32                position_numerator,
      CMC_uint32                position_denominator,
      CMC_extension             *update_cursor_position_extensions
);

/* UPDATE CURSOR POSITION WITH SEED */
CMC_return_code
cmc_update_cursor_position_with_seed(
      CMC_cursor_handle         cursor,
      CMC_object_handle         seed,
      CMC_extension             *update_cursor_position_with_seed_extensions
);

/* CHECK EVENT */
CMC_return_code
cmc_check_event(
      CMC_session_id            session,
      CMC_event                 event_type,
      CMC_uint32                minimum_timeout,
      CMC_buffer                check_event_data,
      CMC_buffer                *callback_data,
      CMC_extension             *check_event_extensions
);

/* REGISTER EVENT */
CMC_return_code
cmc_register_event(
      CMC_session_id            session,
      CMC_event                 event_type,
      CMC_callback              callback,
      CMC_buffer                register_data,
      CMC_extension             *register_event_extensions
);
```

```
/* UNREGISTER EVENT */
CMC_return_code
cmc_unregister_event(
      CMC_session_id          session,
      CMC_flags               event_type,
      CMC_callback            callback,
      CMC_buffer              unregister_data,
      CMC_extension           *unregister_event_extensions
);

/* CALL CALLBACKS */
CMC_return_code
cmc_call_callbacks(
      CMC_session_id          session,
      CMC_event               event_type,
      CMC_extension           *call_callbacks_extensions
);

/* EXPORT STREAM */
CMC_return_code
cmc_export_stream(
      CMC_stream_handle       stream,
      CMC_string              file_specification,
      CMC_uint32              count,
      CMC_flags               export_flags,
      CMC_extension           *export_stream_extensions
);

/* IMPORT FILE TO STREAM */
CMC_return_code
cmc_import_file_to_stream(
      CMC_stream_handle       stream,
      CMC_string              file_specification,
      CMC_uint32              file_offset,
      CMC_extension           *import_file_to_stream_extensions
);

/* OPEN STREAM */
CMC_return_code
cmc_open_stream(
      CMC_object_handle       object,
      CMC_property            *property,
      CMC_enum                operation,
      CMC_stream_handle       *stream,
      CMC_extension           *open_stream_extensions
);

/* READ STREAM */
CMC_return_code
cmc_read_stream(
      CMC_stream_handle       stream,
      CMC_uint32              *count,
      CMC_buffer              content_information,
      CMC_extension           *read_stream_extensions
);

/* SEEK STREAM */
CMC_return_code
cmc_seek_stream(
      CMC_stream_handle       stream,
      CMC_enum                operation,
      CMC_uint32              *location,
      CMC_extension           *seek_stream_extensions
);

/* WRITE STREAM */
CMC_return_code
cmc_write_stream(
      CMC_stream_handle       *stream,
      CMC_uint32              *count,
      CMC_buffer              *content_information,
      CMC_extension           *write_stream_extensions
);
```

```
/* GET LAST ERROR */
CMC_return_code
cmc_get_last_error(
      CMC_session_id            session,
      CMC_object_handle         objRef,
      CMC_string                *error_buffer,
      CMC_extension             *get_last_error_extensions
);

/* MULTIPLE IMPLEMENTATIONS DISPATCH TABLE */

typedef struct {
      CMC_extension                 *dispatch_table_extensions;

      /* SEND */
      CMC_return_code
      (*cmc_send)(
         CMC_session_id             session,
         CMC_message                *message,
         CMC_flags                  send_flags,
         CMC_ui_id                  ui_id,
         CMC_extension              *send_extensions
      );

      /* SEND DOCUMENT */
      CMC_return_code
      (*cmc_send_documents)(
         CMC_string                 recipient_addresses,
         CMC_string                 subject,
         CMC_string                 text_note,
         CMC_flags                  send_doc_flags,
         CMC_string                 file_paths,
         CMC_string                 file_names,
         CMC_string                 delimiter,
         CMC_ui_id                  ui_id
      );

      /* ACT ON */
      CMC_return_code
      (*cmc_act_on)(
         CMC_session_id             session,
         CMC_message_reference      *message_reference,
         CMC_enum                   operation,
         CMC_flags                  act_on_flags,
         CMC_ui_id                  ui_id,
         CMC_extension              *act_on_extensions
      );

      /* LIST */
      CMC_return_code
      (*cmc_list)(
         CMC_session_id             session,
         CMC_string                 message_type,
         CMC_flags                  list_flags,
         CMC_message_reference      *seed,
         CMC_uint32                 *count,
         CMC_ui_id                  ui_id,
         CMC_message_summary        **result,
         CMC_extension              *list_extensions
      );

      /* READ */
      CMC_return_code
      (*cmc_read)(
         CMC_session_id             session,
         CMC_message_reference      *message_reference,
         CMC_flags                  read_flags,
         CMC_message                **message,
         CMC_ui_id                  ui_id,
         CMC_extension              *read_extensions
      );
```

```
/* LOOK UP */
CMC_return_code
(*cmc_look_up)(
   CMC_session_id             session,
   CMC_recipient              *recipient_in,
   CMC_flags                  look_up_flags,
   CMC_ui_id                  ui_id,
   CMC_uint32                 *count,
   CMC_recipient              **recipient_out,
   CMC_extension              *look_up_extensions
);

/* FREE */
CMC_return_code
(*cmc_free)(
   CMC_buffer                 memory
);

/* LOGOFF */
CMC_return_code
(*cmc_logoff)(
   CMC_session_id             session,
   CMC_ui_id                  ui_id,
   CMC_flags                  logoff_flags,
   CMC_extension              *logoff_extensions
);

/* LOGON */
CMC_return_code
(*cmc_logon)(
   CMC_string                 service,
   CMC_string                 user,
   CMC_string                 password,
   CMC_object_identifier      character_set,
   CMC_ui_id                  ui_id,
   CMC_uint16                 caller_cmc_version,
   CMC_flags                  logon_flags,
   CMC_session_id             *session,
   CMC_extension              *logon_extensions
);

/* QUERY CONFIGURATION */
CMC_return_code
(*cmc_query_configuration)(
   CMC_session_id             session,
   CMC_enum                   item,
   CMC_buffer                 reference,
   CMC_extension              *config_extensions
);

/* FULL CMC */

/* COPY OBJECT */
CMC_return_code
(*cmc_copy_object)(
   CMC_object_handle          container,
   CMC_object_handle          source_object,
   CMC_object_handle          *new_object,
   CMC_extension              *copy_object_extensions
);

/* ADD PROPERTIES */
CMC_return_code
(*cmc_add_properties)(
   CMC_object_handle          object,
   CMC_uint32                 number_properties,
   CMC_property               *properties,
   CMC_extension              *add_properties_extensions
);
```

```
/* COMMIT OBJECT */
CMC_return_code
(*cmc_commit_object)(
   CMC_object_handle          source_object,
   CMC_extension              *commit_object_extensions
);

/* COPY OBJECT HANDLE */
CMC_return_code
(*cmc_copy_object_handle)(
   CMC_object_handle          source_handle,
   CMC_object_handle          *new_handle,
   CMC_extension              *copy_object_handle_extensions
);

/* CREATE DERIVED MESSAGE OBJECT */
CMC_return_code
(*cmc_create_derived_message_object)(
   CMC_object_handle          original_message,
   CMC_enum                   derived_action,
   CMC_boolean                inherit_contents,
   CMC_object_handle          *derived_message,
   CMC_boolean                modified_message,
   CMC_extension              *create_derived_object_extensions
);

/* DELETE OBJECTS */
CMC_return_code
(*cmc_delete_objects)(
   CMC_uint32                 number_objects,
   CMC_object_handle          *object,
   CMC_extension              *delete_objects_extensions
);

/* DELETE PROPERTIES */
CMC_return_code
(*cmc_delete_properties)(
   CMC_object_handle          object,
   CMC_uint32                 number_properties,
   CMC_id                     *property_ids,
   CMC_extension              *delete_properties_extensions
);

/* GET ROOT HANDLE */
CMC_return_code
(*cmc_get_root_handle)(
   CMC_session_id             session,
   CMC_object_handle          *root_object_handle,
   CMC_extension              *get_root_handle_extensions
);

/* IDENTIFIER TO NAME */
CMC_return_code
(*cmc_identifier_to_name)(
   CMC_id                     identifier,
   CMC_name                   *name,
   CMC_extension              *identifier_to_name_extensions
);

/* LIST CONTAINED PROPERTIES */
CMC_return_code
(*cmc_list_contained_properties)(
   CMC_cursor_handle          cursor,
   CMC_sint32                 *number_object,
   CMC_sint32                 *number_properties,
   CMC_id                     *property_ids,
   CMC_property               ***properties,
   CMC_extension              *list_contained_properties_extensions
);
```

```
/* LIST NUMBER MATCHED */
CMC_return_code
(*cmc_list_number_matched)(
   CMC_cursor_handle          *cursor,
   CMC_uint32                 *number_matches,
   CMC_extension              *list_number_matched_extensions
);

/* LIST OBJECTS */
CMC_return_code
(*cmc_list_objects)(
   CMC_cursor_handle          *cursor,
   CMC_sint32                 *number_objects,
   CMC_object_handle          **objects,
   CMC_extension              *list_objects_extensions
);

/* LIST PROPERTIES */
CMC_return_code
(*cmc_list_properties)(
   CMC_object_handle          *object,
   CMC_uint32                 *number_properties,
   CMC_id                     **property_ids,
   CMC_extension              *list_properties_extensions
);

/* NAME TO IDENTIFIER */
CMC_return_code
(*cmc_name_to_identifier)(
   CMC_name                   name,
   CMC_id                     *identifier,
   CMC_extension              *name_to_identifier_extensions
);

/* OPEN CURSOR */
CMC_return_code
(*cmc_open_cursor)(
   CMC_object_handle          object,
   CMC_cursor_restriction     *restrictions,
   CMC_uint32                 number_sort_orders,
   CMC_cursor_sort_key        *sort_keys,
   CMC_cursor_handle          *cursor,
   CMC_extension              *open_cursor_extensions
);

/* OPEN OBJECT HANDLE */
CMC_return_code
(*cmc_open_object_handle)(
   CMC_session_id             session,
   CMC_id                     object_class,
   CMC_object_handle          *new_object,
   CMC_extension              *open_object_handle_extensions
);

/* READ CURSOR */
CMC_return_code
(*cmc_read_cursor)(
   CMC_cursor_handle          *cursor,
   CMC_uint32                 *position_numerator,
   CMC_uint32                 *position_denominator,
   CMC_extension              *read_cursor_extensions
);

/* READ PROPERTIES */
CMC_return_code
(*cmc_read_properties)(
   CMC_object_handle          object,
   CMC_uint32                 *number_properties,
   CMC_id                     *property_ids,
   CMC_property               **properties,
   CMC_extension              *read_properties_extensions
);
```

```
/* READ PROPERTY COSTS */
CMC_return_code
(*cmc_read_property_costs)(
   CMC_object_handle          object,
   CMC_uint32                 *number_properties,
   CMC_id                     *property_ids,
   CMC_enum                   **costs,
   CMC_extension              *read_property_costs_extensions
);

/* RESTORE OBJECT */
CMC_return_code
(*cmc_restore_object)(
   CMC_object_handle          container,
   CMC_string                 file_specification,
   CMC_object_handle          *restored_object,
   CMC_flags                  restore_flags,
   CMC_extension              *restore_object_extensions
);

/* SAVE OBJECT */
CMC_return_code
(*cmc_save_object)(
   CMC_object_handle          object,
   CMC_string                 file_specification,
   CMC_flags                  save_flags,
   CMC_extension              *save_object_extensions
);

/* SEND MESSAGE OBJECT */
CMC_return_code
(*cmc_send_message_object)(
   CMC_object_handle          message_to_send,
   CMC_extension              *send_message_object_extensions
);

/* UPDATE CURSOR POSITION */
CMC_return_code
(*cmc_update_cursor_position)(
   CMC_cursor_handle          *cursor,
   CMC_uint32                 position_numerator,
   CMC_uint32                 position_denominator,
   CMC_extension              *update_cursor_position_extensions
);

/* UPDATE CURSOR POSITION WITH SEED */
CMC_return_code
(*cmc_update_cursor_position_with_seed)(
   CMC_cursor_handle          cursor,
   CMC_object_handle          seed,
   CMC_extension              *update_cursor_position_with_seed_extensions
);

/* CHECK EVENT */
CMC_return_code
(*cmc_check_event)(
   CMC_session_id             session,
   CMC_event                  event_type,
   CMC_uint32                 minimum_timeout,
   CMC_buffer                 check_event_data,
   CMC_buffer                 *callback_data,
   CMC_extension              *check_event_extensions
);

/* REGISTER EVENT */
CMC_return_code
(*cmc_register_event)(
   CMC_session_id             session,
   CMC_event                  event_type,
   CMC_callback               callback,
   CMC_buffer                 register_data,
   CMC_extension              *register_event_extensions
);
```

```
/* UNREGISTER EVENT */
CMC_return_code
(*cmc_unregister_event)(
   CMC_session_id             session,
   CMC_flags                  event_type,
   CMC_callback               callback,
   CMC_buffer                 unregister_data,
   CMC_extension              *unregister_event_extensions
);

/* CALL CALLBACKS */
CMC_return_code
(*cmc_call_callbacks)(
   CMC_session_id             session,
   CMC_event                  event_type,
   CMC_extension              *call_callbacks_extensions
);

/* EXPORT STREAM */
CMC_return_code
(*cmc_export_stream)(
   CMC_stream_handle          stream,
   CMC_string                 file_specification,
   CMC_uint32                 count,
   CMC_flags                  export_flags,
   CMC_extension              *export_stream_extensions
);

/* IMPORT FILE TO STREAM */
CMC_return_code
(*cmc_import_file_to_stream)(
   CMC_stream_handle          stream,
   CMC_string                 file_specification,
   CMC_uint32                 file_offset,
   CMC_extension              *import_file_to_stream_extensions
);

/* OPEN STREAM */
CMC_return_code
(*cmc_open_stream)(
   CMC_object_handle          object,
   CMC_property               *property,
   CMC_enum                   operation,
   CMC_stream_handle          *stream,
   CMC_extension              *open_stream_extensions
);

/* READ STREAM */
CMC_return_code
(*cmc_read_stream)(
   CMC_stream_handle          stream,
   CMC_uint32                 *count,
   CMC_buffer                 content_information,
   CMC_extension              *read_stream_extensions
);

/* SEEK STREAM */
CMC_return_code
(*cmc_seek_stream)(
   CMC_stream_handle          stream,
   CMC_enum                   operation,
   CMC_uint32                 *location,
   CMC_extension              *seek_stream_extensions
);

/* WRITE STREAM */
CMC_return_code
(*cmc_write_stream)(
   CMC_stream_handle          *stream,
   CMC_uint32                 *count,
   CMC_buffer                 *content_information,
   CMC_extension              *write_stream_extensions
);
```

```
      /* GET LAST ERROR */
      CMC_return_code
      (*cmc_get_last_error)(
         CMC_session_id              session,
         CMC_object_handle           objRef,
         CMC_string                  *error_buffer,
         CMC_extension               *get_last_error_extensions
      );

} CMC_dispatch_table;

/* BIND IMPLEMENTATION */
CMC_return_code
cmc_bind_implementation(
      CMC_guid                  implementation_name,
      CMC_dispatch_table        **dispatch_table,
      CMC_extension             *cmc_bind_extensions
);

/* UNBIND IMPLEMENTATION */
CMC_return_code
cmc_unbind_implementation(
      CMC_guid                  implementation_name,
      CMC_extension             *cmc_unbind_implementation_extensions
);

#else   /* WCHAR / UNICODE Function Counterparts */

/* SEND */
CMC_return_code
cmc_send_W(
      CMC_session_id            session,
      CMC_message               *message,
      CMC_flags                 send_flags,
      CMC_ui_id                 ui_id,
      CMC_extension             *send_extensions
);

/* SEND DOCUMENT */
CMC_return_code
cmc_send_documents_W(
      CMC_string                recipient_addresses,
      CMC_string                subject,
      CMC_string                text_note,
      CMC_flags                 send_doc_flags,
      CMC_string                file_paths,
      CMC_string                file_names,
      CMC_string                delimiter,
      CMC_ui_id                 ui_id
);

/* ACT ON */
CMC_return_code
cmc_act_on_W(
      CMC_session_id            session,
      CMC_message_reference     *message_reference,
      CMC_enum                  operation,
      CMC_flags                 act_on_flags,
      CMC_ui_id                 ui_id,
      CMC_extension             *act_on_extensions
);

/* LIST */
CMC_return_code
cmc_list_W(
      CMC_session_id            session,
      CMC_string                message_type,
      CMC_flags                 list_flags,
      CMC_message_reference     *seed,
      CMC_uint32                *count,
      CMC_ui_id                 ui_id,
      CMC_message_summary       **result,
      CMC_extension             *list_extensions
);
```

```
/* READ */
CMC_return_code
cmc_read_W(
      CMC_session_id            session,
      CMC_message_reference     *message_reference,
      CMC_flags                 read_flags,
      CMC_message               **message,
      CMC_ui_id                 ui_id,
      CMC_extension             *read_extensions
);

/* LOOK UP */
CMC_return_code
cmc_look_up_W(
      CMC_session_id            session,
      CMC_recipient             *recipient_in,
      CMC_flags                 look_up_flags,
      CMC_ui_id                 ui_id,
      CMC_uint32                *count,
      CMC_recipient             **recipient_out,
      CMC_extension             *look_up_extensions
);

/* FREE */
CMC_return_code
cmc_free_W(
      CMC_buffer                memory
);

/* LOGOFF */
CMC_return_code
cmc_logoff_W(
      CMC_session_id            session,
      CMC_ui_id                 ui_id,
      CMC_flags                 logoff_flags,
      CMC_extension             *logoff_extensions
);

/* LOGON */
CMC_return_code
cmc_logon_W(
      CMC_string                service,
      CMC_string                user,
      CMC_string                password,
      CMC_object_identifier     character_set,
      CMC_ui_id                 ui_id,
      CMC_uint16                caller_cmc_version,
      CMC_flags                 logon_flags,
      CMC_session_id            *session,
      CMC_extension             *logon_extensions
);

/* QUERY CONFIGURATION */
CMC_return_code
cmc_query_configuration_W(
      CMC_session_id            session,
      CMC_enum                  item,
      CMC_buffer                reference,
      CMC_extension             *config_extensions
);

/* FULL CMC */

/* COPY OBJECT */
CMC_return_code
cmc_copy_object_W(
      CMC_object_handle         container,
      CMC_object_handle         source_object,
      CMC_object_handle         *new_object,
      CMC_extension             *copy_object_extensions
);
```

```
/* ADD PROPERTIES */
CMC_return_code
cmc_add_properties_W(
      CMC_object_handle        object,
      CMC_uint32               number_properties,
      CMC_property             *properties,
      CMC_extension            *add_properties_extensions
);

/* COMMIT OBJECT */
CMC_return_code
cmc_commit_object_W(
      CMC_object_handle        source_object,
      CMC_extension            *commit_object_extensions
);

/* COPY OBJECT HANDLE */
CMC_return_code
cmc_copy_object_handle_W(
      CMC_object_handle        source_handle,
      CMC_object_handle        *new_handle,
      CMC_extension            *copy_object_handle_extensions
);

/* CREATE DERIVED MESSAGE OBJECT */
CMC_return_code
cmc_create_derived_message_object_W(
      CMC_object_handle        original_message,
      CMC_enum                 derived_action,
      CMC_boolean              inherit_contents,
      CMC_object_handle        *derived_message,
      CMC_boolean              modified_message,
      CMC_extension            *create_derived_object_extensions
);

/* DELETE OBJECTS */
CMC_return_code
cmc_delete_objects_W(
      CMC_uint32               number_objects,
      CMC_object_handle        *object,
      CMC_extension            *delete_objects_extensions
);

/* DELETE PROPERTIES */
CMC_return_code
cmc_delete_properties_W(
      CMC_object_handle        object,
      CMC_uint32               number_properties,
      CMC_id                   *property_ids,
      CMC_extension            *delete_properties_extensions
);

/* GET ROOT HANDLE */
CMC_return_code
cmc_get_root_handle_W(
      CMC_session_id           session,
      CMC_object_handle        *root_object_handle,
      CMC_extension            *get_root_handle_extensions
);

/* IDENTIFIER TO NAME */
CMC_return_code
cmc_identifier_to_name_W(
      CMC_id                   identifier,
      CMC_name                 *name,
      CMC_extension            *identifier_to_name_extensions
);
```

```
/* LIST CONTAINED PROPERTIES */
CMC_return_code
cmc_list_contained_properties_W(
      CMC_cursor_handle         cursor,
      CMC_sint32                *number_object,
      CMC_sint32                *number_properties,
      CMC_id                    *property_ids,
      CMC_property              ***properties,
      CMC_extension             *list_contained_properties_extensions
);

/* LIST NUMBER MATCHED */
CMC_return_code
cmc_list_number_matched_W(
      CMC_cursor_handle         *cursor,
      CMC_uint32                *number_matches,
      CMC_extension             *list_number_matched_extensions
);

/* LIST OBJECTS */
CMC_return_code
cmc_list_objects_W(
      CMC_cursor_handle         *cursor,
      CMC_sint32                *number_objects,
      CMC_object_handle         **objects,
      CMC_extension             *list_objects_extensions
);

/* LIST PROPERTIES */
CMC_return_code
cmc_list_properties_W(
      CMC_object_handle         *object,
      CMC_uint32                *number_properties,
      CMC_id                    **property_ids,
      CMC_extension             *list_properties_extensions
);

/* NAME TO IDENTIFIER */
CMC_return_code
cmc_name_to_identifier_W(
      CMC_name                  property_name,
      CMC_id                    *property_id,
      CMC_extension             *name_to_identifier_extensions
);

/* OPEN CURSOR */
CMC_return_code
cmc_open_cursor_W(
      CMC_object_handle         object,
      CMC_cursor_restriction    *restrictions,
      CMC_uint32                number_sort_orders,
      CMC_cursor_sort_key       *sort_keys,
      CMC_cursor_handle         *cursor,
      CMC_extension             *open_cursor_extensions
);

/* OPEN OBJECT HANDLE */
CMC_return_code
cmc_open_object_handle_W(
      CMC_session_id            session,
      CMC_id                    object_class,
      CMC_object_handle         *new_object,
      CMC_extension             *open_object_handle_extensions
);

/* READ CURSOR */
CMC_return_code
cmc_read_cursor_W(
      CMC_cursor_handle         *cursor,
      CMC_uint32                *position_numerator,
      CMC_uint32                *position_denominator,
      CMC_extension             *read_cursor_extensions
);
```

```
/* READ PROPERTIES */
CMC_return_code
cmc_read_properties_W(
      CMC_object_handle         object,
      CMC_uint32                *number_properties,
      CMC_id                    *property_ids,
      CMC_property              **properties,
      CMC_extension             *read_properties_extensions
);

/* READ PROPERTY COSTS */
CMC_return_code
cmc_read_property_costs_W(
      CMC_object_handle         object,
      CMC_uint32                *number_properties,
      CMC_id                    *property_ids,
      CMC_enum                  **costs,
      CMC_extension             *read_property_costs_extensions
);

/* RESTORE OBJECT */
CMC_return_code
cmc_restore_object_W(
      CMC_object_handle         container,
      CMC_string                file_specification,
      CMC_object_handle         *restored_object,
      CMC_flags                 restore_flags,
      CMC_extension             *restore_object_extensions
);

/* SAVE OBJECT */
CMC_return_code
cmc_save_object_W(
      CMC_object_handle         object,
      CMC_string                file_specification,
      CMC_flags                 save_flags,
      CMC_extension             *save_object_extensions
);

/* SEND MESSAGE OBJECT */
CMC_return_code
cmc_send_message_object_W(
      CMC_object_handle         message_to_send,
      CMC_extension             *send_message_object_extensions
);

/* UPDATE CURSOR POSITION */
CMC_return_code
cmc_update_cursor_position_W(
      CMC_cursor_handle         *cursor,
      CMC_uint32                position_numerator,
      CMC_uint32                position_denominator,
      CMC_extension             *update_cursor_position_extensions
);

/* UPDATE CURSOR POSITION WITH SEED */
CMC_return_code
cmc_update_cursor_position_with_seed_W(
      CMC_cursor_handle         cursor,
      CMC_object_handle         seed,
      CMC_extension             *update_cursor_position_with_seed_extensions
);

/* CHECK EVENT */
CMC_return_code
cmc_check_event_W(
      CMC_session_id            session,
      CMC_event                 event_type,
      CMC_uint32                minimum_timeout,
      CMC_buffer                check_event_data,
      CMC_buffer                *callback_data,
      CMC_extension             *check_event_extensions
);
```

```
/* REGISTER EVENT */
CMC_return_code
cmc_register_event_W(
      CMC_session_id           session,
      CMC_event                event_type,
      CMC_callback             callback,
      CMC_buffer               register_data,
      CMC_extension            *register_event_extensions
);
/* UNREGISTER EVENT */
CMC_return_code
cmc_unregister_event_W(
      CMC_session_id           session,
      CMC_flags                event_type,
      CMC_callback             callback,
      CMC_buffer               unregister_data,
      CMC_extension            *unregister_event_extensions
);
/* CALL CALLBACKS */
CMC_return_code
cmc_call_callbacks_W(
      CMC_session_id           session,
      CMC_event                event_type,
      CMC_extension            *call_callbacks_extensions
);
/* EXPORT STREAM */
CMC_return_code
cmc_export_stream_W(
      CMC_stream_handle        stream,
      CMC_string               file_specification,
      CMC_uint32               count,
      CMC_flags                export_flags,
      CMC_extension            *export_stream_extensions
);
/* IMPORT FILE TO STREAM */
CMC_return_code
cmc_import_file_to_stream_W(
      CMC_stream_handle        stream,
      CMC_string               file_specification,
      CMC_uint32               file_offset,
      CMC_extension            *import_file_to_stream_extensions
);
/* OPEN STREAM */
CMC_return_code
cmc_open_stream_W(
      CMC_object_handle        object,
      CMC_property             *property,
      CMC_enum                 operation,
      CMC_stream_handle        *stream,
      CMC_extension            *open_stream_extensions
);
/* READ STREAM */
CMC_return_code
cmc_read_stream_W(
      CMC_stream_handle        stream,
      CMC_uint32               *count,
      CMC_buffer               content_information,
      CMC_extension            *read_stream_extensions
);
/* SEEK STREAM */
CMC_return_code
cmc_seek_stream_W(
      CMC_stream_handle        stream,
      CMC_enum                 operation,
      CMC_uint32               *location,
      CMC_extension            *seek_stream_extensions
);
```

```
/* WRITE STREAM */
CMC_return_code
cmc_write_stream_W(
      CMC_stream_handle         *stream,
      CMC_uint32                *count,
      CMC_buffer                *content_information,
      CMC_extension             *write_stream_extensions
);

/* GET LAST ERROR */
CMC_return_code
cmc_get_last_error_W(
      CMC_session_id            session,
      CMC_object_handle         objRef,
      CMC_string                *error_buffer,
      CMC_extension             *get_last_error_extensions
);

/* MULTIPLE IMPLEMENTATIONS DISPATCH TABLE UNICODE */

typedef struct {
      CMC_extension                 *dispatch_table_extensions;
      /* SEND */
      CMC_return_code
      (*cmc_send_W)(
            CMC_session_id          session,
            CMC_message             *message,
            CMC_flags               send_flags,
            CMC_ui_id               ui_id,
            CMC_extension           *send_extensions
      );

      /* SEND DOCUMENT */
      CMC_return_code
      (*cmc_send_documents_W)(
            CMC_string              recipient_addresses,
            CMC_string              subject,
            CMC_string              text_note,
            CMC_flags               send_doc_flags,
            CMC_string              file_paths,
            CMC_string              file_names,
            CMC_string              delimiter,
            CMC_ui_id               ui_id
      );

      /* ACT ON */
      CMC_return_code
      (*cmc_act_on_W)(
            CMC_session_id          session,
            CMC_message_reference   *message_reference,
            CMC_enum                operation,
            CMC_flags               act_on_flags,
            CMC_ui_id               ui_id,
            CMC_extension           *act_on_extensions
      );

      /* LIST */
      CMC_return_code
      (*cmc_list_W)(
            CMC_session_id          session,
            CMC_string              message_type,
            CMC_flags               list_flags,
            CMC_message_reference   *seed,
            CMC_uint32              *count,
            CMC_ui_id               ui_id,
            CMC_message_summary     **result,
            CMC_extension           *list_extensions
      );
```

```
/* READ */
CMC_return_code
(*cmc_read_W)(
      CMC_session_id          session,
      CMC_message_reference   *message_reference,
      CMC_flags               read_flags,
      CMC_message             **message,
      CMC_ui_id               ui_id,
      CMC_extension           *read_extensions
);

/* LOOK UP */
CMC_return_code
(*cmc_look_up_W)(
      CMC_session_id          session,
      CMC_recipient           *recipient_in,
      CMC_flags               look_up_flags,
      CMC_ui_id               ui_id,
      CMC_uint32              *count,
      CMC_recipient           **recipient_out,
      CMC_extension           *look_up_extensions
);

/* FREE */
CMC_return_code
(*cmc_free_W)(
      CMC_buffer              memory
);

/* LOGOFF */
CMC_return_code
(*cmc_logoff_W)(
      CMC_session_id          session,
      CMC_ui_id               ui_id,
      CMC_flags               logoff_flags,
      CMC_extension           *logoff_extensions
);

/* LOGON */
CMC_return_code
(*cmc_logon_W)(
      CMC_string              service,
      CMC_string              user,
      CMC_string              password,
      CMC_object_identifier   character_set,
      CMC_ui_id               ui_id,
      CMC_uint16              caller_cmc_version,
      CMC_flags               logon_flags,
      CMC_session_id          *session,
      CMC_extension           *logon_extensions
);

/* QUERY CONFIGURATION */
CMC_return_code
(*cmc_query_configuration_W)(
      CMC_session_id          session,
      CMC_enum                item,
      CMC_buffer              reference,
      CMC_extension           *config_extensions
);

/* FULL CMC */

/* COPY OBJECT */
CMC_return_code
(*cmc_copy_object_W)(
      CMC_object_handle       container,
      CMC_object_handle       source_object,
      CMC_object_handle       *new_object,
      CMC_extension           *copy_object_extensions
);
```

```
/* ADD PROPERTIES */
CMC_return_code
(*cmc_add_properties_W)(
      CMC_object_handle      object,
      CMC_uint32             number_properties,
      CMC_property           *properties,
      CMC_extension          *add_properties_extensions
);

/* COMMIT OBJECT */
CMC_return_code
(*cmc_commit_object_W)(
      CMC_object_handle      source_object,
      CMC_extension          *commit_object_extensions
);

/* COPY OBJECT HANDLE */
CMC_return_code
(*cmc_copy_object_handle_W)(
      CMC_object_handle      source_handle,
      CMC_object_handle      *new_handle,
      CMC_extension          *copy_object_handle_extensions
);

/* CREATE DERIVED MESSAGE OBJECT */
CMC_return_code
(*cmc_create_derived_message_object_W)(
      CMC_object_handle      original_message,
      CMC_enum               derived_action,
      CMC_boolean            inherit_contents,
      CMC_object_handle      *derived_message,
      CMC_boolean            modified_message,
      CMC_extension          *create_derived_object_extensions
);

/* DELETE OBJECTS */
CMC_return_code
(*cmc_delete_objects_W)(
      CMC_uint32             number_objects,
      CMC_object_handle      *object,
      CMC_extension          *delete_objects_extensions
);

/* DELETE PROPERTIES */
CMC_return_code
(*cmc_delete_properties_W)(
      CMC_object_handle      object,
      CMC_uint32             number_properties,
      CMC_id                 *property_ids,
      CMC_extension          *delete_properties_extensions
);

/* GET ROOT HANDLE */
CMC_return_code
(*cmc_get_root_handle_W)(
      CMC_session_id         session,
      CMC_object_handle      *root_object_handle,
      CMC_extension          *get_root_handle_extensions
);

/* IDENTIFIER TO NAME */
CMC_return_code
(*cmc_identifier_to_name_W)(
      CMC_id                 identifier,
      CMC_name               *name,
      CMC_extension          *identifier_to_name_extensions
);
```

```
/* LIST CONTAINED PROPERTIES */
CMC_return_code
(*cmc_list_contained_properties_W)(
      CMC_cursor_handle       cursor,
      CMC_sint32              *number_object,
      CMC_sint32              *number_properties,
      CMC_id                  *property_ids,
      CMC_property            ***properties,
      CMC_extension           *list_contained_properties_extensions
);

/* LIST NUMBER MATCHED */
CMC_return_code
(*cmc_list_number_matched_W)(
      CMC_cursor_handle       *cursor,
      CMC_uint32              *number_matches,
      CMC_extension           *list_number_matched_extensions
);

/* LIST OBJECTS */
CMC_return_code
(*cmc_list_objects_W)(
      CMC_cursor_handle       *cursor,
      CMC_sint32              *number_objects,
      CMC_object_handle       **objects,
      CMC_extension           *list_objects_extensions
);

/* LIST PROPERTIES */
CMC_return_code
(*cmc_list_properties_W)(
      CMC_object_handle       *object,
      CMC_uint32              *number_properties,
      CMC_id                  **property_ids,
      CMC_extension           *list_properties_extensions
);

/* NAME TO IDENTIFIER */
CMC_return_code
(*cmc_name_to_identifier_W)(
      CMC_name                name,
      CMC_id                  *identifier,
      CMC_extension           *name_to_identifier_extensions
);

/* OPEN CURSOR */
CMC_return_code
(*cmc_open_cursor_W)(
      CMC_object_handle       object,
      CMC_cursor_restriction  *restrictions,
      CMC_uint32              number_sort_orders,
      CMC_cursor_sort_key     *sort_keys,
      CMC_cursor_handle       *cursor,
      CMC_extension           *open_cursor_extensions
);

/* OPEN OBJECT HANDLE */
CMC_return_code
(*cmc_open_object_handle_W)(
      CMC_session_id          session,
      CMC_id                  object_class,
      CMC_object_handle       *new_object,
      CMC_extension           *open_object_handle_extensions
);

/* READ CURSOR */
CMC_return_code
(*cmc_read_cursor_W)(
      CMC_cursor_handle       *cursor,
      CMC_uint32              *position_numerator,
      CMC_uint32              *position_denominator,
      CMC_extension           *read_cursor_extensions
);
```

```
/* READ PROPERTIES */
CMC_return_code
(*cmc_read_properties_W)(
      CMC_object_handle       object,
      CMC_uint32              *number_properties,
      CMC_id                  *property_ids,
      CMC_property            **properties,
      CMC_extension           *read_properties_extensions
);

/* READ PROPERTY COSTS */
CMC_return_code
(*cmc_read_property_costs_W)(
      CMC_object_handle       object,
      CMC_uint32              *number_properties,
      CMC_id                  *property_ids,
      CMC_enum                **costs,
      CMC_extension           *read_property_costs_extensions
);

/* RESTORE OBJECT */
CMC_return_code
(*cmc_restore_object_W)(
      CMC_object_handle       container,
      CMC_string              file_specification,
      CMC_object_handle       *restored_object,
      CMC_flags               restore_flags,
      CMC_extension           *restore_object_extensions
);

/* SAVE OBJECT */
CMC_return_code
(*cmc_save_object_W)(
      CMC_object_handle       object,
      CMC_string              file_specification,
      CMC_flags               save_flags,
      CMC_extension           *save_object_extensions
);

/* SEND MESSAGE OBJECT */
CMC_return_code
(*cmc_send_message_object_W)(
      CMC_object_handle       message_to_send,
      CMC_extension           *send_message_object_extensions
);

/* UPDATE CURSOR POSITION */
CMC_return_code
(*cmc_update_cursor_position_W)(
      CMC_cursor_handle       *cursor,
      CMC_uint32              position_numerator,
      CMC_uint32              position_denominator,
      CMC_extension           *update_cursor_position_extensions
);

/* UPDATE CURSOR POSITION WITH SEED */
CMC_return_code
(*cmc_update_cursor_position_with_seed_W)(
      CMC_cursor_handle       cursor,
      CMC_object_handle       seed,
      CMC_extension           *update_cursor_position_with_seed_extensions
);

/* CHECK EVENT */
CMC_return_code
(*cmc_check_event_W)(
      CMC_session_id          session,
      CMC_event               event_type,
      CMC_uint32              minimum_timeout,
      CMC_buffer              check_event_data,
      CMC_buffer              *callback_data,
      CMC_extension           *check_event_extensions
);
```

```
/* REGISTER EVENT */
CMC_return_code
(*cmc_register_event_W)(
      CMC_session_id          session,
      CMC_event               event_type,
      CMC_callback            callback,
      CMC_buffer              register_data,
      CMC_extension           *register_event_extensions
);

/* UNREGISTER EVENT */
CMC_return_code
(*cmc_unregister_event_W)(
      CMC_session_id          session,
      CMC_flags               event_type,
      CMC_callback            callback,
      CMC_buffer              unregister_data,
      CMC_extension           *unregister_event_extensions
);

/* CALL CALLBACKS */
CMC_return_code
(*cmc_call_callbacks_W)(
      CMC_session_id          session,
      CMC_event               event_type,
      CMC_extension           *call_callbacks_extensions
);

/* EXPORT STREAM */
CMC_return_code
(*cmc_export_stream_W)(
      CMC_stream_handle       stream,
      CMC_string              file_specification,
      CMC_uint32              count,
      CMC_flags               export_flags,
      CMC_extension           *export_stream_extensions
);

/* IMPORT FILE TO STREAM */
CMC_return_code
(*cmc_import_file_to_stream_W)(
      CMC_stream_handle       stream,
      CMC_string              file_specification,
      CMC_uint32              file_offset,
      CMC_extension           *import_file_to_stream_extensions
);

/* OPEN STREAM */
CMC_return_code
(*cmc_open_stream_W)(
      CMC_object_handle       object,
      CMC_property            *property,
      CMC_enum                operation,
      CMC_stream_handle       *stream,
      CMC_extension           *open_stream_extensions
);

/* READ STREAM */
CMC_return_code
(*cmc_read_stream_W)(
      CMC_stream_handle       stream,
      CMC_uint32              *count,
      CMC_buffer              content_information,
      CMC_extension           *read_stream_extensions
);

/* SEEK STREAM */
CMC_return_code
(*cmc_seek_stream_W)(
      CMC_stream_handle       stream,
      CMC_enum                operation,
      CMC_uint32              *location,
      CMC_extension           *seek_stream_extensions
);
```

```
        /* WRITE STREAM */
        CMC_return_code
        (*cmc_write_stream_W)(
              CMC_stream_handle       *stream,
              CMC_uint32              *count,
              CMC_buffer              *content_information,
              CMC_extension           *write_stream_extensions
        );

        /* GET LAST ERROR */
        CMC_return_code
        (*cmc_get_last_error_W)(
              CMC_session_id          session,
              CMC_object_handle       objRef,
              CMC_string              *error_buffer,
              CMC_extension           *get_last_error_extensions
        );

} CMC_dispatch_table;

/* BIND IMPLEMENTATION */
CMC_return_code
cmc_bind_implementation_W(
        CMC_guid                  implementation_name,
        CMC_dispatch_table        **dispatch_table,
        CMC_extension             *cmc_bind_extensions
);

/* UNBIND IMPLEMENTATION */
CMC_return_code
cmc_unbind_implementation_W(
        CMC_guid                  implementation_name,
        CMC_extension             *cmc_unbind_implementation_extensions
);

#endif
typedef CMC_return_code (*CMC_P_BIND_IMPLEMENTATION)(
        CMC_guid                  implementation_name,
        CMC_dispatch_table        **dispatch_table,
        CMC_extension             *cmc_bind_extensions
);
typedef CMC_return_code (*CMC_P_UNBIND_IMPLEMENTATION)(
        CMC_guid                  implementation_name,
        CMC_extension             *cmc_unbind_extensions
);

/* Function Constants */

/* SEND */
#define CMC_SEND_UI_REQUESTED                   ((CMC_flags) 1)

/* SEND DOCUMENT */
#define CMC_FIRST_ATTACH_AS_TEXT_NOTE           ((CMC_flags) 2)

/* ACT ON */
#define CMC_ACT_ON_EXTENDED                     ((CMC_enum)  0)
#define CMC_ACT_ON_DELETE                       ((CMC_enum)  1)

/* LIST */
#define CMC_LIST_UNREAD_ONLY                    ((CMC_flags) 1)
#define CMC_LIST_MSG_REFS_ONLY                  ((CMC_flags) 2)
#define CMC_LIST_COUNT_ONLY                     ((CMC_flags) 4)

#define CMC_LENGTH_UNKNOWN                      0xFFFFFFFF

/* READ */
#define CMC_DO_NOT_MARK_AS_READ                 ((CMC_flags) 1)
#define CMC_MSG_AND_ATT_HDRS_ONLY               ((CMC_flags) 2)
#define CMC_READ_FIRST_UNREAD_MESSAGE           ((CMC_flags) 4)

/* LOOKUP */
#define CMC_LOOKUP_RESOLVE_PREFIX_SEARCH        ((CMC_flags) 1)
#define CMC_LOOKUP_RESOLVE_IDENTITY             ((CMC_flags) 2)
#define CMC_LOOKUP_RESOLVE_UI                   ((CMC_flags) 4)
#define CMC_LOOKUP_DETAILS_UI                   ((CMC_flags) 8)
#define CMC_LOOKUP_ADDRESSING_UI                ((CMC_flags) 16)
```

```
/* LOGOFF */
#define CMC_LOGOFF_UI_ALLOWED                       ((CMC_flags) 1)

/* LOGON */
#define CMC_VERSION                                 ((CMC_uint16) 100)

/* QUERY CONFIGURATION ENUMS */
#define CMC_CONFIG_CHARACTER_SET                    ((CMC_enum) 1)
#define CMC_CONFIG_LINE_TERM                        ((CMC_enum) 2)
#define CMC_CONFIG_DEFAULT_SERVICE                  ((CMC_enum) 3)
#define CMC_CONFIG_DEFAULT_USER                     ((CMC_enum) 4)
#define CMC_CONFIG_REQ_PASSWORD                     ((CMC_enum) 5)
#define CMC_CONFIG_REQ_SERVICE                      ((CMC_enum) 6)
#define CMC_CONFIG_REQ_USER                         ((CMC_enum) 7)
#define CMC_CONFIG_UI_AVAIL                         ((CMC_enum) 8)
#define CMC_CONFIG_SUP_NOMKMSGREAD                  ((CMC_enum) 9)
#define CMC_CONFIG_SUP_COUNTED_STR                  ((CMC_enum) 10)
#define CMC_CONFIG_VER_IMPLEM                       ((CMC_enum) 11)
#define CMC_CONFIG_VER_SPEC                         ((CMC_enum) 12)

/* CONFIG LINE TERM ENUM */
#define CMC_LINE_TERM_CRLF                          ((CMC_enum) 0)
#define CMC_LINE_TERM_CR                            ((CMC_enum) 1)
#define CMC_LINE_TERM_LF                            ((CMC_enum) 2)

/* CONFIG REQUIRED LOGON PARAMETER ENUM */
#define CMC_REQUIRED_NO                             ((CMC_enum) 0)
#define CMC_REQUIRED_YES                            ((CMC_enum) 1)
#define CMC_REQUIRED_OPT                            ((CMC_enum) 2)

/* CREATE DERIVED MESSAGE OBJECT */
#define CMC_DERIVED_ACTION_FORWARD                  ((CMC_enum) 0)
#define CMC_DERIVED_ACTION_REPLY_ORIGINATOR         ((CMC_enum) 1)
#define CMC_DERIVED_ACTION_REPLY_ALL                ((CMC_enum) 2)

/* READ PROPERTY COSTS */
#define CMC_COST_UNDETERMINED                       ((CMC_enum) 0)
#define CMC_COST_NONE                               ((CMC_enum) 1)
#define CMC_COST_MINOR                              ((CMC_enum) 2)
#define CMC_COST_MAJOR                              ((CMC_enum) 3)

/* RESTORE OBJECT FLAGS */
#define CMC_RESTORE_OBJECT_OVERWRITE                ((CMC_flags) 1)

/* SAVE OBJECT FLAGS */
#define CMC_SAVE_OBJECT_OVERWRITE                   ((CMC_flags) 1)
#define CMC_SAVE_OBJECT_NOCREATE                    ((CMC_flags) 2)

/* EXPORT STREAM */
#define CMC_EXPORT_STREAM_OVERWRITE                 ((CMC_flags) 1)
#define CMC_EXPORT_STREAM_NOCREATE                  ((CMC_flags) 2)
#define CMC_EXPORT_STREAM_APPEND                    ((CMC_flags) 3)

/* OPEN STREAM */
#define CMC_OPEN_READ                               ((CMC_enum) 0)
#define CMC_OPEN_WRITE                              ((CMC_enum) 1)

/* SEEK STREAM */
#define CMC_SEEK_BEGINNING                          ((CMC_enum) 0)
#define CMC_SEEK_END                                ((CMC_enum) 1)
#define CMC_SEEK_CURRENT_POSITION                   ((CMC_enum) 2)

/* DEFINED OBJECT ID’S FOR CHARACTER SETS */
#define CMC_CHAR_CP437                              "1 2 840 113556 3 2 437"
#define CMC_CHAR_CP850                              "1 2 840 113556 3 2 850"
#define CMC_CHAR_CP1252                             "1 2 840 113556 3 2 1252"
#define CMC_CHAR_ISTRING                            "1 2 840 113556 3 2 0"
#define CMC_CHAR_UNICODE                            "1 2 840 113556 3 2 1"

/* RETURN CODE FLAGS */
#define CMC_ERROR_DISPLAYED                         ((CMC_return_code) 0x00008000)
#define CMC_ERROR_RSV_MASK                          ((CMC_return_code) 0x0000FFFF)
#define CMC_ERROR_IMPL_MASK                         ((CMC_return_code) 0xFFFF0000)
```

```
/* RETURN CODES */
#define CMC_SUCCESS                          ((CMC_return_code) 0)
#define CMC_E_AMBIGUOUS_RECIPIENT            ((CMC_return_code) 1)
#define CMC_E_ATTACHMENT_NOT_FOUND           ((CMC_return_code) 2)
#define CMC_E_ATTACHMENT_OPEN_FAILURE        ((CMC_return_code) 3)
#define CMC_E_ATTACHMENT_READ_FAILURE        ((CMC_return_code) 4)
#define CMC_E_ATTACHMENT_WRITE_FAILURE       ((CMC_return_code) 5)
#define CMC_E_COUNTED_STRING_UNSUPPORTED     ((CMC_return_code) 6)
#define CMC_E_DISK_FULL                      ((CMC_return_code) 7)
#define CMC_E_FAILURE                        ((CMC_return_code) 8)
#define CMC_E_INSUFFICIENT_MEMORY            ((CMC_return_code) 9)
#define CMC_E_INVALID_CONFIGURATION          ((CMC_return_code) 10)
#define CMC_E_INVALID_ENUM                   ((CMC_return_code) 11)
#define CMC_E_INVALID_FLAG                   ((CMC_return_code) 12)
#define CMC_E_INVALID_MEMORY                 ((CMC_return_code) 13)
#define CMC_E_INVALID_MESSAGE_PARAMETER      ((CMC_return_code) 14)
#define CMC_E_INVALID_MESSAGE_REFERENCE      ((CMC_return_code) 15)
#define CMC_E_INVALID_PARAMETER              ((CMC_return_code) 16)
#define CMC_E_INVALID_SESSION_ID             ((CMC_return_code) 17)
#define CMC_E_INVALID_UI_ID                  ((CMC_return_code) 18)
#define CMC_E_LOGON_FAILURE                  ((CMC_return_code) 19)
#define CMC_E_MESSAGE_IN_USE                 ((CMC_return_code) 20)
#define CMC_E_NOT_SUPPORTED                  ((CMC_return_code) 21)
#define CMC_E_PASSWORD_REQUIRED              ((CMC_return_code) 22)
#define CMC_E_RECIPIENT_NOT_FOUND            ((CMC_return_code) 23)
#define CMC_E_SERVICE_UNAVAILABLE            ((CMC_return_code) 24)
#define CMC_E_TEXT_TOO_LARGE                 ((CMC_return_code) 25)
#define CMC_E_TOO_MANY_FILES                 ((CMC_return_code) 26)
#define CMC_E_TOO_MANY_RECIPIENTS            ((CMC_return_code) 27)
#define CMC_E_UNABLE_TO_NOT_MARK_AS_READ     ((CMC_return_code) 28)
#define CMC_E_UNRECOGNIZED_MESSAGE_TYPE      ((CMC_return_code) 29)
#define CMC_E_UNSUPPORTED_ACTION             ((CMC_return_code) 30)
#define CMC_E_UNSUPPORTED_CHARACTER_SET      ((CMC_return_code) 31)
#define CMC_E_UNSUPPORTED_DATA_EXT           ((CMC_return_code) 32)
#define CMC_E_UNSUPPORTED_FLAG               ((CMC_return_code) 33)
#define CMC_E_UNSUPPORTED_FUNCTION_EXT       ((CMC_return_code) 34)
#define CMC_E_UNSUPPORTED_VERSION            ((CMC_return_code) 35)
#define CMC_E_USER_CANCEL                    ((CMC_return_code) 36)
#define CMC_E_USER_NOT_LOGGED_ON             ((CMC_return_code) 37)
#define CMC_E_INVALID_OBJECT_HANDLE          ((CMC_return_code) 38)
#define CMC_E_PROPERTY_ID_NOT_FOUND          ((CMC_return_code) 39)
#define CMC_E_INVALID_CURSOR_HANDLE          ((CMC_return_code) 40)
#define CMC_E_REQUIRED_PROPS_MISSING         ((CMC_return_code) 41)
#define CMC_E_INVALID_SOURCE_OBJECT          ((CMC_return_code) 42)
#define CMC_E_INVALID_CONTAINER_OBJECT       ((CMC_return_code) 43)
#define CMC_E_UNRECOGNIZED_IDENTIFIER        ((CMC_return_code) 44)
#define CMC_E_INVALID_PROPERTY_NAME          ((CMC_return_code) 45)
#define CMC_E_INVALID_RESTRICTION            ((CMC_return_code) 46)
#define CMC_E_UNSUPPORTED_KEYS               ((CMC_return_code) 47)
#define CMC_E_INVALID_STREAM_HANDLE          ((CMC_return_code) 48)
#define CMC_E_INVALID_FILE_OFFSET            ((CMC_return_code) 49)
#define CMC_E_INVALID_PROPERTY_ID            ((CMC_return_code) 50)
#define CMC_E_NO_MORE_BYTES_TO_WRITE         ((CMC_return_code) 51)
#define CMC_E_NAME_NOT_FOUND                 ((CMC_return_code) 52)
#define CMC_E_ID_NOT_FOUND                   ((CMC_return_code) 53)
#define CMC_E_TOO_MANY_CONTENT_ITEMS         ((CMC_return_code) 54)
#define CMC_E_BIND_FAILURE                   ((CMC_return_code) 55)
#define CMC_E_UNBIND_FAILURE                 ((CMC_return_code) 56)
#define CMC_E_INVALID_EVENT                  ((CMC_return_code) 57)
#define CMC_E_CALLBACK_NOT_SUPPORTED         ((CMC_return_code) 58)
#define CMC_E_ACCESS_DENIED                  ((CMC_return_code) 59)
#define CMC_E_INVALID_FILE_SPECIFICATION     ((CMC_return_code) 60)
#define CMC_E_PROPERTY_NAME_NOT_FOUND        ((CMC_return_code) 61)
#define CMC_E_INVALID_FUNCTION_EXT           ((CMC_return_code) 62)
#define CMC_E_FUNCTION_INTERRUPTED           ((CMC_return_code) 63)

#ifdef __cplusplus
}       /* extern "C" */
#endif

#endif  /* _XCMC_H */
```

Annex B

CMC vendor extensions

B.1 CMC vendor extensions

This Recommendation enables vendor extensions in many areas. Vendors may add extensions to certain CMC data structures and every CMC function contains a parameter to carry functional extensions. Vendors may define new CMC object classes, extend the set of properties associated with an object class, add additional enumerated values, and associate a CMC implementation identifier with an implementation. In addition, some of the functionality of this Recommendation has been defined using common extensions defined in this Recommendation to preserve backwards compatibility with XAPIA's CMC-1.0. Further extension sets may also be defined by future versions of this Recommendation. 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) Extensions 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:

CMC_XS_[vendor id]	for the extension set identifier
CMC_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 CMC_XS_CX          0x00000400
   #define CMC_X_CX_EXT1      0x00000401
   #define CMC_X_CX_EXT2      0x00000402
   ```

3) Extension sets defined by this Recommendation 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.400 API Association by requesting such a number in writing. Pre-defined extension set numbers are given in Annex D. Support for different extension sets is indicated through the configuration of the CMC implementation and can be queried through the function **cmc_query_configuration()** using the CMC_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 implementation. No registration of an 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 CMC implementations. The prefix for these extensions will be CMC_X_BLT_ and the corresponding set identifier is CMC_XS_BLT.

5) Many objects are named using globally unique identifiers or GUIDs. GUIDs may be assigned by vendors under vendor-specific names. With the registration for an extension set, a vendor is also assigned a branch in the GUID name space:

   –//XAPIA/CMC20/OBJECT CLASS/VENDOR [vendor id]//NONSGML [ext. name]//EN for object classes,

   –//XAPIA/CMC20/PROPERTY/VENDOR [vendor id]//NONSGML [ext. name]//EN for property names,

–//XAPIA/CMC20/CONTENT TYPE/VENDOR [vendor id]//NONSGML [ext. name]//EN for content types,

–//XAPIA/CMC20/CHARSET/VENDOR [vendor id]//NONSGML [ext. name]//EN for character sets, and

–//XAPIA/CMC20/ENCODING TYPE/VENDOR [vendor id]//NONSGML [ext. name]//EN for encoding types.

NOTE – The specification of vendor extensions does not imply that the extensions will be carried unchanged through messaging protocols and gateways. Details of protocol and gateway limitations associated with these extensions should be specified in vendor manuals.

6) Vendors may also extend enumerated values to this Recommendation. The enumerated values from 0 to 512 are reserved for this Recommendation. The vendor may reuse item code values for enumerated values. For the definition of constants associated with these values, the vendor should use the prefix CMC_X_[vendor id]_[enum], ensuring that the constants do not conflict with extension names.

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

B.1.1 Function extensions

B.1.1.1 CMC_X_COM_SUPPORT_EXT

This extension is used by client applications to query the CMC 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 **cmc_logon()**, this extension will also indicate which data extensions the client wants added to the data structures for the session.

NOTE – 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 CMC implementation, this extension must be supported.

USED BY

```
cmc_query_config()
cmc_logon()
```

INPUT

extension_flags

All CMC 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 below:

```
typedef struct {
   CMC_uint32            item_code;
   CMC_flags             flags;
} CMC_X_COM_support;
```

The item_code in the structure is set to the item code of the extensions 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:

CMC_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 CMC_X_COM_SUP_EXCLUDE was set on input. The possible values are listed below.

CMC_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.

CMC_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.

CMC_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 CMC_X_COM_SUPPORTED, if this is returned on the **cmc_logon()** call, the data extensions will be included with the data structures for this session.

CMC_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 CMC_X_COM_SUPPORTED, if this is returned on the **cmc_logon()** call, the data extensions available will NOT be included with the data structures for this session and will need to be requested explicitly.

B.1.1.2 CMC_X_UI_ID_EXT

This extension is used by client applications to specify platform-specific user interface information to the CMC functions. The user interface information may be used by the CMC implementation to present user dialogues for resolving additional arguments to the CMC 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 – The CMC implementations are not required to provide UI, and providing a user interface for one feature does not necessarily imply that a user interface is available for all features of the CMC.

Error codes generated as a result of the use of this function extension will be returned as error codes through the nominal return code process.

USED BY

All Full CMC functions

INPUT

extension_flags

All CMC flags are valid. Unspecified flags should always be passed as zero (0). Additional flags used by this function include the following:

CMC_X_ERROR_UI_ALLOWED

CMC_X_ERROR_UI_ALLOWED – Set if the function may display UI on encountering recoverable errors. If not set, the function may not display a UI and will return an error code. This flag is valid for all CMC functions that support this extension.

item_data

zero

item_reference

A pointer to an identifier for a User Interface (e.g. dialogue window) for use in resolving any questions which might otherwise result in an error and queries the user for additional information as required.

OUTPUT

extension_flags

unchanged

item_data

unchanged

item_reference

unchanged

B.1.1.3 CMC_X_COM_CONFIG_DATA

Get all values available with **cmc_query_configuration()** in a structure.

USED BY

cmc_query_configuration()

INPUT

extension_flags

All CMC flags are valid. No further flags are defined.

item_data

zero

item_reference

NULL

OUTPUT

extension_flags

CMC_EXT_OUTPUT will be set if a structure is successfully returned.

item_data

unchanged

item_reference

Pointer to a structure containing all the information available from the query configuration call. The C declaration for this structure is below:

```
typedef struct {
   CMC_uint16             ver_spec;
   CMC_uint16             ver_implem;
   CMC_object_identifier  *character_set;
   CMC_enum               line_term;
   CMC_string             default_service;
   CMC_string             default_user;
```

```
        CMC_enum               req_password;
        CMC_enum               req_service;
        CMC_enum               req_user;
        CMC_boolean            ui_avail;
        CMC_boolean            sup_nomkmsgread;
        CMC_boolean            sup_counted_str;
    } CMC_X_COM_configuration;
```

The definition for each of the structure members corresponds to the data returned via the reference argument by **cmc_query_configuration()** for the similarly named value of the item argument. This structure should be freed with one call to **cmc_free()**.

B.1.1.4 CMC_X_COM_PROPERTY_HINTS

This function extension provides **cmc_list_objects()** with a hint as to what properties the caller will need in the near future. This hint allows implementations to optimize the retrieval of properties by getting all of the hinted at properties at one time.

USED BY

cmc_list_objects()

INPUT

extension_flags

All CMC flags are valid. Unspecified flags should always be passed as zero (0). No additional flags are defined.

item_data

The number of CMC property names in the array of structures pointed to by item_reference.

item_reference

A pointer to an array of CMC property names. These ids specify the properties that are being hinted at.

OUTPUT

extension_flags

unchanged

item_data

unchanged

item_reference

unchanged

B.1.1.5 CMC_X_COM_CAN_SEND_RECIP

Check if the message service is ready to send to the specified recipient.

USED BY

cmc_look_up()

INPUT

extension_flags

All CMC flags are valid. No further flags are defined.

item_data

zero

item_reference

NULL

On input, the **cmc_look_up()** recipient_in parameter will contain the recipient to query the service about. The extension will only look at the first recipient, if there is more than one passed.

OUTPUT

extension_flags

unchanged

item_data

Set to CMC_X_COM_NOT_READY if a transport is not available for this recipient type, CMC_X_COM_READY if the recipient can be sent to immediately, and CMC_X_COM_DEFER if the message will be accepted but deferred until a transport is ready.

item_reference

unchanged

B.1.1.6 CMC_X_COM_SAVE_MESSAGE

Save a message structure to the inbox.

USED BY

cmc_act_on()

INPUT

extension_flags

Must contain CMC_EXT_REQUIRED to indicate that the save action rather than the delete action should be carried out. All CMC flags are valid. No further flags are defined.

item_data

zero

item_reference

Pointer to message structure to save in the inbox. This message will have the CMC_MSG_UNSENT flag set by the CMC implementation to indicate that it has not been sent.

On input, the **cmc_act_on()** operation parameter must be set to CMC_ACT_ON_EXTENDED to indicate that the operation is contained in the extensions.

OUTPUT

extension_flags

CMC_EXT_OUTPUT will be set if a message is successfully saved and the message reference returned.

item_data

unchanged

item_reference

Pointer to the message reference referring to the message saved to the inbox. This pointer must be freed by **cmc_free()**.

B.1.1.7 CMC_X_COM_SENT_MESSAGE

Return a message structure containing all the information for the message just sent. This is useful to obtain information in the message structure set with UI rather than by the calling application.

USED BY

cmc_send()

INPUT

extension_flags

All CMC flags are valid. No further flags are defined.

item_data

zero

item_reference

NULL

OUTPUT

extension_flags

CMC_EXT_OUTPUT will be set if the item_reference contains a pointer to a message.

item_data

unchanged

item_reference

Pointer to a message structure containing all the information for the message just sent. This pointer should be freed with **cmc_free()**.

B.1.1.8 CMC_X_COM_PROP_STATUS

This function extension indicates that the operation performed should return per-property status. An error resulting from an attempted property modification or deletion is called a property problem. If an operation that affects multiple properties encounters problems that prevent it from processing some of these properties, this extension allows the caller to receive reports about the property problems.

USED BY

cmc_add_properties()

cmc_delete_properties()

INPUT

extension_flags

All CMC flags are valid. Unspecified flags should always be passed as zero (0). No additional flags are defined.

item_data

zero

item_reference

NULL

OUTPUT

extension_flags

The CMC_EXT_OUTPUT flag is set if any property problem information is reported.

item_data

Count of items in the array pointed to by item_reference. Zero if no property problems are reported.

item_reference

Pointer to an array of structures listing the property problems reported. The C declaration for the structure is:

```
typedef struct {
   CMC_uint32            index;
   CMC_id                id;
   CMC-return_code       error_code;
} CMC_X_COM_prop_problem;
```

where:

- index specifies the index of the involved property in the input *properties* or *property_ids* array of the function;
- id specifies the involved property;
- error_code specifies the error encountered when processing the request for that property.

The array is allocated by the service and should be freed with a call to **cmc_free()**.

When this extension reports property problems, the function returns the error code CMC_E_PROPERTY_PROBLEMS. In this case, any property that is not mentioned as reporting a problem can be assumed to have been processed successfully.

ERRORS

CMC_E_DISK_FULL
CMC_E_FAILURE
CMC_E_INSUFFICIENT_MEMORY
CMC_E_INVALID_ENUM
CMC_E_INVALID_MEMORY
CMC_E_REQUIRED_PROPS_MISSING
CMC_E_SERVICE_UNAVAILABLE
CMC_E_TEXT_TOO_LARGE
CMC_E_UNRECOGNIZED_MESSAGE_TYPE
CMC_E_UNSUPPORTED_ACTION
CMC_E_UNSUPPORTED_CHARACTER_SET
CMC_E_UNSUPPORTED_FLAG

B.1.2 Data extensions

B.1.2.1 CMC_X_COM_TIME_RECEIVED

Data extension for a time structure for the delivery time of the message.

At logon, the item code is passed in the CMC_X_COM_SUPPORT_EXT array to indicate that this data member should be attached to the message and message summary structures during the session.

USED BY

CMC_message
CMC_message_summary

INPUT

This extension is ignored on input of message structure.

OUTPUT

extension_flags

NULL

item_data

zero

item_reference

Pointer to a time structure indicating the receive time for the message. See the CMC_time structure for more information.

B.1.2.2 CMC_X_COM_RECIP_ID

A data extension to add a unique opaque recipient identifier to the recipient structure. This is provided by the implementation during recipient name resolution and can be used to avoid further name resolution during send in some services. This is analogous to the message reference.

At logon, the item code is passed in the CMC_X_COM_SUPPORT_EXT array to indicate that this data member should be attached to the recipient structure during the session.

USED BY

CMC_recipient

INPUT

extension_flags

All CMC flags are valid. No further flags are defined.

item_data

length of the recipient id

item_reference

pointer to the recipient id

OUTPUT

extension_flags

unchanged

item_data

length of the recipient id

item_reference

pointer to the recipient id

B.1.2.3 CMC_X_COM_ATTACH_CHARPOS

Data extension to support display of a graphic representation of the attachment in the message text note. The extension holds the character position for the representation.

At logon, the item code is passed in the CMC_X_COM_SUPPORT_EXT array to indicate that this data member should be attached to the attachment structure during the session.

USED BY

CMC_attachment

INPUT

extension_flags

All CMC flags are valid. No further flags are defined.

item_data

Zero-based character offset of the attachment within the text_note data.

NOTE – This is a character offset rather than a byte offset, which is an important distinction when multi-byte character sets are in use.

item_reference

NULL

OUTPUT

extension_flags

unchanged

item_data

Zero-based character offset of the attachment within the text_note data.

item_reference

unchanged

B.1.2.4 CMC_X_COM_PRIORITY

Data extension for message priority.

At logon, the item code is passed in the CMC_X_COM_SUPPORT_EXT array to indicate that this data member should be attached to the message structure during the session.

USED BY

CMC_message
CMC_message_summary

INPUT

extension_flags

All CMC flags are valid. No further flags are defined.

item_data

Set to CMC_X_COM_URGENT, CMC_X_COM_NORMAL, or CMC_X_COM_LOW, depending on the urgency of the message.

item_reference

NULL

OUTPUT

extension_flags

unchanged

item_data

Set to CMC_X_COM_URGENT, CMC_X_COM_NORMAL, or CMC_X_COM_LOW, depending on the urgency of the message.

item_reference

unchanged

B.2 Extension set C declaration summary

This subclause lists the declarations that define the CMC interface for the common extensions set in the C programming language.

The declarations assembled here constitute the contents of a header file to be made accessible to application programmers. They are included in the header file **<xcmcext.h>**. The symbols the declarations define are the only symbols the service makes visible to the application.

```
/* COMMON EXTENSIONS DECLARATIONS */

/* EXTENSION SET ID */

#define CMC_XS_COM                   ((CMC_uint32) 0)

/* FUNCTION EXTENSIONS */

/* Query for extension support in implementation */
```

```
#define CMC_X_COM_SUPPORT_EXT        ((CMC_uint32) 16)

typedef struct {
             CMC_uint32         item_code;
             CMC_flags          flags;
          } CMC_X_COM_support;

#define CMC_X_COM_SUPPORTED          ((CMC_flags) 1)
#define CMC_X_COM_NOT_SUPPORTED      ((CMC_flags) 2)
#define CMC_X_COM_DATA_EXT_SUPPORTED ((CMC_flags) 4)
#define CMC_X_COM_FUNC_EXT_SUPPORTED ((CMC_flags) 8)
#define CMC_X_COM_SUP_EXCLUDE        ((CMC_flags) 16)

/* Get back a structure with configuration data */

#define CMC_X_COM_CONFIG_DATA        ((CMC_uint32) 17)

typedef struct {
             CMC_uint16             ver_spec;
             CMC_uint16             ver_implem;
             CMC_object_identifier  character_set;
             CMC_enum               line_term;
             CMC_string             default_service;
             CMC_string             default_user;
             CMC_enum               req_password;
             CMC_enum               req_service;
             CMC_enum               req_user;
             CMC_boolean            ui_avail;
             CMC_boolean            sup_nomkmsgread;
             CMC_boolean            sup_counted_str;
       } CMC_X_COM_configuration;

/* Check to see if a recipient can be sent */

#define CMC_X_COM_CAN_SEND_RECIP     ((CMC_uint32) 18)

#define CMC_X_COM_READY              ((CMC_enum) 0)
#define CMC_X_COM_NOT_READY          ((CMC_enum) 1)
#define CMC_X_COM_DEFER              ((CMC_enum) 2)

/* Save a message to the inbox */

#define CMC_X_COM_SAVE_MESSAGE       ((CMC_uint32) 19)

/* Get back a message structure for the message just sent */

#define CMC_X_COM_SENT_MESSAGE       ((CMC_uint32) 20)

/* DATA EXTENSIONS */

/* attach received data to message and message summary structures */

#define CMC_X_COM_TIME_RECEIVED      ((CMC_uint32) 128)

/* attach a unique id to resolved recipient structures */

#define CMC_X_COM_RECIP_ID           ((CMC_uint32) 129)

/* set character position in the message text to display an icon
  associated with a particular attachment */

#define CMC_X_COM_ATTACH_CHARPOS     ((CMC_uint32) 130)

#define CMC_X_COM_PRIORITY           ((CMC_uint32) 131)

#define CMC_X_COM_NORMAL             ((CMC_enum) 0)
#define CMC_X_COM_LOW                ((CMC_enum) 1)
#define CMC_X_COM_URGENT             ((CMC_enum) 2)
```

B.2.1 X.400 extension set

The following extension set identifiers are being registered with XAPIA for X.400 usage:

```
#define CMC_XS_X400                  ((CMC_uint32) 0x00000600)
#define CMC_X_X400_ERROR             ((CMC_uint32) 0x00000601)
#define CMC_X_X400_MSG_PARENT        ((CMC_uint32) 0x00000602)
```

```
#define CMC_X_X400_MSG_ID             ((CMC_uint32) 0x00000603)
#define CMC_X_X400_MSG_REPORT_ID      ((CMC_uint32) 0x00000604)
#define CMC_X_X400_REPORT             ((CMC_uint32) 0x00000605)
```

B.2.1.1 CMC_Report structure

The following "C" structure is being used in the CMC_X_X400_REPORT extension:

```
      typedef struct {
            CMC_recipient      *msg_recipient;
            CMC_enum           report_type;
            CMC_time           delivered_time;
            CMC_uint32         reason_code;
            CMC_flags          report_flags;
      } CMC_report;

/* report_type */

      #define CMC_X400_DR       ((CMC_enum) 0)
      #define CMC_X400_NDR      ((CMC_enum) 1)

/* report_flags */

      #define CMC_REPORT_LAST_ELEMENT ((CMC_flags) 0x80000000)
```

B.2.1.2 Error code: CMC_EX_X400_STD

A new error code is defined to further qualify that a CMC function's failure is due to an X.400 exception/abort/error condition. This code will be used in the higher-order 16 bits of the CMC_return_code to indicate that the error is associated with the Recommendation X.400-X.420 (1988) messaging services.

The "C" definition of this error is:

```
      #define CMC_EX_X400_STD   ((CMC_uint16) 400)
```

Thus, if the CMC implementation wishes to classify an error condition that is caused by the underlying X.400 message service and optional X.400 related CMC_extensions will be returned, the CMC_return_code can be set to the following:

CMC_return_code.<lower order 16 bits> = CMC_E_FAILURE, or the most appropriate error

CMC_return_code.<higher order 16 bits> = CMC_EX_X400_STD

B.2.2 Additional extensions for simple CMC/X400 mapping

B.2.2.1 CMC_X_X400_ERROR

If the CMC function fails because the underlying X.400 operation is not successful, the error resulted from the X.400 operations is returned to the CMC application so that it can find out the root cause of the error. This extension contains specific errors that are defined by the Recommendation X.400-X.420 (1988). See the related document for the explanation and value of the error.

NOTE – If the CMC application wants the CMC implementation to return this extension should an X.400 error occur, the application must supply the storage of this extension when the CMC function is invoked; otherwise, the CMC implementation cannot return this extension because the extension argument of each CMC function is only an address to where the buffer has been allocated by the application. Unless the CMC V1.0 specification is modified to allow the function extension argument to be an input and output argument, the other alternative is that the CMC implementation supplies a new ErrorInfo function for the application to obtain the error detailed in this extension after a CMC function has been failed with an X.400 related error.

USED BY

cmc_act_on(), cmc_list(), cmc_logon(), cmc_logoff(), cmc_read(), cmc_send()

OUTPUT

item_code

CMC_X_X400_ERROR

item_data

item_data.<higher order 16 bits> = X.400 defined operation number

item_data.<lower order 16 bits> = X.400 defined return codes of the operation

item_reference

NULL

extension_flags

All CMC flags are valid. No further flags are defined.

B.2.2.2 CMC_X_X400_MSG_PARENT

X.400 Message Store supports nested messages using the parent and child messages concept. For example, a body part of an IPM that contains a forwarded IPM, the forwarded IPM is a child message and the forwarding IPM is the parent message, or, the content of a report, a returned IPM is the child message and the report itself is the parent message. A new extension will be used to allow the application to determine whether a message is a parent or a child.

Identification used to indicate whether the message_reference of the CMC_message or CMC_message_summary is an X.413 parent message or child message. If the associated message is a parent message, this extension will not be returned.

USED BY

CMC_message and CMC_message_summary

OUTPUT

item_code

CMC_X_X400_MSG_PARENT

item_data

X.413.parent-sequence-number for child message

item_reference

NULL

extension_flags

All CMC flags are valid. No further flags are defined.

B.2.2.3 CMC_X_X400_MSG_ID

When sending a message, X.400 creates a unique identifier for this message, an MTS identifier. This identifier is used for message tracking and to report delivery/non delivery of a message. This identifier is returned to the CMC application via the message ID extension when reading, listing, or sending a message. Thus the application can respond to or reference a particular message with the appropriate action.

A unique identification of a message that is given by the underlying messaging service when the CMC application is either reading, listing, or sending a message.

USED BY

CMC_message, CMC_message_summary, and **cmc_send()**.

Reading and listing a message:

When a message structure (or a message summary structure) is returned to the user after a call to **cmc_read()** (or **cmc_list()**), the message ID extension is attached to the structure. The item_data for the message ID extension is insignificant and is hence 0. The item_reference points to a CMC_string structure allocated by the service and contains a readable format of the unique MTS identifier.

Sending a message:

When the user sends a message using **cmc_send()**, the CMC service has the capability of returning to the caller the MTS identifier allocated for that message in the send extension structure, if the caller allocates memory for the extension template with the item_code of CMC_X_X400_MSG_ID. If this extension is missing, the CMC service will not return the MTS identifier. The CMC service returns the MTS identifier by allocating a CMC_string with the required data and attaching a pointer to this data to the item_reference. The extension_flags are set with CMC_EXT_OUTPUT on. This indicates to the caller that item_reference should be freed using **cmc_free()** after making use of it.

OUTPUT

item_code

CMC_X_X400_MSG_ID

item_data

NULL

item_reference

pointer to CMC_string of MTS identifier

extension_flags

All CMC flags are valid. No further flags are defined.

B.2.2.4 CMC_X_X400_MSG_REPORT_ID

When reading a delivery or non-delivery report, the underlying messaging service returns a unique identifier for the report (MTS identifier); this is different from the original message that the report is about.

A unique identifier of a delivery or non-delivery report that is given by the underlying messaging service when that report is being read by the application.

USED BY

CMC_message

OUTPUT

item_code

CMC_X_X400_MSG_REPORT_ID

item_data

NULL

item_reference

pointer to cmc_string of a readable format of the unique MTS identifier

extension_flags

All CMC flags are valid. No further flags are defined.

B.2.2.5 CMC_X_X400_REPORT

This is used to convey the specific X.400 delivery or non-delivery information to the CMC application when the information base to be returned is an X.400 report. This extension is returned as the message_extension of the CMC_message.

Return of specific delivery or non-delivery report information that is defined by Recommendation X.411 (1988). See the related document for the explanation and value of the reason codes and diagnostic codes.

USED BY

CMC_message

OUTPUT

item_code

CMC_X_X400_REPORT

item_data

NULL

item_reference

pointer to the CMC_report structure

extension_flags

All CMC flags are valid. No further flags are defined.

B.2.3 Other extension sets

Other extension sets will be defined by the XAPIA and by vendor groups to support various messaging protocols. Currently extension sets are being defined for use with G3 facsimile, G3-64 facsimile, G4 facsimile, telex and Teletex service via Recommendation T.611. To find out what extension sets are available, contact the XAPIA.

B.2.4 Platform-specific information including run-time bindings

CMC implementors are encouraged to provide run-time binding interfaces to their CMC service implementations. In general, these interfaces are platform- and/or operating system-dependent. This subclause 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:

```
byte                       CMC_sint8
16 bit int                 CMC_sint16
32 bit long int            CMC_sint32
16 bit unsigned int        CMC_uint16
32 bit unsigned long int   CMC_uint32
32 bit pointer             CMC_buffer
32 bit char pointer        CMC_string
CMC_uint32                 CMC_ui_id
CMC_uint32                 CMC_session_id
```

B.2.4.1 Explicit and implicit binding

All functions in the CMC API should be linkable implicitly and explicitly. Implicit linking builds the linkage of the application and the CMC service implementation into the application. Explicit linking requires the application to contain run-time code that links a CMC service implementation.

It is also recommended that all extension functions be loaded explicitly, since their absence on some CMC implementations would otherwise prevent the application from loading.

Static and dynamic linking mechanisms are defined for several common platforms below.

B.2.4.2 Apple Macintosh binding

For static linking, applications should use the Pascal calling convention and 32-bit flat pointers to call an Apple Macintosh CMC implementation.

For dynamic linking, contact Apple Computer, Inc.

The CMC implementation should always attempt to provide Apple International Strings (ISTRING).

B.2.4.3 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 CMC implementation. This is compatible with the Microsoft C "large" memory model. Any future changes to this mechanism will be published by Microsoft.

The CMC implementation should always attempt to provide code page 437 or 850.

B.2.4.4 MS-Windows 3.x binding

For dynamic linking, MS-Windows 3.x CMC implementations should use Dynamic Linked Libraries and link by name to the CMC functions.

At run-time, to determine if a CMC service is available, applications should call GetProfileInt() to look for the CMC variable in the [MAIL] clause of WIN.INI. If this variable is present and non-zero, it indicates that a CMC.DLL library is available. If the CMC variable is not found or is zero, then the functions cannot be called. Any future changes to this mechanism will be published by Microsoft.

CMC functions should be called "far", using the Pascal calling convention, and 32-bit segmented "far" pointers.

CMC 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 CMC implementation should always attempt to provide code page 1252.

B.2.4.5 MS-Windows NT binding

For dynamic linking, MS-Windows NT CMC implementations should use Dynamic Linked Libraries and link by name to the CMC functions.

At run-time, to determine if a CMC service is available, applications should query the registry to see if CMC is available. The exact mechanism for this will be published by Microsoft.

CMC functions should be called using the STDCALL calling convention.

B.2.4.6 OS/2 1.x and 2.x 16-bit DLL binding

For dynamic linking, OS/2 1.x and 2.x 16-bit CMC implementations should use Dynamic Linked Libraries and link by name to the functions.

At run-time, to determine if a CMC service is available, applications should call WinQueryProfileInt() to look for the CMC variable in the [MAIL] clause 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 CMC.DLL library is available. If the CMC variable is not found or is zero, then the functions cannot be called. Any future changes to this mechanism will be published by IBM.

CMC functions should be called "far", using the System calling convention, and 32-bit segmented "far" pointers.

The CMC implementation should always attempt to provide code page 850.

B.2.4.7 OS/2 2.0 32-bit DLL binding

For dynamic linking, OS/2 2.0 32-bit CMC implementations should use Dynamic Linked Libraries and link by name to the functions.

At run-time, to determine if a CMC service is available, applications should call WinQueryProfileInt() to look for the CMC variable in the [MAIL] clause 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 CMC.DLL library is available. If the CMC variable is not found or is zero, then the functions cannot be called. Any future changes to this mechanism will be published by IBM.

CMC functions should be called "far", using the System calling convention, and 32-bit flat "far" pointers.

The CMC implementation should always attempt to provide code page 850.

B.2.4.8 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 CMC service is available, applications should look for the CMC implementation on the absolute path /usr/lib/XAPI/libCMC.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.

CMC functions and structures should use the System calling convention.

The CMC implementation should always attempt to provide code page 850.

B.2.5 Simple CMC usage of X.400 backbone services

This subclause describes how Common Messaging Call (CMC) API version 1.0 functions are mapped to an underlying X.400 message handling system on the Message Store (MS) boundary, and how CMC messages are mapped to the X.400 messages. This Recommendation does not address the following:

X.500 directory (address) mapping, which can be accessed via the cmc_look_up.

The User Interface (UI) dialogue, which is an option in the **cmc_send_documents()** function, as this is not basic to the interaction between the CMC (messaging-enabled application) and the X.400 messaging system.

This Recommendation assumes the reader is familiar with the Remote Operation Service Element (ROSE) and P1, P2, P22, P3, and P7 protocols and service elements, as well as the specifications and objectives of the CMC API version 1.0. The following Recommendations are referenced:

– Recommendations X.200-X.219 (OSI Model and Notation, Service Definition);

– Recommendations X.220-X.229 (OSI Protocol Specifications);

– Recommendations X.400-X.420 (1984 X.400 MHS);

– Recommendations X.400-X.420 (1988 X.400 MHS);

– Recommendation F.401 (1988), Annex B (Representation of O/R Addresses for human usage) or its aligned equivalent;

– ISO/IEC 10021-2:1990/Amd.1, Annex F (Representation of O/R Addresses for Human usage);

– XAPIA CMC API version 1.0;

– XAPIA CMC API version 2.0.

The mapping between CMC version 1.0 and X.400 described in this Recommendation is done with two objectives:

- In line with the simple and high level objectives of CMC 1.0, the mapping does not utilize the full set of the X.400 features and so only a basic profile is recommended.

- Balance the major concern of interoperability between different CMC version 1.0 implementations using any of a variety of X.400 messaging systems as a message transport.

B.2.5.1 Introduction

The Common Messaging Call Application Program Interface (CMC API) provides a set of high-level functions for messaging-enabled applications to send and receive electronic messages. This interface requires support by messaging services. A major messaging service is OSI’s X.400 Message Handling System (MHS). This Recommendation is directed to those who wish to integrate the CMC API with the X.400 MHS.

For each CMC implementation, the view and capabilities presented by CMC must be mapped to the view and capabilities of the underlying messaging service. To maximize interoperability between CMC applications that use different underlying messaging services, XAPIA offers several guidelines. Message strings are to be mapped to international character sets wherever possible and message attachment types are to be mapped to commonly recognized attachment types wherever appropriate or possible.

To achieve this mapping, the characteristics of the underlying message service (MHS) must be understood and used in the most appropriate manner. The rest of this Recommendation provides the following discussion:

- A high level overview of Recommendation X.400.

- A general discussion in using simple CMC API over the X.400 messaging service, such as options, considerations, and possible extensions to support a richer set of functionality for messaging-enabled applications and messaging-reliant applications.

- A basic mapping profile for simple CMC API that gives simple interoperability for sending and receiving messages across a variety of X.400 MHS communication services.

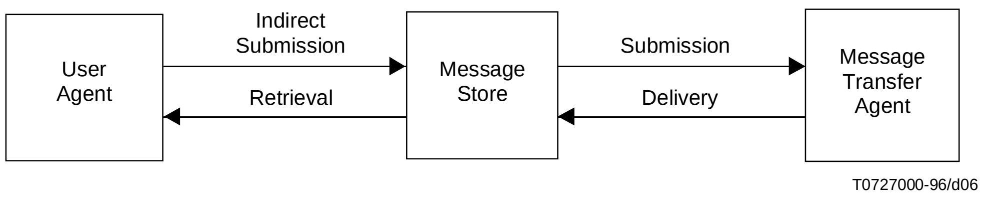
B.2.5.2 X.400 high-level overview

The message handling services provided by X.400 Message Handling System include an Interpersonal Messaging (IPM) service and a message transfer service. These services enable subscribers to exchange messages on a store-and-forward basis. The Message Handling Service defines a set of message types and capabilities that an originator can send to recipients.

An originator prepares a message with the assistance of a User Agent (UA). The User Agent is an application that interacts with the Message Transfer System (MTS) to submit messages. The Message Transfer System (MTS) consists of a number of Message Transfer Agents (MTAs). Operating together, these MTAs relay the message to the intended recipient User Agents that then make the message available to the intended recipients.

The 1988 version of X.400 included an optional Message Store (MS). A user can submit messages through the Message Store and receive messages that have been delivered to that Message Store. The Message Store acts only on behalf of individual users.

Submission and Delivery with a Message Store



The operations between an MS and the MTS correspond to the P3 protocol. The operations between a UA and the MS correspond to the P7 protocol. The P7 operations are:

- Retrieval service (Summarize, List, Fetch, Delete, Register-MS with a possible asynchronous signal, and Alert).
- Indirect-submission uses the submission services of X.411 (message-submission, probe-submission, cancel-deferred-delivery and submission-control).
- Administration services (register and change credentials).
- To connect and disconnect the MS services, the MS-Bind and MS-Unbind operations are used. The bind operation is used to identify, authenticate, and set the security context for an MS service user.

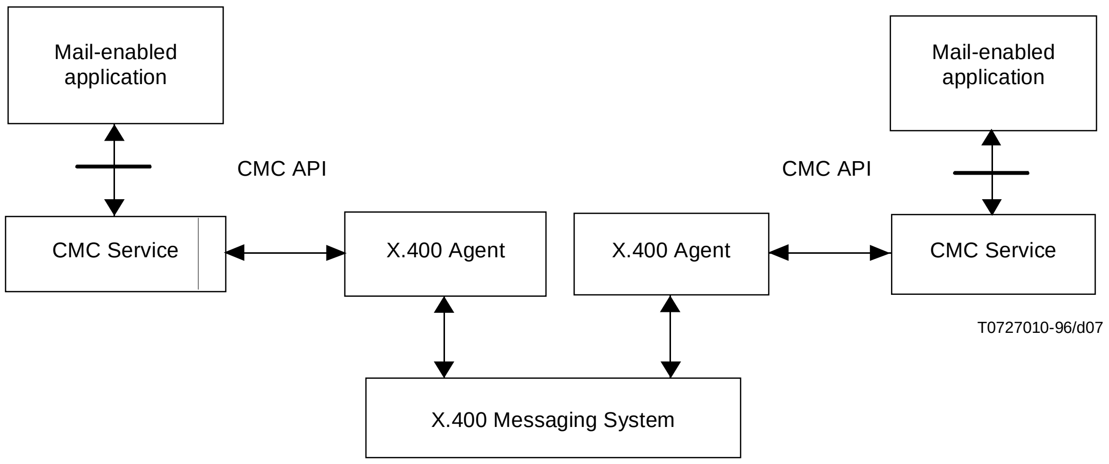
The P7 operations are invoked by the UA using the Remote Operation Service Element (ROSE, defined in Recommendations X.219 and X.229). The ROSE model consists of request and reply interaction. This allows the UA application to request a P7 operation and obtain the result of that P7 operation.

B.2.5.3 General approach and considerations

This subclause presents a generic view of mapping CMC v1.0 (also known as Simple CMC v2.0) to X.400 MHS and some of the possible considerations and options. The discussion does not address the human interface because the messaging-enabled applications are viewed as being capable of running without human interaction although they could be run by user commands or scripts. This means that the applications do not require a graphical user interface and can be run as background processes.

B.2.5.3.1 CMC functions and X.400 MS operations

According to the functional models of CMC v1.0 and X.400 MS, the CMC functions map reasonably well to the MS services. The CMC Logon and Logoff map to the MS Bind and Unbind. The CMC Send maps to the MS indirect submission. The CMC Read maps to MS Fetch. The CMC Act On and CMC List are covered by the MS Delete, Summarize and List.



Application to Application CMC messaging service

The two purposes of the CMC API are to provide a generic set of messaging capabilities that are independent of any operating system and to provide a minimum number of function calls needed to send or receive a message. Minimal function calls and interoperability are key requirements. When mapping the CMC functions to the MS operations and to implement these requirements, two approaches to a simple function call API can be used below. The choice between these two styles determines the way the CMC calls are implemented:

– simple CMC call functions and lots of special extensions for the local environment;

– CMC call functions that hide internal complexity and support generic extensions.

B.2.5.3.2 CMC messages and X.400 messages

Much of the work in translating CMC messages to and from X.400 messages involves conversion between the CMC message structures and X.400 message structures. Thus, the names and addresses of the CMC messaging service users need conversion to X.400 user (originator and recipient) names and addresses that are known as O/R names (which include O/R addresses).

Other parts of a CMC message structure that require conversion are the message_type, the time_sent, the recipients and the attachments. A CMC message subject is simply an X.400 IPM subject (with some restrictions on length), while the text_note may require special handling because of differences between Recommendation X.400 and CMC. The CMC flags, extensions, and other original input parameters are basically used just for assisting the conversion. They are not sent as part of the X.400 message so that information is in most cases lost when the delivered X.400 message is converted back to a CMC message for the destination CMC application.

Some conversion, convention, and other choices needed for using X.400 messaging as the underlying message services are:

Text conversions

- Character set conversion;
- Name and address conversions.

Local operating system and messaging system conventions

- Connection/disconnection conventions;
- Underlying messaging system error awareness;
- Message Store conventions and special requirement conventions.

Outbound message conventions

- Native or generic conventions such as text conversion options;
- End destination special conversions.

Inbound message conversions

- Handling of CMC messages that cannot be handled locally;
- Handling of CMC messages that have parts that cannot be handled locally;
- Handling of bad CMC messages;
- Non-CMC message handling.

Extensions both generic and locally special

- Extensions to handle aspects of the underlying system that are not CMC generic;
- Extensions to add special X.400 features such as priority or delivery notification;
- Extensions to use specific X.400 body parts as attachments;
- Extensions that are used to match the sending and destination local systems.

B.2.5.3.3 CMC message attachments and X.400 body parts

The CMC message attachments (text and binary) are equivalent to X.400 body parts. The most appropriate body part for a CMC message attachment is different for each "version" of Recommendation X.400 (i.e. 1984, 1988, or 1992). The recommended equivalents are listed below.

CMC ASCII	<=>	IA5 text	
CMC text_note as a file	<=>	IA5Text Body Part	
CMC text attachment	<=>	1984 X.400	IA5Text Body Part
CMC text attachment	<=>	1988 X.400	Externally Defined Body Part
CMC text attachment	<=>	1992 X.400	File Transfer Body Part
CMC binary attachment	<=>	1984 X.400	Bilaterally Defined Body Part
CMC binary attachment	<=>	1988 X.400	Externally Defined Body Part
CMC binary attachment	<=>	1992 X.400	File Transfer Body Part

B.2.5.3.4 Input/output conventions and requirements

The common almost universal character set used within X.400 is International Alphabet Number 5 (IA5 Text) which is similar to ASCII but not quite. Such ASCII characters as "@", "%", and "_" are not in the basic IA5 but they are in the International Reference Version (IRV). For display and input purposes, the other (national) versions of IA5 require a conversion convention. The NIST OIW has a conversion algorithm for the interchange of IA5 text and ASCII text and it is recommended that this be used if the text that the local CMC messaging-enabled application deals mainly with is an IA5 non-IRV version. A similar convention is needed for other character sets such as EBCDIC, ISO 10646, UNICODE, etc., as well as ways to handle filenames with embedded blanks that have been passed in through CMC Counted Strings.

The X.400 names and addresses are internally arranged in Recommendation X.400 as a structured set of data objects similar to but more complex than the CMC recipient's structure. There is a convention for displaying one's X.400 name and address on name cards, etc. This convention, together with the other standard display conventions in Annex B of Recommendation F.401 (that refers to ISO/IEC 10021-2 Annex F), should be used for CMC text string representation of X.400 recipient addresses. The use of CMC Look Up could provide a simple way to go from a known name such as "eowens" to either a text string (name)address illustrated below or to a CMC recipient structure.

> CMC recipients (originator in CMC_message_summary or recipients in CMC_message structures) take address strings that match those recommended in Recommendation F.401. For example, the CMC_recipient address string with "S=Owens; G=Edward; P=ccmail; A=telemail; C=US"

There will be a need for multiple conventions for name and address representations if a mixture of underlying messaging systems is supported. The previous illustration assumes that there is only one underlying messaging system (X.400). Also, with the names and addresses that are passed through the messaging system, there are conversion requirements when the character sets used in the messaging system do not support the original name and address conventions. If name@address cannot be passed through except by a convention such as name(a)address in IA5 display text, then the reverse convention must be applied at the other end or the convention understood by the recipient. As an example, when the X.400 address involves domain-defined attributes such as a DDA for an Internet address, then the name and address could be:

DDA:RFC-882=fred(a)widget.co.uk;O=gateway;P=abc;C=gb

Word size and/or byte ordering conventions can also create problems so that some attachments are unusable, and numbers are garbled. Normally, the solution to most of these problems is to standardize the contents and formats of what goes on the wire. However, if what goes on the wire is dependent on the underlying message system and there could be an unknown number of these used (although not all of them by any CMC messaging-enabled application), then a different solution is necessary. One simplistic solution is to have the sender messaging-enabled applications be aware of the receiving messaging-enabled application's capabilities and tailor the message accordingly. This simplifies transfers between systems with similar capabilities. An alternative is to send additional information with the message that tells the receiving system about the sender's capabilities and/or the message's particular formatting and other special details.

If special conventions can be set up, then using CMC extensions provides a way of telling both the CMC send implementations how to pass such information through the underlying messaging system to the CMC read implementation such that the messaging-enabled application can choose how to read the message. At this time it is recommended that CMC 1.0 leave this up to local implementation as the issues are best resolved when all the local requirements are understood.

B.2.5.3.5 X.400 connecting/disconnecting and MS requirements

Connecting to or disconnecting from an X.400 underlying service is subject to local variations and also to what service or interface. The present assumption is that the connection is to the X.413 Message Store. This service encompasses the CMC requirements and provides additional functionality. The various features and functionality of X.413 interface require interpretation so that the expected CMC services are provided in an understandable way.

For the X.413 Bind operation to act as a CMC Logon to an X.400 messaging system, then the arguments "user" and "password" are required to be strings containing the messaging service user name and the password that gives the user access to the underlying service. These in X.413 terms are the ORAddressAndOrDirectoryName and InitiatorCredentials. Assuming only simple authentication is required by the X.413 service, then it is recommended that "user" be the Recommendation F.401 display text version of the X.400 user name and address.

For non-simple authentication and other X.413 MS Bind input arguments that the user could require, the special CMC extensions provide a local way to add security context, fetch restrictions and Mail Store configuration requests. Extra CMC extensions may also be necessary to deal locally with the returned result from the MS Bind call or the returned Bind errors. The simple choice is to ignore the MS Bind call result and to have CMC_E_FAILURE returned by the CMC Logon function if the MS Bind returns an error. A different choice is to use an optional CMC extension to report the associated X.400 error.

Incoming X.400 messages are stored in the Message Store and read either by explicitly requesting a particular (known number) mail message or by requesting the next "unread" mail item. The attachments will be returned in either a temporary directory with their attach_filenames as filenames or in temporary named files in a directory with some indication of the sender's title. Those parts of incoming mail messages that cannot be mapped into the CMC message structure will be discarded.

The Message Store must have a P7 interface and implement both X.413 features for making inquires about the relevant contained client mail messages and for delivering requested items and also "X.420" items such as the header and various Body Parts contained in an X.420 Interpersonal Message's contents.

The MS Unbind closes the association between the user (or messaging-enabled application) and has no argument, result or error. Thus a CMC Logoff has no extra complications due to the underlying X.400 messaging service.

Mail-enabled applications (or users) should be aware of a Message Store feature. The Message Store can hold child-entries beside main-entries for the stored messages. These children are listed as well as their parents but the messaging-enabled application (or user) can only delete child-entries by deleting their parent entry. Deleting a child-entry does not work as expected.

B.2.5.4 Message conversion

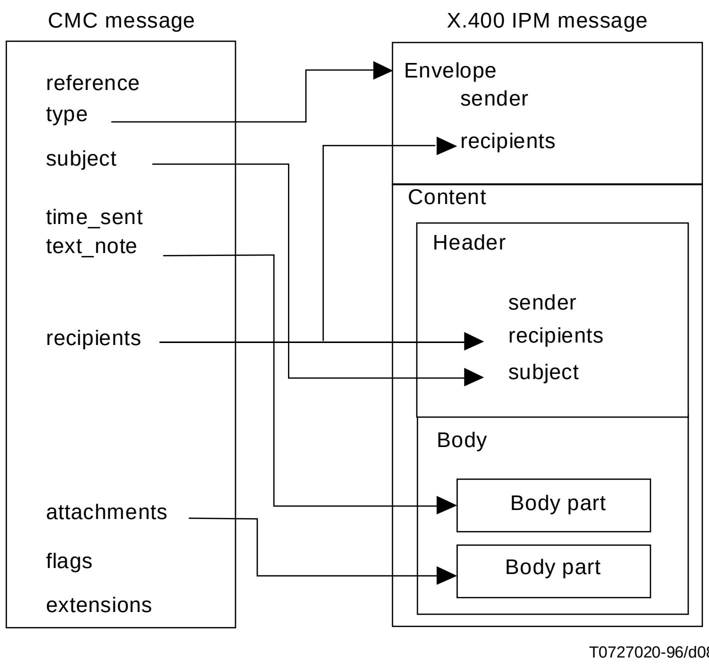
This subclause discusses the conversion of CMC messages to X.400 messages, the conversion of X.400 messages to CMC messages, and the conversion of non-CMC X.400 messages.

B.2.5.4.1 Converting CMC messages to X.400 messages

A CMC message structure includes a message type, a subject, recipient(s) and possibly a note and/or a set of attachments. The X.400 equivalent of this CMC message is a user Message Protocol Data Unit (MPDU). A 1984 MPDU has a basic structure of an envelope and content. Additional services are provided to individuals who want to communicate with others by User Agents (UA). This service is an Interpersonal Messaging System (IPMS) with assigned content type for those messages. IPM UAs communicate with other IPM UAs. The IPM content is divided into header and body. So for conversion purposes the CMC message is split up into envelope, header, and body.

The X.400 envelope contains the sender's identity (name and address), the recipients' identities, and specific X.400 message envelope details. The IPM header contains the identity of the sender, a set of authorizing users, set of recipient identities (primary, copy, and blind copy), a subject and other X.400 specific details. The body is a sequence of body parts. Each body part is one of a set of different types. The most relevant types are IA5Text and Bilaterally Defined Body Part (Type 14). More relevant types have been defined for later versions (1988 and 1992) of Recommendation X.400.

Later body part type additions include the Externally Defined Body Part (1988) and the File Transfer Body Part (1992). These two types are much better for CMC use as more than just the data can be conveyed (and not lost). Thus there is a range of choices for the note and the attachments.



Some of the choices are listed below:

- Convert all CMC messages to X.400 (1984) with text (note and attachment) held in IA5Text body parts and "binary" attachments held in BDBP/Unidentified(Type 14).
- Convert all CMC messages to X.400 (1988) with text_note in IA5Text and attachments held in EDBP(Type 15).
- Convert attachments depending on year (version) of Recommendation X.400.
- Convert according to a special CMC extension that indicates what X.400 body part to use in every case including the use of other body parts.

The later versions of Recommendation X.400 have a greater range and choice of body parts. These later body parts are more closely aligned with the CMC attachment requirements. The 1984 X.400 supported both a text style body part and an octet string style body part. Both IA5Text and Bilaterally Defined Body parts are only able to carry text or "binary" strings but not the extra attachment information of attach_title nor attach_filename. A convention such as using an extra body part for the extra information is possible but not recommended.

The 1988 X.400-Series discourages the use of the Bilaterally Defined body part and recommends the use of the Externally Defined body part. The extra capabilities of the externally defined body part permit the attach_title to be carried with the body part. The 1992 X.400 has defined a File Transfer body part to transfer the contents of a stored file and, optionally, its attributes. The contents portion is like the Externally Defined body part while the optional parameter's portion carries attributes such as the related-stored-file, contents-type, relationships, and file-attributes. Thus the File Transfer body part is ideally suited for carrying CMC attachments.

The 1984 X.400 is the major supported X.400 messaging system; thus, a compromise is needed. A suggested basic profile uses the 1984 X.400 body parts and requires that if the later body parts are used, then they can be downgraded to be equivalent to the 1984 body parts. Also, those X.400 messaging-enabled applications that use the more advanced body parts must be able to accept the basic profile set in their place. This means that attach_titles require an automated substitution at the receiving end because they are not transferred.

X.400 Message Handling Systems require a set of mandatory attributes as part of a message submission. Among these mandatory inputs is that of originator, the ORName of the sender. This is also required for the MS Logon so that if the CMC message's recipients do not include one with a role of CMC_ROLE_ORIGINATOR, then the "user" from the MS Logon parameters shall be used as a default. A recipient name is also required and the message submission should be rejected if one is not supplied.

The X.400 message that is sent as an Interpersonal Message (IPM) must have its content-type set to interpersonal-messaging-198(4 or 8). Other mandatory X.400 mandatory attributes should be supplied either as a default (Priority set as normal) or the PerMessageIndicators set with bits indicating no disclosure of recipients allowed, implicit conversion prohibited, no alternate recipient, and no return of content. Also another mandatory input attribute should request the OriginatorReportRequest bits set to indicate a non-delivery report.

To support the use of the most appropriate body part or to add other X.400 attributes to a message, the use of optional CMC extensions is recommended. They are discussed in the later subclause. However, for initial and basic use of the X.400 messaging service, the basic profile that does not require extra extensions is recommended as this leads to interoperability but not the best interchange between two messaging-enabled systems that support each other's extensions.

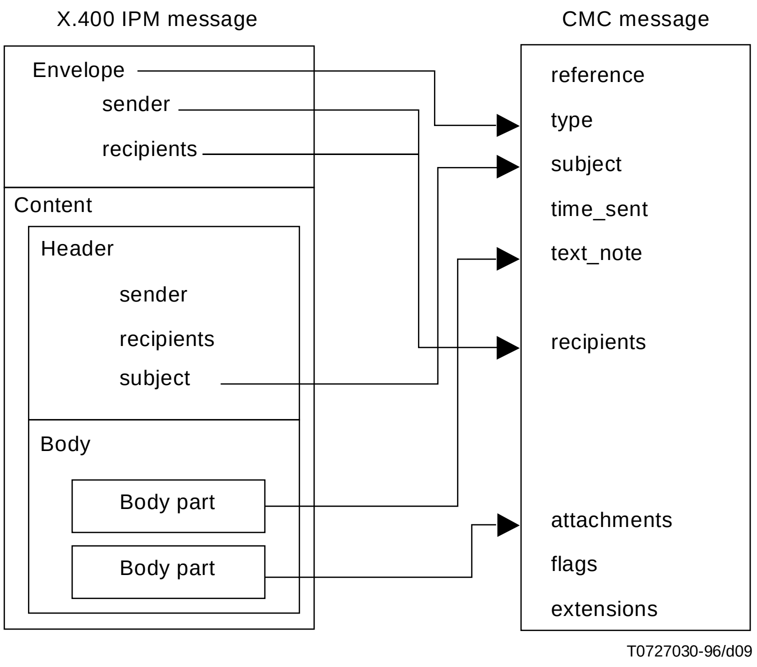
B.2.5.4.2 Converting X.400 messages to CMC messages

X.400 messages come in a variety of types and versions. The major type and size that is appropriate for CMC is the Interpersonal Message (IPM); however, an X.400 Message Store can hold any sort of X.400 message including damaged ones. The major varieties are messages, probes and reports. Within messages are "P1" messages, Interpersonal Messages (IPMs) and other "P2" messages such as X.435 EDI messages. The IPMs include notifications (IPNs) and versions based on 1984, 1988, and 1992 standards.

Several simple choices are easily made:

- The CMC user sees both "CMC" and other X.400 messages held in the Message Store.
- The CMC user can also partially read the other X.400 messages.
- The CMC user can delete those other messages (except when they are children of others).

The conversion of CMC messages into X.400 messages has been described in the previous subclause. Thus the conversion back to CMC messages is fairly straightforward apart from any information that is lost in the process. For example, the 1984 X.400 body parts used to carry attachments cannot carry the attach_name also so that is lost.



The reports that are generated by delivery or non-delivery of a "CMC" X.400 message appear as reports in the sender's Message Store. At the request of the originator, the report will contain the returned IPM that is non-deliverable. It is also possible that an X.400 report will contain a delivery report for some recipients and non-delivery report for other recipients depending on the point of failure. From the MS point of view, both the DRs and NDRs are part of the report that is quite different from the point of view of CMC. These need to be converted in a suitable way so that a CMC user can correlate that report with the message that was sent.

Other X.400 messages that are in the Message Store might be "CMC" messages sent by other CMC users that have a different method of converting their CMC messages to X.400. Thus a whole range of possible conversions needs to be handled. Whatever method chosen should allow the receiving user to choose to either ignore the conversion, accept part of it, or accept all of it, as there are other facilities for conversion available.

There are additional problems surrounding the downgrading and upgrading or other X.400 conversions of X.400 messages. All this means that either there are a few straightforward imports from a "non-CMC" X.400 message into a CMC message that would enable a CMC user to "see" that message or a complex set of rules. For simple conversions, IA5Text body parts are either notes or text attachments. All other body parts could be classed as "binary" and the CMC user left with "type" and perhaps "title" as clues to the attachment's contents. All messages that are not convertible could be indicated by text in the subject line.

Often the local X.400 version (1984, 1988, or 1992) would be the major factor in whether an X.400 message or an IPM body part is convertible. The type of a message or a body part could be used to discard that element and replace with an indicator to the CMC user.

A simple convention is recommended for the basic (minimal) profile. The message sending side's profile has the non-delivery report request set so that reports can appear in the Message Store. As the major information contained in the report is whether the message was delivered or not, then only this needs to be returned to the messaging-enabled application. Thus a required conversion is the substitution of a report by a message containing the text_note of "this message was (not) delivered" along with the recipients, etc.

Other conversions will be needed if the content-type is for a different version (X.400 year) and the body parts are not the ones expected. For example, a Bilaterally Defined body part in place of an Externally Defined Body Part or a File Transfer body part in place of an IA5Text body part. Although conversion from one body part to another in these cases requires extra code, the conversion rules are simple.

B.2.5.4.3 Converting non-CMC/X.400 messages

When a messaging-enabled application or its user has an X.400 address, the X.400 Message Store will store any X.400 message such as an EDI message, IPM, or report that is sent to that address. Thus the user can send many messages that are not CMC-originated messages or are CMC messages that use different conventions that the local CMC mail system cannot handle. To deal with these issues, there needs to be a set of conventions on what is discarded, on what is partially converted so that the user knows some of the message's details and what parts can be substituted or omitted.

Simple conversion possibilities are:

- All non-CMC messages are discarded when they are "read" by a CMC application.
- All CMC messages in a different form (such as a PEM message as an X.400 P22 body part) that cannot be handled by the local system (e.g. 1984 X.400-based) are discarded. Those body parts that the local CMC and X.400 implementation can deal with are replaced by either a simple text_note, a character string file, or a "binary" file.
- Convert all unknown body parts into a partially usable file form by using a simple display conversion of all printable characters as is and converting all non-printable characters into a display form of "\’hex"hex”" so that a "user" can determine if that file is salvageable.

As before, special optional CMC extensions can be used to convey extra information back to the messaging-enabled application or to provide hints and requests to the CMC Read function on how to convert or interpret various attributes and body parts in the received message. As well as handling strange messages, the local CMC interface should have conventions dealing with error returns coming from the underlying messaging system. The issue is whether those messaging system errors are passed up to the CMC messaging-enabled application or not. And, if so, then in what form: either converted to a common CMC error convention, or left in their base form?

Underlying messaging error choices are:

- Replace with CMC_E_FAILURE on calls, else, ignore.
- If present, put raw error into a special X.400 error CMC extension.
- Convert error to a standard CMC extension for communication errors.

B.2.5.4.4 Extra CMC extensions

There are many things that could be done with extensions. The following list is a quick sample:

- extensions to select which of several underlying messaging systems to use;
- extensions to configure and set up the environment that CMC and the messaging system use;
- extensions to add extra X.400 attributes beyond the basic set used by CMC;
- extensions that determine what X.400 body parts to use based on several factors;
- extensions to tell the CMC read function how to handle non-basic incoming messages;
- extensions that are passed with the message system to tell the receiver how to process it.

The use of extensions raises a range of issues. The first issue is that of whether they are really necessary. A basic set-up could or should rely on no extensions but just handle the minimum defined requirements of the simple CMC calls in an acceptable manner. The various implementations can deal specifically with local requirements and so tailor their actions and behaviour.

There are many possible CMC_extensions and also ways to characterize those extensions. There are extensions that are necessary for adapting the local CMC version 1.0 to its local underlying messaging system or even to select which of its local messaging systems to use.

These extensions are characterized as local. Other extensions can be termed special as they deal with the use of non-CMC features (at least for CMC version 1.0). Such extensions are needed for interoperability.

For Recommendation X.400, there are extensions that are required for handling X.400 messaging that are basic and thus generic. The generic X.400 extensions would handle X.400 result and error returns. These generic extensions could be even further generalized to encompass result and error returns from other non-X.400 messaging services.

The error returns from the X.400 calls can be returned to the messaging-enabled application that uses the CMC calls by using a specific CMC_extension. This is essential for those CMC call clients that need to recover by finding out where the call went astray and not be left in the dark.

Other generic X.400 CMC_extensions would deal with message priority, request non-delivery and/or delivery reports. Other extensions could carry the user's X.400 certificate and Message Store restrictions. The CMC_message's message_type carries identifiers such as Object Identifiers that allow the CMC message sender to specify any form of X.400 message. If the CMC functions or the underlying X.400 system does not support that particular X.400 message type (or specifically require that type not be used) then another "generic" extension would be needed to carry back to the CMC call user that error information. Similar issues surround adding a special extension to select the X.400 body part appropriate for the CMC_attachment's attach_type.

Other possible extensions can be used to assist the receiving side deal with incoming messages such as which reports are to be discarded or not read but reported in a special way. End-to-end transfer of extensions (or their information) to assist the receiver in handling the message is too far-out for immediate consideration.

B.2.5.5 Basic mapping profile for simple CMC (CMC 1.0)

When the X.400 Message Handling System is used as the underlying messaging system, the basic message sending and receiving must be supported by all CMC version 1.0 implementations. For this purpose, a basic profile provides that functionality.

Each CMC implementation using the "basic profile" approach must be able to send (CMC Send) any message that requires only the basic IPM format and uses the two recommended X.400 body parts for attachments (and text_note) where required. Also those X.400 messaging-enabled applications that use more advanced body parts must be able to accept the basic profile set in their place. CMC implementations receiving (CMC Read) messages delivered by Recommendation X.400 must be able to correctly read and deal with those "basic profile" messages. All other X.400 messages may be rejected.

For the basic profile, the following rules should be applied: if the incoming message or the structure of the outgoing message does not fit a standard pattern, then that message should be discarded and optionally a warning or error returned to the messaging-enabled application. Similarly, when the input parameters do not match the allowable set or cannot be handled by the underlying messaging service, then an error (or a warning if that's not feasible) should be returned to the messaging-enabled application.

The rest of this subclause describes the minimal basic mapping required for the simplistic usage of the X.400 messaging services. Use of the full X.400 message services that are available can be provided by more additional/optional extensions that have not been added in this basic profile. Thus, considerations must be made for interoperability (if feasible) between the basic profile and that profile extended by extensions.

NOTE – A special naming convention is used when referencing the various X.400 defined fields. Each field is named using the standard followed by the field name as it appears in the standard. If the field is defined within a field, a "." character is used to indicate this. For example: X.420.heading.authorizing-users.

B.2.5.5.1 Mapping of CMC_recipient

The CMC_recipient is mapped to an X.411.ORName using the textual Representation of O/R Address for Human Usage that is defined in ISO/IEC 10021-2:1990/Amd.1, Annex F. All legal form of ORName as defined in X.411 can be used.

Specific mappings are:

CMC_recipient.name

maps to an X.411.ORName.directory-name if the implementation supports the directory name; otherwise, it is ignored.

CMC_recipient.name_type

is assumed to be INDIVIDUAL on output from X.400 to CMC.

CMC_recipient.address

maps to X.411.ORName.ORAddress.

CMC_recipient.role

If CMC_ROLE_ORIGINATOR, maps to X.411.OriginatorName.ORAddress. It also maps to X.420.Heading.originator when the message is an IPM or IPN.

If CMC_ROLE_TO, maps to X.411.RecipientName.ORAddress. It also maps to X.420.Heading.primary-recipients when the message is an IPM or IPN.

If CMC_ROLE_CC, maps to X.411.RecipientName.ORAddress. It also maps to X.420 Heading.copy-recipients when the message is an IPM or IPN.

If CMC_ROLE_BCC, maps to X.411.RecipientName.ORAddress. It also maps to X.420.Heading.blind-copy-recipients when the message is an IPM or IPN.

If CMC_ROLE_AUTHORIZING_USER, maps to X.420.Heading.authorizing-users when the message is an IPM or IPN.

CMC_recipient.recip_flags

are inspected for the last.

CMC_recipient.recip_extensions

are ignored.

B.2.5.5.2 Mapping of CMC_message

CMC_message_summary is mapped to the information base associated with an X.413 message submission operation or an X.413 Fetch operation.

Specific mapping are:

CMC_message.message_reference

maps to X.413.entry-sequence-number.

CMC_message.message_type

maps to X.413.entryType; where "CMC:IPM" = delivered-message, "CMC: REPORT" = delivered-report, and "CMC:IPM" = returned-content with the new extension CMC_X_X400_MSG_PARENT to identify this is a nested message.

CMC_message.subject

maps to X.413.Content of X.420.heading.subject.

CMC_message.time_sent

is NULL on **cmc_send()**, or maps to X.413.Message-submission-time on **cmc_read()**.

CMC_message.text_note

is NULL, or on **cmc_send()** and if CMC_TEXT_NOTE_AS_FILE is set, then maps to the first body part of X.413.Content of X.420.body.ia5text.data, or on **cmc_read()** maps to the first available ia5text body part.

The X.420.body.ia5text.repertoire is ignored.

CMC_message.recipient

maps to X.411.RecipientName AND X.420.heading.originator, authorizing-users, primary-recipients, copy-recipients, and blind-copy-recipients in accordance with the CMC_recipient.role setting.

CMC_message.attachment

maps to X.420.body.BodyPart. Each attachment is mapped to the corresponding body part. See B.2.5.5.4 "Mapping of CMC_attachment".

CMC_message.message_flags

sets according to X.413.EntryStatus for CMC_SUM_READ and CMC_SUM_UNSENT, checking for last element for CMC_SUM_LAST_ELEMENT, and CMC_MSG_TEXT_NOTE_AS_FILE for first attachment and first ia5text body part handling.

CMC_message.message_extensions

is NULL or optionally returns CMC_X_X400_MSG_PARENT extension.

B.2.5.5.3 Mapping of CMC_message_summary

CMC_message_summary is mapped to the parameters associated with an X.413 list or summarize operation.

Specific mappings are:

CMC_message_summary.message_reference

maps to X.413.entry-sequence-number.

CMC_message_summary.message_type

maps to X.413.entryType; where "CMC:IPM" = delivered-message, "CMC: REPORT" = delivered-report, and "CMC:IPM" = returned-content.

CMC_message_summary.subject

maps to X.413.Content of X.420.heading.subject.

CMC_message_summary.time_sent

maps to X.413.Message-submission-time.

CMC_message_summary.byte_length

maps to X.413.Content-length.

CMC_message_summary.originator

maps to X.413.Originator-name.

CMC_message.summary_flags

sets according to X.413.EntryStatus and/or last element.

CMC_message_summary.message_summary_extensions

is NULL or optionally returns CMC_X_X400_MSG_REPORT extension.

B.2.5.5.4 Mapping of CMC_attachment

CMC_attachment maps to body parts of the message associated with an X.413 submission operation or an X.413 Fetch operation. If the CMC_message has a text note, it is used as the first ia5text body part. Each attachment is mapped to an X.420 body part.

Specific mappings are:

CMC_attachment.attach_title

is NULL, or maps to the X.420.body.ExternallyDefinedBodyPart.data.dataValueDescriptor.

CMC_attachment.attach_type

is NULL, or maps to OID of an X.420.body.ExternallyDefinedBodyPart.data.directReference.

A CMC_ATT_TEXT type OID is mapped to the build-in type or OID of an X.420.body.IA5TextBodyPart. The IA5TextBodyPart.data.repertoire field is not used.

A CMC_ATT_BINARY type is mapped to the build-in type or OID of an X.420.body.BilaterallyDefinedBodyPart.

Other attach_type maps to X.420.body.ExternallyDefinedBodyPart. The fields of the EXTERNAL type are mapped as follows:

- Parameter (optional) is not used.
- Direct Reference data maps to the specified OID.
- Indirect Reference data (optional) is not used.
- Data value descriptor (optional) is not used.
- Encoding is set to arbitrary.

CMC_attachment.attach_filename

maps to the implementation's external file name and is not passed to X.400. The content of the file is stored as the data of the corresponding X.400 body parts.

CMC_attachment.attach_flags

maps according to the owner of the attach_filename and the last element of an attachment.

CMC_attachment.attach_extensions

NULL.

B.2.5.6 Mapping of CMC functions

Most of the CMC functions are mapped to a ROSE operation containing an X.413 operation. The invoke Ids used within a ROSE operation must be a unique number. The invoke Ids can be generated by the implementation. Also, linked Ids are not used. If any extensions or message types cannot be supported in this basic profile, then an error is returned to the CMC functions and the corresponding messages (or portion of it) will be discarded or ignored.

B.2.5.6.1 CMC Act On

cmc_act_on() maps to a ROSE envelope containing an X.413 Delete operation. The X.413 Delete operation has the following structure:

[information-base-type, items (choice of selector or sequence-number)]

The information-base-type is X.413.store-message (default). The items are assumed to be X.413.EntrySequenceNumber supplied by message_reference.

Parameter mapping

```
CMC_return_code
cmc_act_on(

CMC_session_id              session,              local session id
CMC_message_reference       *message_reference,   X.413.EntrySequenceNumber
CMC_enum                    operation,            supports CMC_ACT_ON_DELETE
                                                  only.(due to underlying X.400
                                                  operations)
CMC_flags                   act_on_flags,         NULL or ignored
CMC_ui_id                   ui_id,                NULL or ignored
CMC_extension               *act_on_extensions    NULL or CMC_X_X400_ERROR on
                                                  output
);
```

Additional comments

None.

B.2.5.6.2 CMC Free

cmc_free() does not require any mapping to X.400 calls.

Parameter mapping

```
CMC_return_code
cmc_free(
CMC_buffer                memory
);
```

Additional comments

None.

B.2.5.6.3 CMC List

cmc_list() maps to a ROSE envelope with an X.413 List or Summarize operation. The X.413 List and Summarize operations have the following structure:

[information-base-type, selector, (requested-attributes or summary-request)]

The information-base-type is X.413.store-message (default). The items are assumed to be X.413.EntrySequenceNumber supplied by message_reference.

Parameter mapping

```
CMC_return_code
cmc_list(
   CMC_session_id         session,              local session id
   CMC_string             message_type,         list filter on entryType
   CMC_flags              list_flags,           UNREAD, REF_ONLY, COUNT_ONLY
   CMC_message_reference  *seed,                X.413.EntrySequenceNumber
   CMC_uint32             *count,               Counter
   CMC_ui_id              ui_id,                NULL
   CMC_message_summary    **result,             CMC:IPM or CMC: REPORT or OID
   CMC_extension          *list_extensions      Null or CMC_X_X400_ERROR
);
```

Additional comments

1) Child Entries (optional) is not used.

2) If a seed is provided in the CMC call, then it can be used to specify a range: select a sequence number FROM range and use the seed provided. The sequence TO range (optional) is not used.

3) If the CMC message type parameter is specified OR the CMC_LIST_UNREAD_ONLY flag is set, a filter is used. When the two conditions are present, use an AND operator. If CMC_LIST_UNREAD_ONLY is set, a filter element is set to: "Not Item Equality EntryStatus Value (processed)". If message type is specified, a filter element is set to: "Item Equality EntryType Value (message OR report)".

4) Limit is specified if the CMC list count parameter is not zero. If so, list count maps directly to the limit integer value.

5) Override (optional) is not used.

6) The following attributes are to be returned from the list operation:

   id-att-parent-sequence-number

   id-att-entry-type

   id-att-originator-name

   id-att-content-length

   id-att-message-submission-time

   id-att-subject

   id-att-content-type

B.2.5.6.4 CMC Logoff

cmc_logoff() maps to a ROSE envelope with an X.413 Unbind operation. The X.413 Unbind operation has no argument, result, nor error.

Parameter mapping

```
CMC_return_code
cmc_logoff(
   CMC_session_id        session,              local session id
   CMC_ui_id             ui_id,                NULL
   CMC_flags             logoff_flags,         NULL
   CMC_extension         *logoff_extensions    NULL
);
```

Additional comments

None.

B.2.5.6.5 CMC Logon

cmc_logon() maps to a ROSE envelope with an X.413 MS Bind operation. The X.413 MS Bind operation has the following structure:

[initiator-name, initiator-credentials, security-context, fetch-restrictions, ms-configuration-request]

Parameter mapping

```
CMC_return_code
cmc_logon(
   CMC_string            service,              NULL, or local path service
   CMC_string            user,                 that accesses the MS textual
                                               form of initiator-name,
                                               see B.2.4.4.1 "Mapping of
                                               CMC_recipient"
   CMC_string            password,             X.411.initiator-
                                               credentials.simple
   CMC_object_identifier character_set,        Password NULL or local return
                                               from Query
   CMC_ui_id             ui_id,                Configuration NULL
   CMC_uint16            caller_cmc_version,   local version number v1.0
   CMC_flags             logon_flags,          NULL or is not used
   CMC_session_id        *session,             local session id
   CMC_extension         *logon_extensions     NULL, or CMC_X_X400_ERROR
);
```

Additional comments

The following optional elements are not used:

a) MS Security Context;

b) Fetch Restriction;

c) MS Configuration Request.

B.2.5.6.6 CMC Look Up

cmc_look_up() does not require any mapping or X.400 calls. This function is not mandatory for supporting of the X.400 messaging service.

Parameter mapping

```
CMC_return_code
cmc_look_up(
   CMC_session_id        session               local session id
   CMC_recipient         *recipient_in         see mapping of CMC_recipient
   CMC_flags             look_up_flags         all zero, or local
                                               implementations
   CMC_ui_id             ui_id                 NULL
   CMC_uint32            *count                output from local
                                               implementation
   CMC_recipient         **recipient_out       see mapping of CMC_recipient
   CMC_extension         *look_up_extensions   NULL
);
```

Additional comments

None.

B.2.5.6.7 CMC Read

cmc_read() maps to a ROSE envelope with an X.413 Fetch operation. The X.413 Fetch operation has the following structure:

[information-base-type, item choice of search (set of seq#) or precise (sequence#), requested_attributes]

The information-base-type is X.413.store-message (default). The items are assumed to be X.413.EntrySequenceNumber supplied by message_reference.

Parameter mapping

```
CMC_return_code
cmc_read(
   CMC_session_id        session,              local session id
   CMC_message_reference *message_reference,   NULL for first UNREAD, or
                                               X.413.EntrySequenceNumber
   CMC_flags             read_flags,           cannot support
                                               CMC_DO_NOT_MARK_AS_READ
   CMC_message           **message             pointer to stored-message,
                                               see mapping of CMC_message
   CMC_ui_id             ui_id                 NULL
   CMC_extension         *read_extensions      NULL, or CMC_X_X400_ERROR
);
```

Additional comments

If the CMC message reference parameter is specified, select a precise fetch. Otherwise, select a search fetch.

Case Precise:

The supplied message reference can be used as the MS sequence number.

Case Search:

Child Entries (optional) is not used.

Select a sequence number FROM range and use the "0" sequence number.

The sequence TO range (optional) is not used.

A filter is specified if the CMC_READ_FIRST_UNREAD_ONLY flag is set. The filter item is set to "Not Item Equality EntryStatus Value (processed)".

Limit (optional) is not used.

Override (optional) is not used.

The following attributes are to be returned:

id-att-parent-sequence-number

id-att-entry-type

id-att-message-submission-time

id-att-content-type (IPM, IPN, Externally defined, etc.)

B.2.5.6.8 CMC Send

cmc_send() maps to a ROSE envelope with X.413 Submission operation. The X.413 Submission operation has the following structure:

[envelope(MessageSubmissionEnvelope) with an IPM content(Content)]

Parameter mapping

```
CMC_return_code
cmc_send(
   CMC_session_id        session,              local session id
   CMC_message           *message,             see mapping of CMC_message
   CMC_flags             send_flags,           always zero
   CMC_ui_id             ui_id,                NULL
   CMC_extension         *send_extensions      NULL, or CMC_X_COM_PRIORITY,
                                               or CMC_X_X400_ERROR
);
```

Additional comments

X.411 MessageSubmissionEnvelope (P3 Envelope):

The Originator is filled-in with the originator ORName (see ORName mapping). If originator is not specified, then the ORName used in **cmc_logon()** will be used.

Original Encoded Information type (optional) is not used.

Content type is set to Built-in and the type is IPM-84.

Content Identifier (optional) is not used.

Priority (optional) is normal (default), or is mapped from CMC_X_COM_PRIORITY if present.

Per Message Indicator (optional) is using the default values:

disclosure-of-recipient is prohibited (default);

implicit-conversion-prohibited is allowed (default);

alternate-recipient-allowed is prohibited (default);

content-return-requested is not requested (default).

Deferred Delivery Time (optional) is not used.

Extensions (optional) is not used.

All Recipients' ORName addresses are filled-in (see mapping of CMC_recipient). After each Recipient's ORName is filled-in the following fields are set:

Originator Report Request is "non-delivery report" (default).

Explicit Conversion (optional) is prohibited (default).

Extensions (optional) is not used.

The content of the P3 envelope is a P2 IPM message that is a P2 envelope and one or more body parts.

X.420.IPM.heading (P2 Envelope):

The user is filled-in with the originator ORName (see mapping of CMC_recipient).

The user relative identifier is formed by appending the same invoke ID generated for the ROSE envelope to the string "CMC:IPM".

For Primary recipients, Copy recipients, and Blind Copy recipients, Notification requests (optional) and Reply requested (optional) are not used.

Replied To IPM (optional) is not used.

Obsolete IPMs (optional) is not used.

Related IPMs (optional) is not used.

Message Subject maps directly to the CMC_message subject field.

Expire Time (optional) is not used.

Reply Time (optional) is not used.

Reply Recipients (optional) is not used.

Importance (optional) is normal (default).

Sensitivity (optional) is not used.

Auto-Forwarded (optional) is not used.

Extensions (optional) is not used.

X.420.IPM.body:

If the CMC_message has a text note, it is used as the first IA5Text body part. When creating an IA5Text body part, the repertoire field (optional) is ignored.

For all other CMC attachments, a corresponding X.400 body part is created. If CMC_attachment type is CMC_ATT_TEXT, it is mapped to an IA5Text body part. If type is CMC_ATT_BINARY, it is mapped to a Bilaterally defined body part. For other CMC types, the supplied OID value is used and creates an externally defined body part (EDBP).

B.2.5.6.9 CMC Send Documents

cmc_send_documents() maps to the ROSE envelopes each containing an X.413 MS Bind operation, X.413 Submission operation, X.413 MS Unbind operation. They are the combined sequence of **cmc_logon()**, **cmc_send()**, and **cmc_logoff()**.

Parameter mapping

```
CMC_return_code
cmc_send_documents(
    CMC_string          recipient_addresses,   X.411.envelope.recipients and
                                               X.420 authorizing users,
                                               primary, copy, and blind copy
                                               recipients; see mapping of
                                               CMC_recipient
    CMC_string          subject,               X.420.heading.subject
    CMC_string          text_note,             first X.420.body.IA5TextBo-
                                               dyPart.data
    CMC_flags           send_doc_flags,        caller-supplied flags
    CMC_string          file_paths,            local filenames, not passed
                                               in X.400 message
    CMC_string          attach_titles,         X.420.EDBP.data.dataVa-
                                               lueDescriptor
    CMC_string          delimiter,             delimiter character
    CMC_ui_id           ui_id,                 NULL
);
```

Additional comments

None.

B.2.5.6.10 CMC Query Configuration

This call does not require any mapping or X.400 calls. It simply returns the configuration of the specified item.

Parameter mapping

```
CMC_return_code
cmc_query_configuration(
   CMC_session_id        session,              local session id
   CMC_enum              item,                 local implementation
   CMC_buffer            reference,            local implementation
   CMC_extension         *config_extensions    NULL
);
```

Additional comments

None.

Annex C

Programming examples

C.1 Programming examples

This Recommendation offers the following programming examples.

C.1.1 Query Configuration, Logon, and Logoff

```
/* local variables used */

CMC_return_code      Status;
CMC_boolean          UI_available;
CMC_session_id       Session;

/* find out if UI is available with this implementation before starting */

Status = cmc_query_configuration(
          NULL,                             /* No session id.                */
          CMC_CONFIG_UI_AVAIL,              /* See if UI is available.       */
          &UI_available,                    /* Return value.                 */
          NULL);                            /* No extensions.                */
       /* error handling */

/* Log on to system using UI */

Status = cmc_logon(
          NULL,                             /* Default service.              */
          NULL,                             /* Prompt for username.          */
          NULL,                             /* Prompt for password.          */
          NULL,                             /* Default Character set.        */
          (CMC_ui_id)NULL,                  /* Default UI ID.                */
          CMC_VERSION,                      /* Version 1 CMC calls.          */
          CMC_LOGON_UI_ALLOWED |            /* Full logon UI.                */
          CMC_ERROR_UI_ALLOWED,             /* Use UI to display errors.     */
          &Session,                         /* Returned session id.          */
          NULL);                            /* No extensions.                */
       /* error handling */

/* Do various CMC calls */

/* Log off from the implementation */

Status = cmc_logoff(

          Session,                          /* Session id.                   */
          (CMC_ui_id)NULL,                  /* No UI will be used.           */
          0,                                /* No flags.                     */
          NULL);                            /* No extensions.                */
       /* error handling */
```

C.1.2 Send and Send Documents functions

```
/* local variables used */

CMC_attachment       Attach;
CMC_session_id       Session;
CMC_message          Message;
CMC_recipient        Recip[2];
CMC_return_code      Status;

/* Build recipient list with two recipients. Add one "To" recipient. */

Recip[0].name            = "Bob Weaver";          /* Send to Bob Weaver.      */
Recip[0].name_type       = CMC_TYPE_INDIVIDUAL;   /* Bob’s a person.          */
Recip[0].address         = NULL;                  /* Look_up Bob’s address.   */
```

```
Recip[0].role            = CMC_ROLE_TO;            /* He’s a "To" recipient.    */
Recip[0].flags           = 0;                      /* Not the last element.     */
Recip[0].extensions      = NULL;                   /* No recipient extensions.  */

/* Add one "Cc" recipient. */

Recip[1].name            = "Mary Yu";              /* Send to Mary Yu.          */
Recip[1].name_type       = CMC_TYPE_INDIVIDUAL;    /* Mary’s a person.          */
Recip[1].address         = NULL;                   /* Look_up Mary’s address.   */
Recip[1].role            = CMC_ROLE_CC;            /* She’s a "Cc" recipient.   */
Recip[1].flags           = CMC_RECIP_LAST_ELEMENT;/* Last recipient element.   */
Recip[1].extensions      = NULL;                   /* No recipient extensions.  */

/* Attach a file. */

Attach.attach_title      = "stock.wks";            /* Original file name.       */
Attach.attach_typ        = NULL;                   /* No specific type.         */
Attach.attach_filename   = "tmp22.tmp";            /* File to attach.           */
Attach.attach_flags      = CMC_ATT_LAST_ELEMENT;   /* Last attachment.          */
Attach.attach_extensions= NULL;                    /* No attach. extensions.    */

/* Put it together in the message structure. */

Message.message_reference  = NULL;             /* Ignored on cmc_send calls.  */
Message.message_type       = NULL;             /* Interpersonal message type. */
Message.subject            = "Stock";          /* Message subject.            */
Message.time_sent          = NULL;             /* Ignored on cmc_send calls.  */
Message.text_note          = "Time to buy";    /* Message note.               */
Message.recipients         = Recip;            /* Message recipients.         */
Message.attachments        = &Attach;          /* Message attachments.        */
Message.message_flags      = 0;                /* No flags.                   */
Message.message_extensions = NULL;             /* No message extensions.      */

/* Send the message! */

Status = cmc_send(
          Session,                /* Session id. - set with logon call.      */
          &Message,               /* Message structure.                      */
          0,                      /* No flags.                               */
          (CMC_ui_id)NULL,        /* No UI will be used.                     */
          NULL);                  /* No extensions.                          */
       /* error handling */

/* Now do the same thing with the send documents call and UI */

Status = cmc_send_documents(
          "to:Bob Weaver,cc:Mary Yu",        /* Message recipients.          */
          "Stock",                           /* Message subject.             */
          "Time to buy",                     /* Message note.                */
          CMC_LOGON_UI_ALLOWED |
          CMC_SEND_UI_REQUESTED |
          CMC_ERROR_UI_ALLOWED,              /* Flags (allow various UI’s).  */
          "stock.wks",                       /* File to attach.              */
          "tmp22.tmp",                       /* File name to carry on attach. */
          ",",                               /* Multi-value delimiter.       */
          NULL);                             /* Default UI ID.               */
       /* error handling */
```

C.1.3 List, read, and delete the first unread message

```
/* local variables used */

CMC_message_summary  *pMsgSummary;
CMC_message          *pMessage;
CMC_uint32           iCount;

/* read the first unread message and delete it */

iCount = 5;

Status = cmc_list(
          Session,                /* Session id.                          */
          NULL,                   /* List ALL message types.              */
          CMC_LIST_UNREAD_ONLY,   /* Get only unread messages.            */
          NULL,                   /* Starting at the top.                 */
          &iCount,                /* Input/Output message count.          */
          (CMC_ui_id)NULL,        /* No UI will be used.                  */
          &pMsgSummary,           /* Return message summary list.         */
          NULL);                  /* No extensions.                       */
       /* error handling */

Status = cmc_read(
          Session,                          /* Session id.                */
          pMsgSummary[0]->message_reference, /* Message to read.          */
          CMC_MSG_AND_ATT_HDRS_ONLY,        /* don't get attach files.    */
          &pMessage,                        /* Returned message.          */
          (CMC_ui_id)NULL,                  /* No UI.                     */
          NULL);                            /* No extensions.             */
       /* error handling */

Status = cmc_act_on(
          Session,                          /* Session id.                */
          pMsgSummary[0]->message_reference, /* Message to delete.        */
          CMC_ACT_ON_DELETE,                /* Message to read.           */
          0,                                /* no flags.                  */
          (CMC_ui_id)NULL,                  /* No UI.                     */
          NULL);                            /* No extensions.             */
       /* error handling */

/* free the memory returned by the implementation */

Status = cmc_free(pMsgSummary);
Status = cmc_free(pMessage);

/* do the same thing without the list call, since the read call can get the first
unread mail message */

Status = cmc_read(
          Session,                          /* Session id.                */
          NULL,                             /* Read the first message.    */
          CMC_READ_FIRST_UNREAD_MESSAGE |   /* get first unread msg.      */
          CMC_MSG_AND_ATT_HDRS_ONLY,        /* don't get attach files.    */
          &pMessage,                        /* Returned message.          */
          (CMC_ui_id)NULL,                  /* No UI.                     */
          NULL);                            /* No extensions.             */
       /* error handling */

Status = cmc_act_on(
          Session,                          /* Session id.                */
          pMessage->message_reference,      /* message to delete.         */
          CMC_ACT_ON_DELETE,                /* Message to read.           */
          0,                                /* no flags.                  */
          (CMC_ui_id)NULL,                  /* No UI.                     */
          NULL);                            /* No extensions.             */
       /* error handling */

/* free the memory returned by the implementation */

Status = cmc_free(pMessage);
```

C.1.4 Look up a specific recipient and get its details

```
/* local variables used */

CMC_session_id       Session;
CMC_recipient        *pRecipient;
CMC_recipient        Recip;
CMC_return_code      Status;

/* look up a name to pick correct recipient */

Recip.name              = "Bob Stack";          /* Send to Bob Weaver.       */
Recip.name_type         = CMC_TYPE_INDIVIDUAL;  /* Bob’s a person.           */
Recip.address           = NULL;                 /* Look_up Bob’s address.    */
Recip.role              = NULL;                 /* Role not used.            */
Recip.recip_flags       = 0;                    /* No flags.                 */
Recip.recip_extensions  = NULL;                 /* No recipient extensions.  */

Status = cmc_look_up(
          Session,                          /* Session id.                */
          &Recip,                           /* Name to look up.           */
          CMC_LOOKUP_RESOLVE_UI |           /* Disambiguate using UI.     */
          CMC_ERROR_UI_ALLOWED,             /* Display errors using UI.   */
          (CMC_ui_id)NULL,                  /* Default UI ID.             */
          1,                                /* Only want 1 back.          */
          pRecipient,                       /* Returned recipient ptr.    */
          NULL);                            /* No extensions.             */

/* Display details stored for this recipient */

Status = cmc_look_up(
          Session,                          /* Session id.                */
          pRecipient,                       /* Name to get details on.    */
          CMC_LOOKUP_DETAILS_UI |           /* Show details UI.           */
          CMC_ERROR_UI_ALLOWED,             /* Display errors using UI.   */
          (CMC_ui_id)NULL,                  /* Default UI ID.             */
          0,                                /* No limit on return count.  */
          NULL,                             /* No records returned.       */
          NULL);                            /* No extensions.             */

/* free the memory returned by the implementation */

cmc_free(pRecipient);
```

C.1.5 Use of extensions

```
/* local variables used */

CMC_return_code      Status;
CMC_session_id       Session;
CMC_extension        Extension;
CMC_X_COM_support    Supported[2];
CMC_uint16           index;

/*  find out if the common extension set is supported, but I don’t need
    COM_X_CONFIG_DATA support */

Supported[0].item_code =        CMC_XS_COM;
Supported[0].flags =            0;

Supported[1].item_code =        CMC_X_COM_CONFIG_DATA;
Supported[1].flags =            CMC_X_COM_SUP_EXCLUDE;

Extension.item_code =           CMC_X_COM_SUPPORT_EXT;
Extension.item_data =           2;
Extension.item_reference =      Supported;
Extension.extension_flags =     CMC_EXT_LAST_ELEMENT;
```

```
Status = cmc_query_configuration(
          NULL,                              /* No session id.                */
          CMC_CONFIG_UI_AVAIL,               /* See if UI is available.       */
          &UI_available,                     /* Return value.                 */
          &Extension);                       /* Pass in extensions.           */
       /* error handling */
if (Supported[0].flags & CMC_X_COM_NOT_SUPPORTED)
       return FALSE;  /* common extensions I need are not available */

/* Log on to system and get the data extensions for this session */

Supported[0].item_code =        CMC_XS_COM;
Supported[0].flags =            0;

Supported[1].item_code =        CMC_X_COM_CONFIG_DATA;
Supported[1].flags =            CMC_X_COM_SUP_EXCLUDE;

Extension.item_code =           CMC_X_COM_SUPPORT_EXT;
Extension.item_data =           2;
Extension.item_reference =      Supported;
Extension.extension_flags =     CMC_EXT_REQUIRED | CMC_EXT_LAST_ELEMENT;

Status = cmc_logon(
          NULL,                              /* Default service.              */
          NULL,                              /* Prompt for username.          */
          NULL,                              /* Prompt for password.          */
          NULL,                              /* Default Character set.        */
          (CMC_ui_id)NULL,                   /* Default UI ID.                */
          CMC_VERSION,                       /* Version 1 CMC calls.          */
          CMC_LOGON_UI_ALLOWED |             /* Full logon UI.                */
          CMC_ERROR_UI_ALLOWED,              /* Use UI to display errors.     */
          &Session,                          /* Returned session id.          */
          &Extension);                       /* Logon extensions.             */
       /* error handling */
if (Supported[0].flags & CMC_X_COM_NOT_SUPPORTED)
       return FALSE;  /* common extensions I need are not available */
       /* the common data extensions will be used for this session */

/*  example of how to free data returned from the CMC implementation in
   function output extensions. */

for (index = 0; ; index++)  {
       if (Extensions[index].extension_flags & CMC_EXT_OUTPUT) {
          if (cmc_free(Extensions[index].item_reference) != CMC_success)      {
             /* Handle unexpected error here */
             }
             }
          (Extensions[index].extension_flags & CMC_EXT_LAST_ELEMENT)
             break;
       }

/* Do various CMC calls */

/* Log off from the implementation */

Status = cmc_logoff(
          Session,                 /* Session id.                             */
          (CMC_ui_id)NULL,         /* No UI will be used.                     */
          0,                       /* No flags.                               */
          NULL);                   /* No extensions.                          */
       /* error handling */
```

C.1.6 cmc_bind_implementation

```
   CMC_return_code   Status = CMC_SUCCESS;
   CMC_boolean       UI_available;
   CMC_session_id    Session;
   HINSTANCE         hlibCMC = (HINSTANCE)NULL;
```

```
CMC_P_BIND_IMPLEMENTATION lpfnCMCBindImplementation = NULL;
CMC_dispatch_table           *pDispatchTable = NULL;
CMC_object_handle            root_object_handle = CMC_NULL_HANDLE;
extern CMC_guid              selected_implementation_name;

if (!(hlibCMC = LoadLibrary ("CMC.DLL")))
       {
       /* error handling */
       }

if (!(lpfnCMCBindImplementation =
   (CMC_P_BIND_IMPLEMENTATION)GetProcAddress (hlibCMC, "cmc_bind_implement-
    ation")))
       {
       /* error handling */
       }

/* Call into a selected CMC Manager to bind to specific CMC implementation.*/

Status = lpfnCMCBindImplementation(selected_implementation_name,
                             &pDispatchTable,
                             NULL);

if (pDispatchTable == NULL)
       {
       /* error handling */
       }

/* find out if UI is available with this implementation before starting.
   Mixing calls, be careful. */

Status = pDispatchTable->cmc_query_configuration(
       NULL,                            /* No session id.                */
       CMC_CONFIG_UI_AVAIL,             /* See if UI is available.       */
       &UI_available,                   /* Return value.                 */
       NULL);                           /* No extensions.                */
   /* error handling */

/* Log on to system using UI */

Status = pDispatchTable->cmc_logon(
       NULL,                            /* Default service.              */
       NULL,                            /* Prompt for username.          */
       NULL,                            /* Prompt for password.          */
       NULL,                            /* Default Character set.        */
       (CMC_ui_id)NULL,                 /* Default UI ID.                */
       CMC_VERSION,                     /* Version 1 CMC calls.          */
       CMC_LOGON_UI_ALLOWED |           /* Full logon UI.                */
       CMC_ERROR_UI_ALLOWED,            /* Use UI to display errors.     */
       &Session,                        /* Returned session id.          */
       NULL);                           /* No extensions.                */
   /* error handling */

/* Make calls into specific CMC implementation. */

Status = pDispatchTable->cmc_get_root_handle(Session,
                             &root_object_handle,
                             NULL);
   /* error handling */

/* Log off from the implementation */

Status = pDispatchTable->cmc_logoff(
       Session,             /* Session id.            */
       (CMC_ui_id)NULL,     /* No UI will be used.    */
       0,                   /* No flags.              */
       NULL);               /* No extensions.         */
   /* error handling */
/* Let implementation free the dispatch table it gave us. */

pDispatchTable->cmc_free(pDispatchTable);

/* Unbind from the CMC implementation. Cleans up storage
   created in CMC Manager and/or CMC implementation other
   than the CMC_dispatch_table. */

cmc_unbind_implementation(selected_implementation_name,
                             NULL);

   /* error handling */

/* Free the DLL instance. */

FreeLibrary (hlibCMC);
```

C.2 Example of cmc_bind_implementation

```
CMC_return_code
cmc_bind_implementation(
   CMC_guid                implementation_name,
   CMC_dispatch_table      **dispatch_table,
   CMC_extension           *cmc_bind_extensions
)
{
SCODE sc = SUCCESS_SUCCESS;
CMC_dispatch_table *pDispatchTable=NULL;

   pDispatchTable = (CMC_dispatch_table *)(calloc(1, sizeof(CMC_dispatch_table)));
   if (pDispatchTable == NULL)
      return CMC_E_INSUFFICIENT_MEMORY;

   else

      /* Store local for later cmc_free processing. */

   /* Populate the Dispatch Table. */

   pDispatchTable->cmc_send = cmc_send;
   pDispatchTable->cmc_send_documents = cmc_send_documents;
   pDispatchTable->cmc_act_on = cmc_act_on;
   pDispatchTable->cmc_list = cmc_list;
   pDispatchTable->cmc_read = cmc_read;
   pDispatchTable->cmc_look_up = cmc_look_up;
   pDispatchTable->cmc_free = cmc_free;
   pDispatchTable->cmc_logoff = cmc_logoff;
   pDispatchTable->cmc_logon = cmc_logon;
   pDispatchTable->cmc_query_configuration = cmc_query_configuration;
   pDispatchTable->cmc_add_object = cmc_add_object;
   pDispatchTable->cmc_add_properties = cmc_add_properties;
   pDispatchTable->cmc_commit_object = cmc_commit_object;
   pDispatchTable->cmc_copy_object_handle = cmc_copy_object_handle;
   pDispatchTable->cmc_create_derived_message_object =
                 cmc_create_derived_message_object;
   pDispatchTable->cmc_delete_objects = cmc_delete_objects;
   pDispatchTable->cmc_delete_properties = cmc_delete_properties;
   pDispatchTable->cmc_get_root_handle = cmc_get_root_handle;
   pDispatchTable->cmc_identifier_to_name = cmc_identifier_to_name;
   pDispatchTable->cmc_list_contained_properties = cmc_list_contained_properties;
   pDispatchTable->cmc_list_number_matched = cmc_list_number_matched;
   pDispatchTable->cmc_list_objects = cmc_list_objects;
   pDispatchTable->cmc_list_properties = cmc_list_properties;
   pDispatchTable->cmc_name_to_identifier = cmc_name_to_identifier;
   pDispatchTable->cmc_open_cursor = cmc_open_cursor;
   pDispatchTable->cmc_open_object_handle = cmc_open_object_handle;
   pDispatchTable->cmc_read_cursor = cmc_read_cursor;
   pDispatchTable->cmc_read_properties = cmc_read_properties;
   pDispatchTable->cmc_read_property_costs = cmc_read_property_costs;
   pDispatchTable->cmc_restore_object = cmc_restore_object;
   pDispatchTable->cmc_save_object = cmc_save_object;
   pDispatchTable->cmc_send_message_object = cmc_send_message_object;
   pDispatchTable->cmc_update_cursor_position = cmc_update_cursor_position;
   pDispatchTable->cmc_update_cursor_position_with_seed =
                 cmc_update_cursor_position_with_seed;
```

```
    pDispatchTable->cmc_check_event = cmc_check_event;
    pDispatchTable->cmc_register_event = cmc_register_event;
    pDispatchTable->cmc_unregister_event = cmc_unregister_event;
    pDispatchTable->cmc_call_callbacks = cmc_call_callbacks;
    pDispatchTable->cmc_export_stream = cmc_export_stream;
    pDispatchTable->cmc_import_file_to_stream = cmc_import_file_to_stream;
    pDispatchTable->cmc_open_stream = cmc_open_stream;
    pDispatchTable->cmc_read_stream = cmc_read_stream;
    pDispatchTable->cmc_seek_stream = cmc_seek_stream;
    pDispatchTable->cmc_write_stream = cmc_write_stream;
    pDispatchTable->cmc_get_last_error = cmc_get_last_error;

    /* Load the output variable. */

    *dispatch_table = pDispatchTable;

    return CMC_SUCCESS;
}
```

C.3 Composing a message

```
    #define NUM_RECIP_PROPS              4
    #define NUM_MESSAGE_PROPS            4
    #define NUM_CONTENT_PROPS            6

    #define RECIP_NAME_INDEX             0
    #define RECIP_ADDRESS_INDEX          1
    #define RECIP_ROLE_INDEX             2
    #define RECIP_TYPE_INDEX             3

    #define MSG_TYPE_INDEX               0
    #define MSG_PRIORITY_INDEX           1
    #define MSG_SUBJECT_INDEX            2
    #define MSG_ROLE_INDEX               3

    #define CONTENT_CHARSET_INDEX        0
    #define CONTENT_INFORMATION_INDEX    1
    #define CONTENT_SIZE_INDEX           2
    #define CONTENT_TITLE_INDEX          3
    #define CONTENT_ITEMNUM_INDEX        4
    #define CONTENT_ITEMTYPE_INDEX       5

    CMC_return_code         Status = CMC_SUCCESS;
    extern                  CMC_session_id Session;
    extern                  CMC_dispatch_table *pDispatchTable;
    CMC_object_handle       Message = CMC_NULL_HANDLE;
    CMC_object_handle       Recipient = CMC_NULL_HANDLE;
    CMC_object_handle       ContentItem = CMC_NULL_HANDLE;
    CMC_string              error_buf = NULL;
    CMC_property            RecipientProps[NUM_RECIP_PROPS];
    CMC_property            MessageProps[NUM_MESSAGE_PROPS];
    CMC_property            ContentProps[NUM_CONTENT_PROPS];
    CMC_opaque_data         MessageBody;
    CMC_CHAR                MsgBuffer[MAX_BODY_LEN];

    /* Load Recipient Property Structure. */

    RecipientProps[RECIP_NAME_INDEX].property_id = CMC_PV_RECIPIENT_NAME;
    RecipientProps[RECIP_NAME_INDEX].type = CMC_string;
    RecipientProps[RECIP_NAME_INDEX].value.CMC_pv_string =
                                "Pierre Peret";

    RecipientProps[RECIP_ADDRESS_INDEX].property_id = CMC_PV_RECIPIENT_ADDRESS;
    RecipientProps[RECIP_ADDRESS_INDEX].type = CMC_string;
    RecipientProps[RECIP_ADDRESS_INDEX].value.CMC_pv_string =
                                "uunet!p.peret@A205.bull.com!USENET";

    RecipientProps[RECIP_ROLE_INDEX].property_id = CMC_PV_RECIPIENT_ROLE;
    RecipientProps[RECIP_ROLE_INDEX].type = CMC_enum;
    RecipientProps[RECIP_ROLE_INDEX].value.CMC_pv_enumerated =
                                CMC_RECIPIENT_ROLE_TO;

    RecipientProps[RECIP_TYPE_INDEX].property_id = CMC_PV_RECIPIENT_TYPE;
    RecipientProps[RECIP_TYPE_INDEX].type = CMC_enum;
    RecipientProps[RECIP_TYPE_INDEX].value.CMC_pv_enumerated =
                                CMC_RCT_INDIVIDUAL;
```

```
/* Load Message Property Structure. */

MessageProps[MSG_TYPE_INDEX].property_id = CMC_PV_MESSAGE_TYPE;
MessageProps[MSG_TYPE_INDEX].type = CMC_enum;
MessageProps[MSG_TYPE_INDEX].value.CMC_pv_enumerated =
                               CMC_MT_IPM;

MessageProps[MSG_PRIORITY_INDEX].property_id = CMC_PV_MESSAGE_PRIORITY;
MessageProps[MSG_PRIORITY_INDEX].type = CMC_enum;
MessageProps[MSG_PRIORITY_INDEX].value.CMC_pv_enumerated =
                               CMC_PRIORITY_NORMAL;

MessageProps[MSG_SUBJECT_INDEX].property_id = CMC_PV_MESSAGE_SUBJECT;
MessageProps[MSG_SUBJECT_INDEX].type = CMC_string;
MessageProps[MSG_SUBJECT_INDEX].value.CMC_pv_string =
                               "Hey Pierre, don't forget";

MessageProps[MSG_ROLE_INDEX].property_id = CMC_PV_MESSAGE_ROLE;
MessageProps[MSG_ROLE_INDEX].type = CMC_enum;
MessageProps[MSG_ROLE_INDEX].value.CMC_pv_enumerated =
                               CMC_MESSAGE_ROLE_ORIGINAL;

/* Load Message Content Item Property Structure. */

ContentProps[CONTENT_CHARSET_INDEX].property_id =
                               CMC_PV_CONTENT_ITEM_CHARACTER_SET;
ContentProps[CONTENT_CHARSET_INDEX].type = CMC_guid;
ContentProps[CONTENT_CHARSET_INDEX].value.CMC_pv_guid =
                               CMC_CHARSET_1252;

ContentProps[CONTENT_INFORMATION_INDEX].property_id =
                               CMC_PV_CONTENT_ITEM_CONTENT_INFORMATION;
ContentProps[CONTENT_INFORMATION_INDEX].type = CMC_opaque_data;

strcpy(MsgBuffer, "to FOCUS on the API");

MessageBody.size = strlen(MsgBuffer) + 1;
MessageBody.data = (CMC_byte *)calloc(1, strlen(MsgBuffer) + 1);

ContentProps[CONTENT_INFORMATION_INDEX].value.CMC_pv_opaque_data.
                               data = MessageBody.data;
ContentProps[CONTENT_INFORMATION_INDEX].value.CMC_pv_opaque_data.
                               size = MessageBody.size;

ContentProps[CONTENT_SIZE_INDEX].property_id =
                               CMC_PV_CONTENT_ITEM_SIZE;
ContentProps[CONTENT_SIZE_INDEX].type = CMC_uint32;
ContentProps[CONTENT_SIZE_INDEX].value.CMC_pv_uint32 =
                               MessageBody.size;

ContentProps[CONTENT_TITLE_INDEX].property_id =
                               CMC_PV_CONTENT_ITEM_TITLE;
ContentProps[CONTENT_TITLE_INDEX].type = CMC_string;
ContentProps[CONTENT_TITLE_INDEX].value.CMC_pv_string =
                               "Message Body";

ContentProps[CONTENT_ITEMNUM_INDEX].property_id =
                               CMC_PV_CONTENT_ITEM_ITEM_NUMBER;
ContentProps[CONTENT_ITEMNUM_INDEX].type = CMC_uint32;
ContentProps[CONTENT_ITEMNUM_INDEX].value.CMC_pv_uint32 = 0;

ContentProps[CONTENT_ITEMTYPE_INDEX].property_id =
                               CMC_PV_CONTENT_ITEM_ITEM_TYPE;
ContentProps[CONTENT_ITEMTYPE_INDEX].type = CMC_enum;
ContentProps[CONTENT_ITEMTYPE_INDEX].value.CMC_pv_enumerated =
                               CMC_IT_NOTE;

/* Create a Recipient Object. */

Status = pDispatchTable->cmc_open_object_handle(Session,
                                     &Recipient,
                                     CMC_TYPE_OC_RECIPIENT,
                                     NULL);
   /* error handling */
/* Populate the Recipient Object with some properties. */

Status = pDispatchTable->cmc_add_properties(Recipient,
                                     NUM_RECIP_PROPS,
                                     &RecipientProps,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     Recipient,
                                     &error_buf,
                                     NULL);

   /* NOTE - The add properties extension parameter in
      the cmc_add_properties call above could have been
      used for obtaining per property error information. */
   /* error handling */
   }

/* Create a Message Object. */

Status = pDispatchTable->cmc_open_object_handle(Session,
                                     &Message,
                                     CMC_TYPE_OC_MESSAGE,
                                     NULL);
   /* error handling */

/* Populate the Message Object with some properties. */

Status = pDispatchTable->cmc_add_properties(Message,
                                     NUM_MESSAGE_PROPS,
                                     &MessageProps,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     Message,
                                     &error_buf,
                                     NULL);

   /* NOTE – The add properties extension parameter in
      the cmc_add_properties call above could have been
      used for obtaining per property error information. */
   /* error handling */
   }

/* Create a Content Item Object. */

Status = pDispatchTable->cmc_open_object_handle(Session,
                                     &ContentItem,
                                     CMC_TYPE_OC_CONTENT_ITEM,
                                     NULL);
   /* error handling */

/* Populate the Content Item Object with some properties. */

Status = pDispatchTable->cmc_add_properties(ContentItem,
                                     NUM_CONTENT_PROPS,
                                     &ContentProps,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     ContentItem,
                                     &error_buf,
                                     NULL);

   /* NOTE - The add properties extension parameter in
      the cmc_add_properties call above could have been
      used for obtaining per property error information. */
   /* error handling */
   }
```

```
/* Now move the Recipient and Content Item Objects into the
   Message Object. */

Status = pDispatchTable->cmc_copy_object(Message,
                                     Recipient,
                                     &Message,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     Message,
                                     &error_buf,
                                     NULL);
   /* error handling */
   }

Status = pDispatchTable->cmc_copy_object(Message,
                                     ContentItem,
                                     &Message,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     Message,
                                     &error_buf,
                                     NULL);
   /* error handling */
   }

/* Try sending the message. */

Status = pDispatchTable->cmc_send_message_object(Message,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     Message,
                                     &error_buf,
                                     NULL);
   /* error handling */
   }

/* Cleanup. */

cfree(MessageBody.data);

pDispatchTable->cmc_free(ContentItem);
pDispatchTable->cmc_free(Recipient);
pDispatchTable->cmc_free(Message);
pDispatchTable->cmc_free(error_buf);
```

C.4 Check for new messages

```
CMC_return_code              Status = CMC_SUCCESS;
extern                       CMC_session_id Session;
extern                       CMC_dispatch_table *pDispatchTable;
CMC_object_handle            root_object_handle = NULL;
CMC_cursor_handle            RootCursor = CMC_NULL_HANDLE;
CMC_cursor_handle            FolderCursor = CMC_NULL_HANDLE;
CMC_cursor_restriction       RootRestriction;
CMC_string                   error_buf = NULL;
CMC_enum                     InboxTypeFlag = CMC_MCT_INBOX;
CMC_uint32                   NewMessageFlag = CMC_EVENT_NEW_MESSAGES;
CMC_uint32                   MinTimeOut = 0;
CMC_new_message_check_data   CheckData;
CMC_new_message_callback_data*CallbackData = NULL;
CMC_callback                 NewMessageCallBack;
```

```
/*  Get the Root Object Handle. */

Status = pDispatchTable->cmc_get_root_handle(
                                    Session,
                                    &root_object_handle,
                                    NULL);
   /* error handling */

/*  Open a cursor on the Root Container. Start by
   setting up a restriction to find the Inbox Folder.
   Assuming the Existence of single Inbox Folder. */

RootRestriction.type = CMC_RESTRICTION_CONTENT;
RootRestriction.cr.restriction_content.logical = CMC_LOGICAL_EQ;
RootRestriction.cr.restriction_content.property =
                  CMC_PV_MESSAGE_CONTAINER_TYPE;

RootRestriction.cr.restriction_content.property_value =
                  (CMC_buffer)&InboxTypeFlag;
RootRestriction.Property_extensions = NULL;

Status = pDispatchTable->cmc_open_cursor(
                         root_object_handle,
                         &RootRestriction,
                         0,
                         NULL,
                         &RootCursor,
                         NULL);
   /* error handling */

/* Register to poll for new messages */

/* Sets Container for which new messages are checked to the Inbox */

CheckData.number_containers = 1;
CheckData.containers = &RootCursor;

Status = pDispatchTable->cmc_register(
                         Session,
                         NewMessageFlag,
                         NULL,
                         (CMC_buffer) &CheckData,
                         NULL);
   /* error handling */

/* Check for New Messages */

Status = pDispatchTable->cmc_check_event(
                         Session,
                         NewMessageFlag,
                         MinTimeOut,
                         (CMC_buffer) &CheckData,
                         CallBackData,
                         NULL);
   /* error handling */

if (CheckData != NULL) {
   printf("You have new mail!\n");
}

/* Set up callback for new messages */

Status = pDispatchTable->cmc_register_event(
                             Session,
                             NewMessageFlag,
                             NewMessageCallBack,
                             (CMC_buffer) &CheckData,
                             NULL);
   /* error handling */

/* Force callback function */

Status = pDispatchTable->cmc_call_callbacks(
                             Session,
                             NewMessageFlag,
                             NULL);
   /* error handling */
   /* Unregister for the new messages event */

   Status = pDispatchTable->cmc_unregister_event(
                                 Session,
                                 NewMessageFlag,
                                 NewMessageCallBack,
                                 (CMC_buffer) &CheckData,
                                 NULL);
      /* error handling */

   /* Cleanup. */

   pDispatchTable->cmc_free(RootCursor);
   pDispatchTable->cmc_free(FolderCursor);
   pDispatchTable->cmc_free(hFolder);
   pDispatchTable->cmc_free(Inbox);

CMC_return code
(*NewMessageCallBack)(      CMC_session_id     session,
                            CMC_event          event,
                            CMC_buffer         callback_data,
                            CMC_buffer         register_data,
                            CMC_extension      *callback_extensions)
{
   printf("You have new mail!\n");

   pDispatchTable->cmc_free(callback_data);
   pDispatchTable->cmc_free(register_data);

   return(CMC_SUCCESS);
}
```

C.5 Filing a message

```
   #define NUM_RECIP_PROPS                    4
   #define NUM_MESSAGE_PROPS                  5
   #define NUM_CONTENT_PROPS                  6

   #define RECIP_NAME_INDEX                   0
   #define RECIP_ADDRESS_INDEX                1
   #define RECIP_ROLE_INDEX                   2
   #define RECIP_TYPE_INDEX                   3

   #define MSG_TYPE_INDEX                     0
   #define MSG_PRIORITY_INDEX                 1
   #define MSG_SUBJECT_INDEX                  2
   #define MSG_ROLE_INDEX                     3
   #define MSG_CLIENT_MSG_STATUS_INDEX        4

   #define CONTENT_CHARSET_INDEX              0
   #define CONTENT_INFORMATION_INDEX          1
   #define CONTENT_SIZE_INDEX                 2
   #define CONTENT_TITLE_INDEX                3
   #define CONTENT_ITEMNUM_INDEX              4
   #define CONTENT_ITEMTYPE_INDEX             5

   CMC_return_code          Status = CMC_SUCCESS;
   extern                   CMC_session_id Session;
   extern                   CMC_dispatch_table *pDispatchTable;
   CMC_object_handle        root_object_handle = CMC_NULL_HANDLE;
   CMC_object_handle        hFolder = CMC_NULL_HANDLE;
   CMC_object_handle        Message = CMC_NULL_HANDLE;
   CMC_object_handle        Recipient = CMC_NULL_HANDLE;
   CMC_object_handle        ContentItem = CMC_NULL_HANDLE;
   CMC_cursor_handle        RootCursor = CMC_NULL_HANDLE;
   CMC_cursor_restriction   RootRestriction;
   CMC_sint32               FolderCount = 1; /* Assume 1 Drafts Folder. */
   CMC_string               error_buf = NULL;
   CMC_enum                 DraftsTypeFlag = CMC_MCT_INBOX;
   CMC_property             RecipientProps[NUM_RECIP_PROPS];
   CMC_property             MessageProps[NUM_MESSAGE_PROPS];
   CMC_property             ContentProps[NUM_CONTENT_PROPS];
   CMC_opaque_data          MessageBody;
   CMC_CHAR                 MsgBuffer[MAX_BODY_LEN];
```

```
/* Get the Root Object Handle. */

Status = pDispatchTable->cmc_get_root_handle(Session,
                                &root_object_handle,
                                NULL);
   /* error handling */

/* Open a cursor on the Root Container. Start by
   setting up a restriction to find the Drafts Folder.
   Assuming the Existence of the Drafts Folder. */

RootRestriction.type = CMC_RESTRICTION_CONTENT;
RootRestriction.cr.restriction_content.logical = CMC_LOGICAL_EQ;
RootRestriction.cr.restriction_content.property =
                  CMC_PV_MESSAGE_CONTAINER_TYPE;
RootRestriction.cr.restriction_content.property_value =
                  (CMC_buffer)&DraftsTypeFlag;
RootRestriction.Property_extensions = NULL;

Status = pDispatchTable->cmc_open_cursor(root_object_handle,
                         &RootRestriction,
                         0,
                         NULL,
                         &RootCursor,
                         NULL);
   /* error handling */

Status = pDispatchTable->cmc_list_objects(&RootCursor,
                         &FolderCount,
                         &hFolder,
                         NULL);
   /* error handling */

/* Create and populate a Message. */

/* Load Recipient Property Structure. */

RecipientProps[RECIP_NAME_INDEX].property_id = CMC_PV_RECIPIENT_NAME;
RecipientProps[RECIP_NAME_INDEX].type = CMC_string;
RecipientProps[RECIP_NAME_INDEX].value.CMC_pv_string =
                             "Pierre Peret";
RecipientProps[RECIP_ADDRESS_INDEX].property_id = CMC_PV_RECIPIENT_ADDRESS;
RecipientProps[RECIP_ADDRESS_INDEX].type = CMC_string;
RecipientProps[RECIP_ADDRESS_INDEX].value.CMC_pv_string =
                             "uunet!p.peret@A205.bull.com!USENET";
RecipientProps[RECIP_ROLE_INDEX].property_id = CMC_PV_RECIPIENT_ROLE;
RecipientProps[RECIP_ROLE_INDEX].type = CMC_enum;
RecipientProps[RECIP_ROLE_INDEX].value.CMC_pv_enumerated =
                             CMC_RECIPIENT_ROLE_TO;
RecipientProps[RECIP_TYPE_INDEX].property_id = CMC_PV_RECIPIENT_TYPE;
RecipientProps[RECIP_TYPE_INDEX].type = CMC_enum;
RecipientProps[RECIP_TYPE_INDEX].value.CMC_pv_enumerated =
                             CMC_RCT_INDIVIDUAL;

/* Load Message Property Structure. */

MessageProps[MSG_TYPE_INDEX].property_id = CMC_PV_MESSAGE_TYPE;
MessageProps[MSG_TYPE_INDEX].type = CMC_enum;
MessageProps[MSG_TYPE_INDEX].value.CMC_pv_enumerated =
                             CMC_MT_IPM;
MessageProps[MSG_PRIORITY_INDEX].property_id = CMC_PV_MESSAGE_PRIORITY;
MessageProps[MSG_PRIORITY_INDEX].type = CMC_enum;
MessageProps[MSG_PRIORITY_INDEX].value.CMC_pv_enumerated =
                             CMC_PRIORITY_NORMAL;
MessageProps[MSG_SUBJECT_INDEX].property_id = CMC_PV_MESSAGE_SUBJECT;
MessageProps[MSG_SUBJECT_INDEX].type = CMC_string;
MessageProps[MSG_SUBJECT_INDEX].value.CMC_pv_string =
                             "Lunch";
MessageProps[MSG_ROLE_INDEX].property_id = CMC_PV_MESSAGE_ROLE;
MessageProps[MSG_ROLE_INDEX].type = CMC_enum;
MessageProps[MSG_ROLE_INDEX].value.CMC_pv_enumerated =
                             CMC_MESSAGE_ROLE_ORIGINAL;
MessageProps[MSG_CLIENT_MSG_STATUS_INDEX].property_id =
                             CMC_PV_MESSAGE_CLIENT_MSG_STATUS;
MessageProps[MSG_CLIENT_MSG_STATUS_INDEX].type = CMC_enum;
MessageProps[MSG_CLIENT_MSG_STATUS_INDEX].value.CMC_pv_enumerated =
                             CMC_MESSAGE_STATUS_DRAFT;
```

```
/* Load Message Content Item Property Structure. */

ContentProps[CONTENT_CHARSET_INDEX].property_id =
                             CMC_PV_CONTENT_ITEM_CHARACTER_SET;
ContentProps[CONTENT_CHARSET_INDEX].type = CMC_guid;
ContentProps[CONTENT_CHARSET_INDEX].value.CMC_pv_guid =
                             CMC_CHARSET_1252;

ContentProps[CONTENT_INFORMATION_INDEX].property_id =
                             CMC_PV_CONTENT_ITEM_CONTENT_INFORMATION;
ContentProps[CONTENT_INFORMATION_INDEX].type = CMC_opaque_data;

strcpy(MsgBuffer, "What time are we leaving for lunch?");

MessageBody.size = strlen(MsgBuffer) + 1;
MessageBody.data = (CMC_byte *)calloc(1, strlen(MsgBuffer) + 1);

ContentProps[CONTENT_INFORMATION_INDEX].value.CMC_pv_opaque_data.
                             data = MessageBody.data;
ContentProps[CONTENT_INFORMATION_INDEX].value.CMC_pv_opaque_data.
                             size = MessageBody.size;

ContentProps[CONTENT_SIZE_INDEX].property_id =
                             CMC_PV_CONTENT_ITEM_SIZE;
ContentProps[CONTENT_SIZE_INDEX].type = CMC_uint32;
ContentProps[CONTENT_SIZE_INDEX].value.CMC_pv_uint32 =
                             MessageBody.size;

ContentProps[CONTENT_TITLE_INDEX].property_id =
                             CMC_PV_CONTENT_ITEM_TITLE;
ContentProps[CONTENT_TITLE_INDEX].type = CMC_string;
ContentProps[CONTENT_TITLE_INDEX].value.CMC_pv_string =
                             "Message Body";

ContentProps[CONTENT_ITEMNUM_INDEX].property_id =
                             CMC_PV_CONTENT_ITEM_ITEM_NUMBER;
ContentProps[CONTENT_ITEMNUM_INDEX].type = CMC_uint32;
ContentProps[CONTENT_ITEMNUM_INDEX].value.CMC_pv_uint32 = 0;

ContentProps[CONTENT_ITEMTYPE_INDEX].property_id =
                             CMC_PV_CONTENT_ITEM_ITEM_TYPE;
ContentProps[CONTENT_ITEMTYPE_INDEX].type = CMC_enum;
ContentProps[CONTENT_ITEMTYPE_INDEX].value.CMC_pv_enumerated =
                             CMC_IT_NOTE;

/* Create a Recipient Object. */

Status = pDispatchTable->cmc_open_object_handle(Session,
                                    &Recipient,
                                    CMC_TYPE_OC_RECIPIENT,
                                    NULL);

   /* error handling */

/* Populate the Recipient Object with some properties. */

Status = pDispatchTable->cmc_add_properties(Recipient,
                             NUM_RECIP_PROPS,
                             &RecipientProps,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             Recipient,
                             &error_buf,
                             NULL);

   /*   NOTE - The add properties extension parameter in
        the cmc_add_properties call above could have been
        used for obtaining per property error information. */
   /* error handling */
   }
```

```
/* Create a Message Object. */

Status = pDispatchTable->cmc_open_object_handle(Session,
                                     &Message,
                                     CMC_TYPE_OC_MESSAGE,
                                     NULL);
   /* error handling */

/* Populate the Message Object with some properties. */

Status = pDispatchTable->cmc_add_properties(Message,
                             NUM_MESSAGE_PROPS,
                             &MessageProps,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             Message,
                             &error_buf,
                             NULL);
   /*   NOTE - The add properties extension parameter in
        the cmc_add_properties call above could have been
        used for obtaining per property error information. */
   /*   error handling */
   }

/* Create a Content Item Object. */

Status = pDispatchTable->cmc_open_object_handle(Session,
                                     &ContentItem,
                                     CMC_TYPE_OC_CONTENT_ITEM,
                                     NULL);
   /* error handling */

/* Populate the Message Object with some properties. */

Status = pDispatchTable->cmc_add_properties(ContentItem,
                             NUM_CONTENT_PROPS,
                             &ContentProps,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             ContentItem,
                             &error_buf,
                             NULL);

   /*   NOTE - The add properties extension parameter in
        the cmc_add_properties call above could have been
        used for obtaining per property error information. */
   /*   error handling */
   }

/* Now move the Recipient and Content Item Objects into the
   Message Object. */

Status = pDispatchTable->cmc_copy_object(Message,
                             Recipient,
                             &Message,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             Message,
                             &error_buf,
                             NULL);
   /* error handling */
   }
```

```
Status = pDispatchTable->cmc_copy_object(Message,
                             ContentItem,
                             &Message,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             Message,
                             &error_buf,
                             NULL);
   /* error handling */
   }

/* Move message into the Drafts Folder. */

Status = pDispatchTable->cmc_copy_object(hFolder,
                             Message,
                             &Message,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             hFolder,
                             &error_buf,
                             NULL);
   /* error handling */
   }

Status = pDispatchTable->cmc_commit_object(Message,
                             NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                             hFolder,
                             &error_buf,
                             NULL);
   /* error handling */
   }

/* Cleanup. */

cfree(MessageBody.data);
pDispatchTable->cmc_free(ContentItem);
pDispatchTable->cmc_free(Recipient);
pDispatchTable->cmc_free(Message);
pDispatchTable->cmc_free(hFolder);
pDispatchTable->cmc_free(RootCursor);
pDispatchTable->cmc_free(error_buf);
```

C.6 Deleting a message

```
CMC_return_code         Status = CMC_SUCCESS;
extern                  CMC_session_id Session;
extern                  CMC_object_handle root_object_handle;
extern                  CMC_dispatch_table *pDispatchTable;
extern                  CMC_object_handle hFolder;
extern                  CMC_cursor_handle FolderCursor;
CMC_object_handle       hDeletedFolder = CMC_NULL_HANDLE;
CMC_object_handle       Message = CMC_NULL_HANDLE;
CMC_object_handle       MessageInDeleted = CMC_NULL_HANDLE;
CMC_cursor_restriction  RootRestriction;
CMC_cursor_handle       RootCursor = CMC_NULL_HANDLE;
CMC_sint32              FolderCount = 1; /* Assume 1 Drafts Folder. */
CMC_sint32              MessageCount = 1; /* Assume deletion of 1 entry */
CMC_string              error_buf = NULL;
CMC_enum                DeletedTypeFlag = CMC_MCT_DELETED;

/* Open a cursor on the Root Container. Start by
   setting up a restriction to find the Deleted Folder.
   Assuming the Existence of the Deleted Folder. */
```

```
RootRestriction.type = CMC_RESTRICTION_CONTENT;
RootRestriction.cr.restriction_content.logical = CMC_LOGICAL_EQ;
RootRestriction.cr.restriction_content.property =
                  CMC_PV_MESSAGE_CONTAINER_TYPE;
RootRestriction.cr.restriction_content.property_value =
                  (CMC_buffer)&DeletedTypeFlag;
RootRestriction.Property_extensions = NULL;

Status = pDispatchTable->cmc_open_cursor(root_object_handle,
                                     &RootRestriction,
                                     0,
                                     NULL,
                                     &RootCursor,
                                     NULL);
   /* error handling */

Status = pDispatchTable->cmc_list_objects(&RootCursor,
                                     &FolderCount,
                                     &hDeletedFolder,
                                     NULL);
   /* error handling */

/* NOTE - Assuming UI code has set a FolderCursor on a Message entry
   in a list box which is the message to delete. */

/* Get the message to be deleted. */

Status = pDispatchTable->cmc_list_objects(&FolderCursor,
                                     &MessageCount,
                                     &Message,
                                     NULL);
   /* error handling */

/* Let’s first move the message to the Deleted Folder. */

Status = pDispatchTable->cmc_copy_object(hDeletedFolder,
                                     Message,
                                     &MessageInDeleted,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     hFolder,
                                     &error_buf,
                                     NULL);
   /* error handling */
   }

Status = pDispatchTable->cmc_commit_object(MessageInDeleted,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     hDeletedFolder,
                                     &error_buf,
                                     NULL);
   /* error handling */
   }

/* Now let’s permanently delete the message from the source folder.
   Invalidates Message handle. */

Status = pDispatchTable->cmc_delete_objects(MessageCount,
                                     &Message,
                                     NULL);

if (Status != CMC_SUCCESS)
   {
   pDispatchTable->cmc_get_last_error(Session,
                                     Message,
                                     &error_buf,
                                     NULL);
   /* error handling */
   }
```

```
/* Cleanup. */

pDispatchTable->cmc_free(hDeletedFolder);
pDispatchTable->cmc_free(MessageInDeleted);
pDispatchTable->cmc_free(RootCursor);
pDispatchTable->cmc_free(error_buf);
```

C.7 Retrieving a message

```
#define MAX_CACHE              25

CMC_return_code         Status = CMC_SUCCESS;
extern                  CMC_session_id Session;
extern                  CMC_dispatch_table *pDispatchTable;
CMC_object_handle       root_object_handle = NULL;
CMC_object_handle       hFolder = NULL;
CMC_object_handle       Messages = NULL;
CMC_cursor_handle       RootCursor,
CMC_cursor_handle       FolderCursor,
CMC_cursor_restriction  RootRestriction;
CMC_sint32              FolderCount = 1;
CMC_sint32              MessageCount = MAX_CACHE;
CMC_enum                InboxTypeFlag = CMC_MCT_INBOX;
CMC_enum                MessageClassFlag = CMC_TYPE_OC_MESSAGE;

/* Get the Root Object Handle. */

Status = pDispatchTable->cmc_get_root_handle(Session,
                                     &root_object_handle,
                                     NULL);
   /* error handling */

/* Open a cursor on the Root Container. Start by
   setting up a restriction to find the Inbox Folder.
   Assuming the Existence of single Inbox Folder. */

RootRestriction.type = CMC_RESTRICTION_CONTENT;
RootRestriction.cr.restriction_content.logical = CMC_LOGICAL_EQ;
RootRestriction.cursor_restriction.restriction_content.property =
                  CMC_PV_MESSAGE_CONTAINER_TYPE;
RootRestriction.cursor_restriction.restriction_content.property_value =
                  (CMC_buffer)&InboxTypeFlag;
RootRestriction.Property_extensions = NULL;

Status = pDispatchTable->cmc_open_cursor(root_object_handle,
                             &RootRestriction,
                             0,
                             NULL,
                             &RootCursor,
                             NULL);
   /* error handling */

Status = pDispatchTable->cmc_list_objects(&RootCursor,
                             &FolderCount,
                             &hFolder,
                             NULL);
   /* error handling */

/* Build a restriction on the Folder. Fetch all messages. */

FolderRestriction.type = CMC_RESTRICTION_CONTENT;
FolderRestriction.cr.restriction_content.logical = CMC_LOGICAL_EQ;
FolderRestriction.cr.restriction_content.property =
                  CMC_PV_OBJECT_CLASS;
FolderRestriction.cr.restriction_content.property_value =
                  (CMC_buffer)&MessageClassFlag;

/* Open a cursor on the Folder. */

Status = pDispatchTable->cmc_open_cursor(hFolder,
                             &FolderRestriction,
                             0,
                             NULL,
                             &FolderCursor,
                             NULL);
   /* error handling */
/* Enumerate all the messages in the inbox in chunks of MAX_CACHE. */

while (MessageCount != 0)
   {
Status = pDispatchTable->cmc_list_objects(&FolderCursor,
                             &MessageCount,
                             &Messages,
                             NULL);
   /* error handling */

/* Build Property array of desired properties (see composing a
   message example), call cmc_read_properties and Display in
   Listbox for each Message Object returned.

   NOTE - The individual object handles need to be copied by a call to
   cmc_copy_object_handle() prior to invoking cmc_free() on this pointer. */

pDispatchTable->cmc_free(Messages);

}

/* Cleanup. */

pDispatchTable->cmc_free(RootCursor);
pDispatchTable->cmc_free(FolderCursor);
pDispatchTable->cmc_free(hFolder);
```

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks and open system communication**
Series Z	Programming languages