A. P. I. 3000

Application Programming Interface

for

HP3000s running MPE/iX

Reference Manual

3k Associates, Inc.

6901 Old Keene Mill Rd, Suite 500
Springfield, VA 22150
(703) 569-9189
Fax: (703) 451-3720
Sales@3k.com
E-Mail: Support@3k.com

NOTICE

3k Associates, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. 3k Associates, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

The information contained in this document is subject to change without notice.

NOTE: Common Messaging Call API, Version 1.0 (June 1, 1993) reprinted with permission from X.400 API Association. All additions made by 3k Associates, Inc. appear with gray, shaded text and a "  **" symbol.**

PRINTING HISTORY

First Printing ..... May, 1997

Second Printing ..... April, 1998

Table of Contents

Introduction

From the people that brought you the first SMTP/MIME e-mail system for the HP3000, an easy to use standards based programmatic interface that allows you to mail-enable your HP3000-based applications.

A.P.I./3000 is a library of procedure calls that make it easy for your application to send a message (including file attachments), retrieve messages from a mailbox, and interrogate the mail directory to verify e-mail addresses. And, since the API accesses our SMTP/MIME messaging engine, your messages are automatically MIME compatible.

- Ideal as a transport for EDI or other transaction data
- Your applications can transmit data as e-mail messages to other systems or trading partners
- Store and forward SMTP-compatible message engine ensures delivery to any other SMTP compatible system
- User-defined scripts can be attached to mailboxes to invoke your applications automatically upon receipt of inbound mail messages - passing the message contents to your application for processing
- Simple programmatic calls to send messages (with attachments allowed), retrieve messages from a mailbox, and interrogate the mail system directory
- Example C, COBOL, and SPLash! programs provided

The CMC (Common Messaging Call) standard was designed by the XAPI group as a standardized way of mail-enabling applications on a wide variety of platforms. Up to that point, shops with a variety of platforms had to learn different proprietary interfaces to mail enable their applications and the same application when ported to different platforms would require significant code changes to work with the mail system on the new platform. With the CMC standard interface, programmers need only learn the standard interface routines (as documented in the standard specifications) and could not only leverage this knowledge across multiple platforms, but could now easily port their applications to new platforms without having to re-code the mail interface. Now, with this interface, mail-enabled applications written on the HP3000 platform can potentially be easily ported to other platforms, and mail-enabled applications written to the CMC standard for other platforms can be ported to the HP3000.

We provide a copy of the CMC specification in this manual for your convenience (reprinted with permission).

Installation Instructions

From Tape

All 3k software products install automatically by simply restoring one job stream from the installation tape, inserting appropriate passwords into it and streaming it. The installation job automatically determines whether you are in stalling the software for the first time or are updating to a new software release. The steps are:

Note: If this is an update (you already have an existing version) verify that there are no users accessing files in the THREEK account. This account may include other 3k products: ***NetMail/3000, Pop Server/3000, DeskLink, A.P.I./3000 (Application Programming Interface),*** an HP3000 Client for ***Office Exent Fax,*** or ***Office Extend FTP*** software. Once verified, mount a blank tape or DAT and :STORE @.@.THREEK before you begin the new installation.

1. Log on as MANAGER.SYS
2. Issue a file equation for your tape drive: :FILE THREEK;DEV=TAPE (for magnetic tape or DAT users) (or) :FILE THREEK;DEV=CTAPE (for cartridge tape systems)
3. RESTORE *THREEK;THREEKLD.PUB.SYS;SHOW (You should see one file restored.)
4. Use your favorite text editor and modify the first line of the file (THREEKLD.PUB.SYS) to include the appropriate passwords, -OR- simply remove the passwords from MANAGER.SYS,PUB for the duration of the installation process and replace them when done. If you have a third party security system installed, make sure you have enabled logon access for MGR.THREEK and MANAGER.SYS.
5. STREAM THREEKLD.PUB.SYS (You will see informative messages on the console reporting the progress of the software installation or update.)

When the job(s) have completed, you will see a message reporting that the software was successfully installed. If this is a DEMO version you downloaded over the Internet, you now need to ACTIVATE your software (see Step 7). **Demos received on tape or DAT are already activated and ready to run at this point.**

To activate a demo:

call the 3k Associates sales office at:
(US/Canada) 1-800 NetMail (800 638-6245)
(Other countries) +1 703 569-9189

Personnel are available 9AM-8PM Eastern (US) time.

CMC Standard Specifications

Following this page is the actual XAPI group's CMC (Common Messaging Call) standard application programming interface. This interface was designed to be portable to as many systems as possible, yet is designed to accommodate individual system extensions. The standard is similar to the windows-based "MAPI" standard, with the obvious exception that it is not dependent on windows-style frames and boxes.

Special implementation notes and extensions applicable to the HP3000 based implementation are noted with 3k logo marks and are shaded.

X.400 API Association Specification

COMMON MESSAGING CALL API

Version 1.0

June 1, 1993

X.400 API Association

Any comments relating to the material contained in this document may be submitted to XAPIA at:

X.400 API Association
2141 Landings Drive
Mountain View, CA 74043

Contributing Editors:

John Ankorn, CTC
Kurt Christofferson and Pat Reilly, RAM Mobile Data
Rodolphe Goldstein, First Telecom
Stephen J. Griesmer, AT&T
Keith Grochow and Jonathan Kauffman, Microsoft
Chris Harding, Datalogic
Ken Krechmer, Action Consulting
Brian Lambert, Iris Associates
Sue Klein Lebeck, Tandem Computers, Inc.
Joseph M. Mansur, Digital Equipment Corporation
Pauline Moller, OSIWare
Pierre Perret and Daniella Sirocchi, Bull
Elena Seifrid, Retix
Enzo Signore, ISOCOR
Jack Skinner, IBM
Mike Timms, Novell
Mike Weston and Tom Brant, cc:Mail/Lotus

1. Introduction

This chapter introduces the Common Messaging Call Application Program Interface 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.

1.1. Purpose

The purpose of this document is to specify a high-level messaging application program interface (API) 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 *mail-enabled applications*.

This document is directed toward messaging service developers who might wish to support such an application program interface. This document may also guide application developers in understanding implementation-independent features of the 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 (CMC API) provides a set of high-level functions for mail-enabled applications to send and receive electronic messages.

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 (RFC822) 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 minimize the number of function calls needed to send or receive a message. For example a mail-enabled application can send a message with a single function call and receive a specific message with two calls.

The CMC API is designed to be complementary to existing XAPIA-X/OPEN API's 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 XAPIA will define the common mapping between CMC and the X.400 protocol, from the perspective of other XAPIA-defined X.400 APIs.

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. This list is not comprehensive:

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

1.3. Abbreviations

The following abbreviations are used in this document.

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
T.611	CCITT API for use with facsimile, telex, and teletex services

1.4. Referenced Documents

This section identifies other documents on which this document relies.

ANSI C	American National Standard for Information Systems - Programming Language C, X3.159-1989.
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.
X.208	CCITT Recommendation X.208, "Specification of Abstract Syntax Notation One (ASN.1), 1992.

1.5. Levels

This document 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 Section 3, the CMC data structures are described generically, but include a C declaration. In Section 4, the CMC functions are specified generically, but include a synopsis written in C. For clarity, constants and error codes throughout this specification are written in the C syntax described below. Section 4.6 gives a summary of the C declarations and constants used throughout the specification.

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 1-1 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.

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
Reserved for extension sets	CMC_XS_	any
Reserved for extensions	CMC_X_	any
Reserved for use by implementors	CMCP	any

Table 1-1: Derivation of C Naming Conventions

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.

1.7. Conformance

In order for an implementation of the Common Messaging Calls API to conform to this specification it must meet the following criteria:

- All functions and data structures must be implemented as defined. Statements elsewhere in the specification which describe features as optional or with exceptions take precedence over this criterion.
- The implementation must be able to send and receive at least the CMC IPM message type.
- 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 document. 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 be registered with the XAPIA.

- Minimum conformance for an extension set will be defined by the creator of the extension set.

2. Functional Architecture

This chapter describes the functional architecture of services supporting the CMC API. It provides a model and a functional overview along with a discussion of sessions, configuration, and extensions.

2.1. Model

The CMC interface is defined between a mail-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 SMTP. 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 the use of extensions (see **Section 2.5, Extensions**). The CMC interface is depicted in Figure 2-1 below.

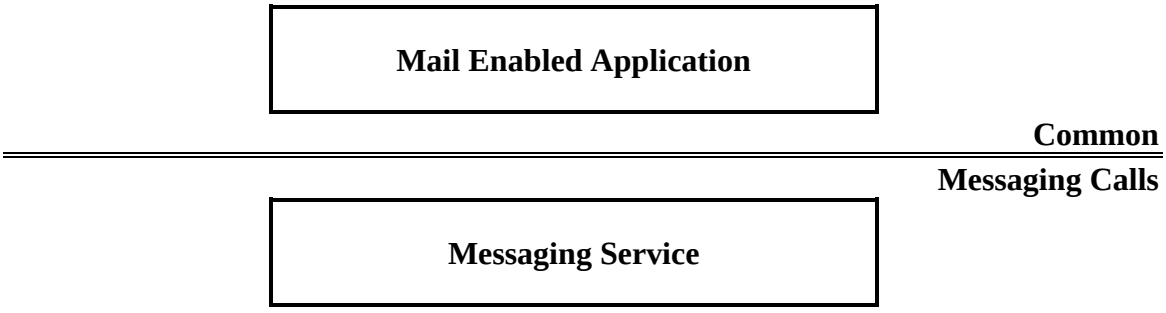

Figure 2-1: Positioning of the Common Messaging Call API.

The model of the CMC interface can be divided into three components: a directory, a submission queue, and a receiving mailbox. These components are shown in Figure 2-2.

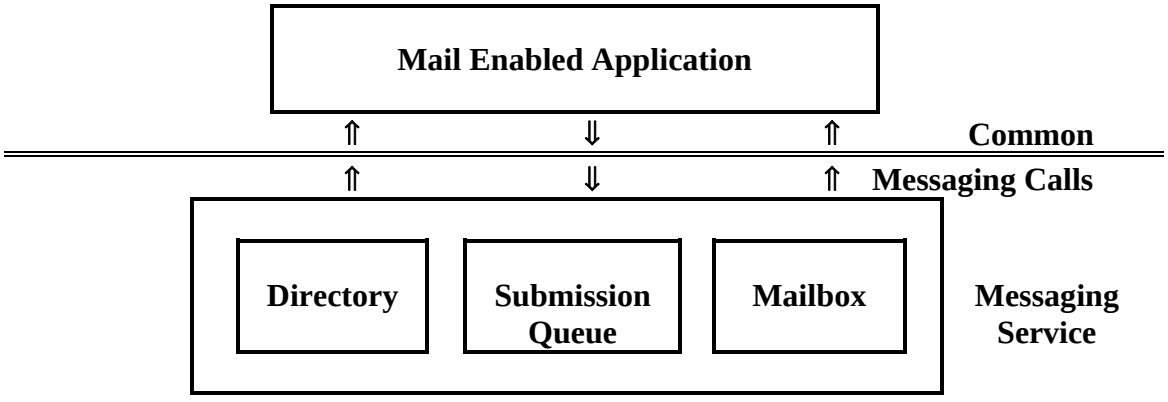

Figure 2-2: Model of the Common Messaging Call API.

There is a submission queue for each mail-enabled application. The CMC model provides for synchronous submission of mail. Once the application has completed a send call, all responsibility for the message is transferred to the CMC implementation.

On the receiving side, all messages are delivered to a mailbox for a user. Mailboxes are maintained on behalf of messaging users and are accessible by users of mail-enabled applications with the proper permissions. With the CMC API, the application can retrieve selective summaries of the contents of a mailbox along with identifiers for the particular messages. These identifiers can then be used to select and read individual messages.

The directory allows the mail enabled application to look up information about users of the messaging service. The directory will allow resolution of user names to addresses. Some services may also provide a user interface to create recipient lists for messages or to find out details about a specific recipient.

2.2. Functional Overview

The CMC interface supports three principle tasks: sending messages, reading messages, and looking up addressing information.

To send a message, the mail-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 CMC Send() function. An application submits a message to the submission queue through a CMC Send() function. The mail-enabled application is responsible for populating the CMC message structure used in the CMC Send() function. The mail-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 mail-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. CMC Act On() 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.

To look up names in the directory, the mail-enabled application establishes a session through the CMC Logon() function or interactively by setting the LOGON_UI_ALLOWED flag in the CMC Look Up() function. The application then uses CMC Look Up() to translate a user-friendly name into a messaging address. This function also allows the application to request UI to create addressing lists or recipient specific details. 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.

2.3. Session

CMC function calls occur within the context of a session. A session is established with a CMC Logon() call and terminated with a CMC Logoff() call. The CMC Logon() call also authenticates the user to the messaging service and sets session attributes. Session attributes include character set and version number. Currently, there is no support for sharing sessions among applications.

2.4. Configuration

The persistent configuration of the service is available for query by the mail-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 specification.

2.5. Extensions

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

Extensions that are additional parameters to a function call may be input or output. That is, the extension may be passed as input parameters from the application to the 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 CMC Free() call.

Extensions play a dual role in this specification. First, they provide a mechanism whereby features not common across all messaging services can be accommodated. Second, they provide a mechanism to extend the specification 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 document, see the appendices.

3. Data Structures

This section defines, and Table 3-1 lists, the data structures used in the CMC API.

Data Type Name	Description
Attachment	Message attachment structure
Boolean	A value that indicates logical true or false
Buffer	Pointer to a data item
Counted String	String with an explicit length designation
Enumerated	Data type containing a value from an enumeration
Extension	Extension structure
Flags	Container for flag bits
Message	Message structure
Message Reference	Message Reference structure
Message Summary	Message summary structure
Object Identifier	Object Identifier structure
Recipient	Originator/recipient structure
Return Code	Return value indicating either that a function succeeded or why it failed
Session ID	Unique identifier for session
String	Character string pointer
Time	Time structure
User Interface ID	User interface handle

Table 3-1: CMC Data Structures

3.1. Basic Data Types

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

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., byte                CMC_sint8;
typedef system-defined, e.g., int                 CMC_sint16;
typedef system-defined, e.g., long int            CMC_sint32;
typedef system-defined, e.g., unsigned int        CMC_uint16;
typedef system-defined, e.g., unsigned long int   CMC_uint32;
```

3.2. 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. 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 Section 3.12. 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.

The current release of ***A.P.I./3000*** only supports file names in MPE filespace.

4. *attach_flags*. Bits for boolean attributes. Unused bits must be clear.

   1. 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.

   2. 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.

3.3. 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.

3.4. 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.

3.5. 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.

Currently, A.P.I./3000 does not support counted strings.

3.6. 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.

3.7. 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 document, see Appendix B.

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.

    1. CMC_EXT_REQUIRED
       Set: Return an error if this extension cannot be supported.
       Clear: Allow "best effort" support, including no support, of this extension.

    2. 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.

    3. 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.

Currently, ***A.P.I./3000*** does not support any extensions. You may pass them, but they will be ignored. If the CMC_EXT_REQUIRED flag is set on any of them, the CMC_E_UNSUPPORTED_FUNCTION_EXT error will be returned.

3.8. 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.

3.9. 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. 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 CCITT Recommendation X.208.
   b. CMC Registered Values - used for types defined in this specification.
   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 1 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"
```

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.

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 messages 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 whitespace
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.

A.P.I./3000 currently treats all messages as CMC: IPM (no special optimizations are done).

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 that 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.

   1. CMC_MSG_READ
      Set: Message has been read.
      Clear: Message has not been read.

   2. 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.

   3. 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.

   4. 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_extensions*. Pointer to first element in array of per-message extensions.

3.10. 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. 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.

3.11. 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. 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.

    1. CMC_SUM_READ
       Set: Message has been read.
       Clear: Message has not been read.

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

3. CMC_SUM_LAST_ELEMENT
   Set: Identifies the last structure in an array of such structures.
   Clear: This is not the last array element.

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

3.12. 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 CCITT Recommendation X.208. It is globally unambiguous. Its syntax as used in this specification shall match the Number form in 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.

3.13. 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 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 distribution list.

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 specification. 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 the 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.

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 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.

1. 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.

2. 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.

3. 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.

3.14. 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 specification. Values contained within the high order 16 bits are reserved for implementation defined error codes while the low order 16 bits should be set to an appropriate CMC error.

Errors may be resolved within the scope of a CMC call using, for example, dialogs available through the user interface. If a dialog is invoked to resolve the error, but the error remains unresolved after the dialog 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.

3.15. Session ID

NAME

Session ID - type definition for a CMC session ID.

C DECLARATION

typedef system-defined, e.g., uint32 CMC_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 Appendix D for the definition for a specific platform.

3.16. String

NAME

String - type definition for a CMC character string.

C DECLARATION

```
typedef  char *    CMC_string;
```

DESCRIPTION

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

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.

3.17. 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. 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 that 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.

3.18. User Interface ID

NAME

User Interface Identifier - type definition for a CMC user interface handle.

C DECLARATION

typedef system-defined, e.g., uint32 CMC_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 behavior defined by the implementation.
Note that 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.

A.P.I./3000 does not currently provide a UI.

See Appendix D for the definition for a specific platform.

4. Functional Interface

This section 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 **Section 4.6, Declaration Summary**. Table 4-1 lists the functions of the CMC interface.

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.
Logoff	Terminate a session with the messaging service.
Logon	Establish a session with the messaging service.
Query Configuration	Determine information about the installed CMC service.

Table 4-1: CMC Interface Functions

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

4.1. Sending Messages

4.1.1. 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. It can, at the option of the caller, either prompt via a user interface (e.g. a dialog box) for message creation or proceed without any user interaction.

The caller can optionally provide a list of recipients, subject text, attachments and/or note text. If required message elements are not provided, the function can prompt the user for them if UI is specified and supported. If one or more recipients are provided, the function can send the message without prompting the user. If the optional parameters are specified and a dialog box is requested, the parameters provide the initial values for the dialog box.

The successful return of this function does not necessarily imply the validation of recipients.

ARGUMENTS

Session (Session Id)

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

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

If the session handle 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 flag CMC_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 dialog box to prompt for logon if required. If not set, the function will not display a dialog 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 dialog box to prompt for recipients, the message fields, and other sending options. If not set, the function will not display a dialog box, but at least one recipient must be specified.

CMC_ERROR_UI_ALLOWED - Set if the function may display a dialog box on encountering recoverable errors. If not set, the function may not display a dialog box and will simply return an error code.

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. dialog 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

4.1.2. Send Documents

NAME

Send Documents- string based function to send mail.

SYNOPSIS

```
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 dialog.

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_LOGON_UI_ALLOWED
CMC_SEND_UI_REQUESTED
CMC_ERROR_UI_ALLOWED
CMC_COUNTED_STRING_TYPE
CMC_FIRST_ATTACH_AS_TEXT_NOTE
```

CMC_LOGON_UI_ALLOWED - Set if the function should display a dialog box to prompt for logon if required. If not set, the function will not display a dialog box and will return the error CMC_E_USER_NOT_LOGGED_ON if the user is not logged on.

CMC_SEND_UI_REQUESTED - Set if the function should display a dialog UI to prompt for recipients, the message fields, and other sending options. If not set, the function will not display a dialog box, but at least one recipient must be specified.

CMC_ERROR_UI_ALLOWED - Set if the function may display a dialog box on encountering recoverable errors. If not set, the function may not display a dialog box and will simply return an error code.

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 FilePaths, 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. dialog 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

4.2. Receiving Messages

4.2.1. 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 handle which represents a session with the messaging service.

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

If the session handle is invalid, then the error 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 dialog box on encountering recoverable errors. If not set, the function may not display a dialog box and will simply return an error code.

UI Id (UI Id)

User Interface handle (e.g. dialog 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

4.2.2. 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 handle which represents a session with the messaging service.

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

If the session handle is invalid, then the error 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 Section 3.9.

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 dialog box on encountering recoverable errors. If not set, the function may not display a dialog 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. dialog 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 are 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 that 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

4.2.3. 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 handle which represents a session with the messaging service.

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

If the session handle is invalid, then the error 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 dialog box on encountering recoverable errors. If not set, the function may not display a dialog 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 that 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's marked as read or not.

UI Id (UI Id)

User Interface handle (e.g. dialog 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

4.3. Looking Up Names

4.3.1. Look Up

NAME

Look Up - Looks 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 resolves a friendly name to an address, optionally prompting the user to choose among multiple resolved names or addresses when necessary if UI is specified and supported. It can also be used to display UI for creation of address lists or displaying details about a recipient.

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 handle which represents a session with the messaging service.

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

If the session handle 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 dialog box to prompt for logon if required. If not set, the function will not display a dialog 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 dialog box on encountering recoverable errors. If not set, the function may not display a dialog 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 dialog 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. dialog 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

4.4. Administration

4.4.1. 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_read(), cmc_list(), cmc_query_configuration(),** and **cmc_look_up()**.

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 section 5 **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

4.4.2. Logoff

NAME

Logoff - log off the CMC service.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_logoff(
        CMC_session_idsession,
        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.

ARGUMENTS

Session (Session Id)

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

If the session handle 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 dialog box on encountering recoverable errors. If not set, the function may not display a dialog 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

4.4.3. 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. It can, at the option of the caller, either prompt via a user interface (e.g. a dialog box) if UI is supported by the implementation or proceed without any user interaction.

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 or if UI is allowed. 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 specification is 1.00 and is represented as the value 100.

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_LOGON_UI_ALLOWED - Set if the function should display a dialog box to prompt for logon if required. If not set, the function will not display a dialog box and will return an error if not enough information has been supplied.

CMC_ERROR_UI_ALLOWED - Set if the function may display a dialog box on encountering recoverable errors. If not set, the function may not display a dialog box and will simply return an error code.

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.

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 see the common extensions section of the extensions appendix in this document.

RESULTS

Session (Session Id)

Opaque session handle 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

4.4.4. Query Configuration

NAME

Query Configuration - Determine information about the installed CMC configuration.

SYNOPSIS

```
#include <xcmc.h>

CMC_return_code
cmc_query_configuration(
        CMC_session_idsession,
        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 that the configuration may not be changed through CMC, and that any underlying configuration file format is implementation dependent.

ARGUMENTS

Session (Session Id)

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

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

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

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

If the session handle 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 chapter in the appendix 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 logon, CMC_REQUIRED_OPT if the password is optional to logon, or CMC_REQUIRED_YES if the password is required to logon.

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 logon, CMC_REQUIRED_OPT if the service name is optional to logon, or CMC_REQUIRED_YES if the service name is required to logon.

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 logon, CMC_REQUIRED_OPT if the user name is optional to logon, or CMC_REQUIRED_YES if the user name is required to logon.

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 log on.

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 section of the extensions appendix in this document

RESULTS

Reference (Buffer)

This argument points to the buffer in which to receive the configuration information. The 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

4.4.5. COBOL Memory Cell Read

NAME

COBOL Memory Cell Read - Move data from memory to COBOL WORKING-STORAGE.

SYNOPSIS

```
#include <xcmc.h>

CMC_sint16
cmc_cobol_cell_read(
        CMC_sint32          cell_address,
        CMC_buffer          cobol_data,
        CMC_sint16          max_length,
        CMC_boolean         binary_data
);
```

DESCRIPTION

This function allows a COBOL program to get data that has been stored in a block of memory allocated by the malloc() call. The COBOL program will pass the pointer address in cell_address, the pointer to a WORKING-STORAGE area in cobol_data, the maximum length of bytes to be returned in max_length, and CMC_true in binary_data if the move is not to be terminated by encountering a NULL character. If binary_data is set to CMC_true, maxlength bytes will always be returned. Otherwise, data transfer will stop when a NULL is encountered, with space padding if max_length has not been reached..

ARGUMENTS

cell_address (sint32)

Pointer to a memory cell returned by the malloc() call..

Some CMC calls (cmc_query_configuration for one) return memory cells allocated by malloc(). These cells are not directly addressable by HP COBOL programs. This parameter is the 32-bit address that was returned to the COBOL program.

max_length(sint16)

The maximum number of bytes to be transferred in non-BINARY mode, or the exact number of bytes to be transferred in ASCII mode.

binary_data(Boolean)

A true/false value indicating if the data is to be treated as BINARY.

If true, the procedure will move exactly max_length bytes. Otherwise, it will move bytes until a NULL is found, or max_length bytes have been moved. If the data is not BINARY, and a NULL is found before max_length bytes have been moved, the remaining area of cobol_data will be padded with spaces.

RESULTS

cobol_data(Buffer)

Pointer to an area of WORKING-STORAGE.

This function does no type or bounds checking, so it is up to the caller to ensure that there is sufficient room for the data requested.

bytes_transferred (sint16)

This is the number of bytes that were transferred. In BINARY mode, this is always the same as max_length, but in non-BINARY mode, is the number of bytes moved before a NULL was encountered. The remainder of the cobol_data buffer will be padded with spaces.

ERRORS

NONE

4.5. Return Codes

This section 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 section 4.6 **C Declaration Summary**. Table 4-2 lists the generic return codes and the functions to which the return codes pertain. Following the table, 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.

Return Code	Act	Free	List	Logoff	Logon	Query	Read	Look	Send	SndDoc
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_COUNTED_STRING_UNSUPPORTED	-	-	-	-	X	-	-	-	-	-
CMC_E_DISK_FULL	-	-	-	-	-	-	X	-	-	-
CMC_E_FAILURE	X	X	X	X	X	X	X	X	X	X
CMC_E_INSUFFICIENT_MEMORY	X	-	X	X	X	X	X	X	X	X
CMC_E_INVALID_CONFIGURATION	-	-	-	-	X	-	-	-	-	-
CMC_E_INVALID_ENUM	X	-	-	-	X	X	-	-	-	-
CMC_E_INVALID_FLAG	X	-	X	X	X	-	X	X	X	-
CMC_E_INVALID_MEMORY	-	X	-	-	-	-	-	-	-	-
CMC_E_INVALID_MESSAGE_PARAMETER	-	-	-	-	-	-	-	-	X	-
CMC_E_INVALID_MESSAGE_REFERENCE	X	-	X	-	-	-	X	-	-	-
CMC_E_INVALID_PARAMETER	X	X	X	X	X	X	X	X	X	X
CMC_E_INVALID_SESSION_ID	X	-	X	X	-	-	X	-	-	-
CMC_E_INVALID_UI_ID	X	-	X	X	X	-	X	X	X	X
CMC_E_LOGON_FAILURE	-	-	-	-	X	-	-	X	X	X
CMC_E_MESSAGE_IN_USE	X	-	-	-	-	-	-	-	-	-
CMC_E_NOT_SUPPORTED	-	-	-	-	-	X	-	X	-	-
CMC_E_PASSWORD_REQUIRED	-	-	-	-	X	-	-	-	-	-
CMC_E_RECIPIENT_NOT_FOUND	-	-	-	-	-	-	-	X	X	X
CMC_E_SERVICE_UNAVAILABLE	-	-	-	-	X	-	-	-	-	-
CMC_E_TEXT_TOO_LARGE	-	-	-	-	-	-	-	-	X	X
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_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_VERSION	-	-	-	-	X	-	-	-	-	-
CMC_E_USER_CANCEL	-	-	-	-	-	-	-	X	X	X
CMC_E_USER_NOT_LOGGED_ON	-	-	-	X	-	-	-	X	X	X

Table 4-2: CMC Interface Return Codes

The return codes are defined as follows:

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_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_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_ENUM	A CMC_enum value is invalid.
CMC_E_INVALID_FLAG	A flag value in the flags parameter was 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_PARAMETER	A function parameter was invalid.
CMC_E_INVALID_SESSION_ID	The specified session id is invalid or no longer valid (e.g., after logging off).
CMC_E_INVALID_UI_ID	The specified user interface id is invalid or no longer 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_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_RECIPIENT_NOT_FOUND	One or more of the specified recipients were not found.
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_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_DO_NOT_MARK_AS_READ flag cannot be supported.
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_VERSION	The version specified in the call cannot be supported by this CMC implementation.
CMC_E_USER_CANCEL	The operation was canceled by the user.
CMC_E_USER_NOT_LOGGED_ON	The user is not logged on and the CMC_LOGON_UI_ALLOWED flag is not set.

4.6. C Declaration Summary

This section 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 **Chapter 3, Data Structures** or **Section 4.1, Interface Functions**.

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 INTERFACE */

/*BASIC DATA TYPES*/

#ifndef DIFFERENT_PLATFORM
typedef byte                  CMC_sint8;
typedef int                   CMC_sint16;
typedef long int              CMC_sint32;
typedef unsigned int          CMC_uint16;
typedef unsigned long int     CMC_uint32;
typedef void *             CMC_buffer;
typedef char *             CMC_string;
typedef CMC_uint32            CMC_session_id;
typedef CMC_uint32            CMC_ui_id;
#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;

#define CMC_FALSE ((CMC_boolean)0)
#define CMC_TRUE ((CMC_boolean)1)

/*DATA STRUCTURES*/

/*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)
```

```
/* ATTACHMENT OBJECT IDS */
#define CMC_ATT_OID_BINARY        "1 2 840 113658 1 1"
#define CMC_ATT_OID_TEXT          "1 2 840 113658 1 1 0"

/*COUNTED STRING*/
typedef struct {
   CMC_uint32        length;
   char              string[1];
} CMC_counted_string;

/*EXTENSION*/
typedef struct {
   CMC_uint32           item_code;
   CMC_uint32           item_data;
   CMC_buffer           item_reference;
   CMC_flags            extension_flags;
} CMC_extension;

/* 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)

/*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_LAST_ELEMENT              ((CMC_flags) 0x80000000)

/*MESSAGE REFERENCE*/
typedef  CMC_counted_string   CMC_message_reference;

/*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_LAST_ELEMENT            ((CMC_flags) 0x80000000)

/*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)

/* 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)

/*TIME*/
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;

/* TIME FLAGS */
#define CMC_NO_TIMEZONE                 ((CMC_flags) 0x8000)
```

```
/*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)

/*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
);

#define CMC_SEND_UI_REQUESTED             ((CMC_flags) 1)

/*SEND DOCUMENTS*/
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,
);

#define CMC_FIRST_ATTACH_AS_TEXT_NOTE     ((CMC_flags) 2)


/*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
);

#define CMC_ACT_ON_EXTENDED               ((CMC_enum) 0)
#define CMC_ACT_ON_DELETE                 ((CMC_enum) 1)

/*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
);

#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*/
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
);

#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)



/*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
);

#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)


/*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
);

#define CMC_LOGOFF_UI_ALLOWED            ((CMC_flags) 1)


/* 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
);

#define CMC_VERSION     ((CMC_uint16) 100)

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

#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)


/* DEFINED OBJECT ID'S FOR CHARACTER SETS */
#define CMC_CHARSET_437       "1 2 840 113658 1 2 437"
#define CMC_CHARSET_850       "1 2 840 113658 1 2 850"
#define CMC_CHARSET_1252      "1 2 840 113658 1 2 1252"
#define CMC_CHARSET_ISTRING   "1 2 840 113658 1 3 0"
#define CMC_CHARSET_UNICODE   "1 2 840 113658 1 3 1"
#define CMC_CHARSET_T61       "0 0 20 61"
#define CMC_CHARSET_IA5       "0 0 20 50"
#define CMC_CHARSET_ISO_10646 "2 1 0 0 0"
#define CMC_CHARSET_ISO_646   "1 0 646"


/* RETURN CODES FLAGS */
#define CMC_ERROR_UI_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_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)
```

5. Programming Examples

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 handle.      */
              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 */
```

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_type  = 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 */
```

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 handle.               */
            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);
```

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 handle.        */
             &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 handle.         */
             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);
```

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 handle.      */
              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 */
```

6. Appendices

Appendix A Extension Registration

A set of common extensions are defined by this specification, and vendor specified extensions may be defined by any implementor of the CMC API. Further extension sets may also be defined by future versions of this specification. Because of this, it is important to have a set of guidelines for the naming and definition of extensions. These guidelines are given below:

1. 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 specification will be allocated an extension set number and prefix from the X.400 API Association. Implementors may also obtain an extension set prefix, and a block of extension codes, from the X.400 API Association by requesting such a number in writing. Pre-defined extension set numbers are given in Appendix 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 implementor. No registration of a extension set number is required. This set is provided so that implementors may define extensions without any formal registration. Because of this freedom, extensions from different vendors may conflict and inhibit application portability and the co-residency of different CMC implementations. The prefix for these extensions will be CMC_X_BLT_ and the corresponding set identifier is CMC_XS_BLT.

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.

Appendix B Common Extension Set

The Common Messaging Calls common extension set contains those function and data extensions that are common to most mail services, but are not in the base specification for various reasons. After the documentation for all of the extensions, a C declaration section is provided as the basis of a header file for this extension set. This section should be used as a model for creation of other extension sets. Explanations of extensions and extensions structures are provided in sections 2.5 and 3.7.

CMC_XS_COM

This extension identifier is used to represent all the extensions in the common extension set.

This identifier should be used with the CMC_X_COM_SUPPORT_EXT extension (described below) on cmc_query_configuration() and cmc_logon() to determine support for the common extension set. By asking the implemtation if it supports the entire common extension set, the application does not need to individually request all the extensions it might be interested in. If used during cmc_logon() it will also indicate data extensions that should be attached to the structures for this session (as described below). The implementation should return the support level based on the description in the CMC_X_COM_SUPPORT_EXT extension.

FUNCTION EXTENSIONS

CMC_X_COM_SUPPORT_EXT

Description:

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 that some implementations may support different extensions based on what service the client application creates a session with, so using this extension at logon time is recommended to verify extension support.

If any extensions are supported by a 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 extension the application is querying the service about. These can be either extension sets or individual extensions. An item code of null will be ignored. The flags for the structures that are used on input are:

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 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.

CMC_X_COM_CONFIG_DATA

Description:
Get all values available with cmc_query_configuration() in a structure.

Used by:
cmc_query_config()

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 form 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 definitions 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().

CMC_X_COM_CAN_SEND_RECIP

Description:
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

CMC_X_COM_SAVE_MESSAGE

Description:
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().

CMC_X_COM_SENT_MESSAGE

Description:
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().

DATA EXTENSIONS

CMC_X_COM_TIME_RECEIVED

Description:
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.

CMC_X_COM_RECIP_ID

Description:

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

CMC_X_COM_ATTACH_CHARPOS

Description:

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 that 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

CMC_X_COM_PRIORITY

Description:

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

C Declaration Summary

This section 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 when 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 receive 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)
```

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 the X.400 messaging protocol based on existing XAPIA interfaces and also for use with G3 facsimile, G3-64 facsimile, G4 facsimile, telex and Teletex service via the ITU-TS (CCITT) T.611 Recommendation. To find out what extension sets are available, contact the XAPIA.

Appendix C Platform Specific Information including Runtime 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 section provides several general requirements and platform-specific requirements for several common platforms and operating systems.

Unless specified otherwise below, the following definitions apply to all platforms:

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

Explicit and Implicit Binding

All functions in the CMC API should be link-able 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.

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).

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.

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] section 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.

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.

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() look for the CMC variable in the [MAIL] section of OS2.INI. The variable will indicate whether the DLL is 16-bit or 32-bit. If this variable is present and non-zero, it indicates that a 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.

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 WinQueryProfileInt() look for the CMC variable in the [MAIL] section of OS2.INI. The variable will indicate whether the DLL is 16-bit or 32-bit. If this variable is present and non-zero, it indicates that a 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.

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.

MPE/ix Binding

Dynamic linking is not available. The program must be linked using the file CMC.RL.THREEK, and currently, PH,PM and MR capabilities are required.

Contact Information

3k Associates, Inc.
6901 Old Keene Mill Rd, Suite 500
Springfield, VA 22150
Phone: (703) 569-9189
Fax: (703) 451-3720
E-Mail: Sales@3k.com • Support@3k.com

Office Hours are 9am to 8pm Eastern (U.S.) Time

Source Code Examples

Source code examples are provided with each installation in the SOURCE group of the THREEK account. Source files start with "CMC". All source code examples are provided by 3k Associates, Inc and are not part of the CMC standard specs. For example:

```
ACCOUNT=  THREEK      GROUP=  SOURCE

FILENAME  CODE  ------------LOGICAL RECORD-------
                  SIZE  TYP         EOF      LIMIT

CMCCEX             80B  FA          511        511
CMCCOBEX  EDTCT    80B  FA          719        719
CMCSPLEX           80B  FA           66         68
```

CMCCEX is sample code in "C". CMCCOBEX is in COBOL, and CMCSPLEX is in SPLash!

The sample program in COBOL (CMCCOBEX) that uses various CMC calls:

```
001000$CONTROL USLINIT
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID.     COBCMC.
001300 AUTHOR.         TOM KIRBY, 3k Associates, Inc.
001400 ENVIRONMENT DIVISION.
001500 CONFIGURATION SECTION.
001600 SOURCE-COMPUTER. HP-3000.
001700 OBJECT-COMPUTER. HP-3000.
001800 DATA DIVISION.
001900 WORKING-STORAGE SECTION.
002000
002100* FIELDS USED BY CMC CALLS
002200
002300* CMC return code
002400 01  RC                        PIC 9(9) COMP VALUE 0.
002500
002600* CMC session id
002700 01  SID                       PIC 9(9) COMP VALUE 0.
002800 01  TEMP-ID                       PIC 9(9) COMP VALUE 0.
002900
003000* CMC message structure
003100 01  MSG.
003200     02  MSG-ENTRY OCCURS 2 TIMES.
003300         03  MESSAGE-REFERENCE       PIC S9(9) BINARY VALUE 0.
003400         03  MESSAGE-TYPE            PIC S9(9) BINARY VALUE 0.
003500         03  SUBJECT                 PIC S9(9) BINARY VALUE 0.
003600         03  TIME-SENT.
003700             04  FILLER OCCURS 3 TIMES PIC 9(9) COMP.
003800         03  TEXT-NOTE               PIC S9(9) BINARY VALUE 0.
003900         03  RECIPIENTS              PIC S9(9) BINARY VALUE 0.
004000         03  ATTACHMENTS             PIC S9(9) BINARY VALUE 0.
004100         03  MESSAGE-FLAGS           PIC S9(9) BINARY VALUE 0.
004200         03  MESSAGE-EXTENSIONS      PIC S9(9) BINARY VALUE 0.
004300
```

```
004400* CMC flags parameter
004500 01  FLGS                    PIC 9(9) COMP VALUE 0.
004600
004700* CMC user interface ID (not used by us)
004800 01  UI-ID                   PIC 9(9) COMP VALUE 0.
004900
005000* NETMAIL user
005100 01  USER                    PIC X(17) VALUE SPACES.
005200
005300* NETMAIL user password
005400 01  PASS                    PIC X(9) VALUE SPACES.
005500
005600* CMC version (must match RL)
005700 01  VERS                    PIC 9(4) COMP VALUE 0.
005800
005900* CMC attachment structure
006000 01  ATTCH.
006100     02  A-ENTRY OCCURS 10 TIMES.
006200         03  ATTACH-TITLE        PIC S9(9) BINARY VALUE 0.
006300         03  ATTACH-TYPE         PIC 9(9) COMP VALUE 0.
006400         03  ATTACH-FILENAME     PIC S9(9) BINARY VALUE 0.
006500         03  ATTACH-FLAGS        PIC 9(9) COMP VALUE 0.
006600         03  ATTACH-EXTENSIONS   PIC S9(9) BINARY VALUE 0.
006700
006800* CMC recipient structure
006900 01  RECPT.
007000     02  REC OCCURS 3 TIMES.
007100         03  RNAME               PIC S9(9) BINARY.
007200         03  NAME-TYPE           PIC S9(9) COMP.
007300         03  RADDRESS            PIC S9(9) BINARY.
007400         03  ROLE                PIC S9(9) COMP.
007500         03  RECIP-FLAGS         PIC 9(9) COMP.
007600         03  RECIP-EXTENSIONS    PIC S9(9) BINARY.
007700
007800* Another recipient structure for the second message in the
007900* message structure...
008000 01  RECPT-2.
008100     02  R2NAME                  PIC S9(9) BINARY.
008200     02  R2NAME-TYPE             PIC S9(9) COMP.
008300     02  R2ADDRESS               PIC S9(9) BINARY.
008400     02  R2ROLE                  PIC S9(9) COMP.
008500     02  R2RECIP-FLAGS           PIC 9(9) COMP.
008600     02  R2RECIP-EXTENSIONS      PIC S9(9) BINARY.
008700
008800* CMC extensions structure (not used)
008900 01  EXTENSIONS                  PIC S9(9) BINARY VALUE 0.
009000
009100* CMC object identifier (character set), also not used
009200 01  CHARSET                     PIC S9(9) BINARY VALUE 0.
009300
009400* CMC service (not used)
009500 01  SERVICE                     PIC S9(9) BINARY VALUE 0.
009600
009700* VARIABLES TO TAKE THE PLACE OF CMC DEFINES
```

```
009800
009900 01  CMC-MSG-TEXT-NOTE-AS-FILE       PIC 9(9) COMP VALUE 2.
010000 01  CMC-RECIP-LAST-ELEMENT          PIC 9(9) COMP.
010100 01  CMC-MSG-LAST-ELEMENT            PIC 9(9) COMP.
010200 01  CMC-ATT-LAST-ELEMENT            PIC 9(9) COMP.
010300 01  CMC-TYPE-INDIVIDUAL             PIC 9(9) COMP VALUE 1.
010400 01  CMC-ROLE-TO                     PIC 9(9) COMP VALUE 0.
010500 01  CMC-ROLE-CC                     PIC 9(9) COMP VALUE 1.
010600 01  CMC-ROLE-BCC                    PIC 9(9) COMP VALUE 2.
010700 01  CMC-ATT-OID-BINARY.
010800     02  FILLER         PIC X(18) VALUE "1 2 840 113658 1 1".
010900     02  FILLER         PIC X     VALUE %0.
011000 01  CMC-ATT-OID-TEXT.
011100     02  FILLER         PIC X(20) VALUE "1 2 840 113658 1 1 0".
011200     02  FILLER         PIC X     VALUE %0.
011300 01  CMC-CONFIG-CHARACTER-SET        PIC 9(9) COMP VALUE 1.
011400 01  CMC-CONFIG-LINE-TERM            PIC 9(9) COMP VALUE 2.
011500 01  CMC-CONFIG-DEFAULT-SERVICE      PIC 9(9) COMP VALUE 3.
011600 01  CMC-CONFIG-DEFAULT-USER         PIC 9(9) COMP VALUE 4.
011700 01  CMC-CONFIG-REQ-PASSWORD         PIC 9(9) COMP VALUE 5.
011800 01  CMC-CONFIG-REQ-SERVICE          PIC 9(9) COMP VALUE 6.
011900 01  CMC-CONFIG-REQ-USER             PIC 9(9) COMP VALUE 7.
012000 01  CMC-CONFIG-UI-AVAIL             PIC 9(9) COMP VALUE 8.
012100 01  CMC-CONFIG-SUP-NOMKMSGREAD      PIC 9(9) COMP VALUE 9.
012200 01  CMC-CONFIG-SUP-COUNTED-STR      PIC 9(9) COMP VALUE 10.
012300 01  CMC-CONFIG-VER-IMPLEM           PIC 9(9) COMP VALUE 11.
012400 01  CMC-CONFIG-VER-SPEC             PIC 9(9) COMP VALUE 12.
012500 01  CMC-LINE-TERM-CRLF              PIC 9(9) COMP VALUE 0.
012600 01  CMC-LINE-TERM-CR                PIC 9(9) COMP VALUE 1.
012700 01  CMC-LINE-TERM-LF                PIC 9(9) COMP VALUE 2.
012800 01  CMC-REQUIRED-NO                 PIC 9(9) COMP VALUE 0.
012900 01  CMC-REQUIRED-YES                PIC 9(9) COMP VALUE 1.
013000 01  CMC-REQUIRED-OPT                PIC 9(9) COMP VALUE 2.
013010 01  CMC-FALSE                       PIC 9(4) COMP VALUE 0.
013020 01  CMC-TRUE                        PIC 9(4) COMP VALUE 1.
013100
013200* WORK AREAS
013300
013400 01  II                          PIC S9(4) COMP VALUE 0.
013500 01  NULL-TERMINATOR                 PIC X VALUE LOW-VALUE.
013600 01  CARR-RET                        PIC X VALUE %15.
013700 01  NEWLINE                         PIC X VALUE %12.
013800 01  PTR1                        PIC S9(9) BINARY VALUE 0.
013900 01  PTR2                        PIC S9(9) BINARY VALUE 0.
014000 01  M-SUB                         PIC X(80) VALUE SPACES.
014100 01  A-FILE                        PIC X(80) VALUE SPACES.
014200 01  B-FILE                        PIC X(80) VALUE SPACES.
014300 01  NAME-WORK                       PIC X(30) VALUE SPACES.
014400 01  NAME-AND-ADDRESS.
014500     02  U-NAME OCCURS 3 TIMES       PIC X(30).
014600     02  U-ADDR OCCURS 3 TIMES       PIC X(80).
014700 01  U2-NAME                         PIC X(30).
014800 01  U2-ADDR                         PIC X(80).
014900 01  N-TEXT                        PIC X(256) VALUE SPACES.
```

```
015000 01  BIG-NUM                    PIC 9(16) COMP VALUE 0.
015100 01  LIL-NUMS REDEFINES BIG-NUM.
015200     02  FILLER                 PIC 9(9) COMP.
015300     02  LIL-NUM                PIC 9(9) COMP.
015400
015500* FIELDS FOR CMC_SEND_DOCUMENTS
015600
015700* These appear in the CALL statement, and hold addresses...
015800 01  SD-ADDRESSES               PIC S9(9) BINARY VALUE 0.
015900 01  SD-TEXT                    PIC S9(9) BINARY VALUE 0.
016000 01  SD-SUBJECT                 PIC S9(9) BINARY VALUE 0.
016100 01  SD-FILES                   PIC S9(9) BINARY VALUE 0.
016200 01  SD-FLAGS                   PIC 9(9) COMP    VALUE 0.
016300 01  SD-TITLES                  PIC S9(9) BINARY VALUE 0.
016400 01  SD-DELIMITER               PIC S9(9) BINARY VALUE 0.
016500 01  SD-UI-ID                   PIC 9(9) COMP    VALUE 0.
016600
016700* These are pointed to by the fields in the CALL statement...
016800 01  SDS-ADDRESSES              PIC X(256) VALUE SPACES.
016900 01  SDS-TEXT                   PIC X(256) VALUE SPACES.
017000 01  SDS-SUBJECT                PIC X(80)  VALUE SPACES.
017100 01  SDS-TITLES                 PIC X(256) VALUE SPACES.
017200 01  SDS-FILES                  PIC X(256) VALUE SPACES.
017300 01  SDS-DELIMITER              PIC X      VALUE ",".
017400
017500* FIELDS FOR CMC_QUERY_CONFIGURATION
017600
017800 01  CMC-ITEM                   PIC S9(9) COMP VALUE 0.
017900 01  CMC-ENUMVAL                PIC S9(9) COMP VALUE 0.
018000 01  CMC-CELL                   PIC S9(9) BINARY VALUE 0.
018100 01  CMC-CELLPTR                PIC S9(9) BINARY VALUE 0.
018200 01  CMC-BOOLEAN                PIC 9(4) COMP VALUE 0.
018300 01  CMC-UINT                   PIC 9(4) COMP VALUE 0.
018400
018500 01  CELL-CONTENTS              PIC X(60) VALUE SPACES.
018600 01  CELL-ADDR                  PIC S9(9) BINARY VALUE 0.
018700
018800 PROCEDURE DIVISION.
018900 XXXX-MAIN.
019000     MOVE LOW-VALUES TO RECPT, RECPT-2, MSG.
019100     MOVE 2147483648 TO BIG-NUM.
019200     MOVE LIL-NUM TO  CMC-RECIP-LAST-ELEMENT
019300                      CMC-ATT-LAST-ELEMENT
019400                      CMC-MSG-LAST-ELEMENT.
019500
019600* 0000, 1000 and 3000 are used for mailing via CMC_SEND, while 2000
019700* is used for the CMC_SEND_DOCUMENTS method (simpler to use, but
019800* higher in overhead costs)...
019900
020000     PERFORM 4000-PRINT-CONFIG THRU 4999-EXIT.
020100
020200*    PERFORM 0000-CMCLOGON THRU 0999-EXIT.
020300*    PERFORM 1000-CMCSEND-FILE THRU 1999-EXIT.
020400*    PERFORM 2000-CMCSENDDOC THRU 2999-EXIT.
```

```
020500*    PERFORM 3000-CMCLOGOFF THRU 3999-EXIT.
020600
020700     STOP RUN.
020800
020900 0000-CMCLOGON.
021000* CMCLOGON
021100*************************************************************
021200* This will not be needed with CMC_send_document, but it is *
021300* needed with CMC_send.                                     *
021400*************************************************************
021500
021600* UI-ID is not used anyway, set to 0 *
021700     MOVE 0 to UI-ID.
021800
021900* USER must be 16 or less characters, terminated with a \0 *
022000     STRING "MY_USER" NULL-TERMINATOR DELIMITED BY SIZE
022100       INTO USER.
022200* NOTE: If you wish to pass a NULL, you must move 0 to PTR1,
022300*       because a string of spaces doth not a true NULL make.
022400*       Otherwise, you can skip the PTR1 step and just pass
022500*       the variable USER to the CMC routine.
022600*
022700*       If you use PTR1, be sure to pass it as \PTR1\, because
022800*       its value of 0 is what you really want to pass...
022900*    MOVE 0 TO PTR1.
023000
023100* PASS must be 8 characters, terminated with a \0 *
023200     STRING "pAsSwOrD" NULL-TERMINATOR DELIMITED BY SIZE
023300       INTO PASS.
023400* NOTE: If you wish to pass a NULL, you must move 0 to PTR2,
023500*       because a string of spaces doth not a true NULL make.
023600*       Otherwise, you can skip the PTR2 step and just pass
023700*       the variable PASS to the CMC routine.
023800*
023900*       If you use PTR2, be sure to pass it as \PTR2\, because
024000*       its value of 0 is what you really want to pass...
024100*    MOVE 0 TO PTR2.
024200
024300* SERVICE, CHARACTER SET, AND EXTENSIONS ARE NOT USED, PASS NULLS. *
024400
024500* Programmed to specification 1.00 *
024600     MOVE 100 TO VERS.
024700
024800* Pass a 0 in FLGS because we don't support LOGON_UI, ERROR_UI, *
024900* or COUNTED_STRING_TYPE.                                       *
025000     MOVE 0 TO FLGS.
025100
025200* SID is the session ID that must be used in all the other calls. *
025300     CALL "CMC_LOGON" USING \SERVICE\, USER, PASS,
025400                            \CHARSET\, \UI-ID\, \VERS\,
025500                            \FLGS\, SID, \EXTENSIONS\
025600                       GIVING RC.
025700
025800* I am just printing the return, you will probably want to stop *
```

```
025900* if you don't get a 0 return...                          *
026000     DISPLAY "LOGON = " RC.
026100
026200 0999-EXIT.
026300     EXIT.
026400
026500 1000-CMCSEND-FILE.
026600* CMCSEND
026700*************************************************************************
026800* There are two ways of sending a text file. First, there is the *
026900* sending as a file method. Here is how to do this method:       *
027000*************************************************************************
027100
027200* FLGS is 0 because we don't support COUNTED_STRING_TYPE or any *
027300* of the UIs...                                          *
027400     MOVE 0 TO FLGS.
027500
027600* To indicate that the message text is a file, we set the *
027700* CMC_MSG_TEXT_NOTE_AS_FILE flag in the CMC_message       *
027800* structure.                                             *
027900     MOVE 0 TO BIG-NUM.
028000
028100* NOTE: If you wish to set more than one of the flags, just add
028200*       them together. If you need to set the CMC_MSG_LAST_ELEMENT
028300*       flag, MOVE it to something like BIG-NUM first, add the
028400*       others, and move the second half of BIG-NUM into your
028500*       FLGS. I do this because COBOL may come up with funny
028600*       results moving what is actually a 10-digit number into
028700*       a field that specifies 9 (but will actually hold some of
028800*       the low 10's)...
028900
029000* Set the CMC_MSG_TEXT_NOTE_AS_FILE flag *
029100     ADD CMC-MSG-TEXT-NOTE-AS-FILE TO BIG-NUM.
029200     MOVE LIL-NUM TO MESSAGE-FLAGS (1).
029300
029400* MESSAGE-REFERENCE, MESSAGE-TYPE are ignored. *
029500
029600* TEXT-NOTE is NULL in this case because the text is in a file. *
029700     MOVE 0 TO TEXT-NOTE (1).
029800
029900* SUBJECT is a string terminated by \0. *
030000     STRING "Re: Geoff's behavior..." NULL-TERMINATOR DELIMITED
030100       BY SIZE INTO M-SUB.
030200     CALL INTRINSIC ".LOC." USING M-SUB GIVING SUBJECT (1).
030300
030400* NOTE: The CMC routines were meant to be called from C, or some
030500*       other pointer-intensive language. We must simulate this in
030600*       COBOL, as in the above example...
030700
030800* Load attachments with attachment structures. *
030900     CALL INTRINSIC ".LOC." USING ATTCH GIVING ATTACHMENTS (1).
031000
031100* attach_title isn't used yet... *
031200* Actually, you can go ahead and put one in, but currently, the
```

```
031300* NETMAIL engine won't use it...
031400* If you do use TITLE, be sure to put POINTERS in these fields...
031500     MOVE 0 TO ATTACH-TITLE (1).
031600     MOVE 0 TO ATTACH-TITLE (2).
031700
031800* attach_type isn't used yet... *
031900* You can put one in, either CMC-ATT-OID-BINARY or -TEXT, but
032000* it will have no affect...
032100* If you do use TYPE, once again, these are POINTERS...
032200     MOVE 0 TO ATTACH-TYPE (1).
032300     MOVE 0 TO ATTACH-TYPE (2).
032400
032500* Set the ATT_LAST_ELEMENT flag in the second entry... *
032600     MOVE 0 TO ATTACH-FLAGS (1).
032700     MOVE CMC-ATT-LAST-ELEMENT TO ATTACH-FLAGS (2).
032800
032900* attach_filename is a string terminated by a \0... *
033000     STRING "FILE.GRP.ACCT01" NULL-TERMINATOR DELIMITED BY SIZE
033100       INTO A-FILE.
033200     CALL INTRINSIC ".LOC." USING A-FILE
033300                         GIVING ATTACH-FILENAME (1).
033400     STRING "FILE.GRP.ACCT02" NULL-TERMINATOR
033500       DELIMITED BY SIZE INTO B-FILE.
033600     CALL INTRINSIC ".LOC." USING B-FILE
033700                         GIVING ATTACH-FILENAME (2).
033800
033900* Load recipients with the address of our recipient array... *
034000     CALL INTRINSIC ".LOC." USING RECPT GIVING RECIPIENTS (1).
034100
034200* name and address must be strings terminated with a \0... *
034300     STRING "JOHN Q. PUBLIC" NULL-TERMINATOR DELIMITED BY SIZE
034400       INTO U-NAME (1).
034500     STRING "ME@HERE.COM" NULL-TERMINATOR DELIMITED BY SIZE
034600       INTO U-ADDR (1).
034700
034800     CALL INTRINSIC ".LOC." USING U-NAME (1) GIVING RNAME (1).
034900     MOVE CMC-TYPE-INDIVIDUAL TO NAME-TYPE (1).
035000     CALL INTRINSIC ".LOC." USING U-ADDR (1) GIVING RADDRESS (1).
035100
035200* This one (entry 0) goes on the "TO:" list... *
035300     MOVE CMC-ROLE-TO TO ROLE (1).
035400
035500* The recip_flags available are CMC_RECIP_IGNORE,       *
035600* CMC_RECIP_LIST_TRUNCATED, and CMC_RECIP_LAST_ELEMENT. *
035700* We don't need any of these for this recipient...      *
035800     MOVE 0 TO RECIP-FLAGS (1).
035900
036000* Same stuff for entry 1... *
036100     STRING "JOHN DOE" NULL-TERMINATOR DELIMITED BY SIZE
036200       INTO U-NAME (2).
036300     STRING "YOU@THERE.COM" NULL-TERMINATOR DELIMITED BY SIZE
036400       INTO U-ADDR (2).
036500     CALL INTRINSIC ".LOC." USING U-NAME (2) GIVING RNAME (2).
036600     CALL INTRINSIC ".LOC." USING U-ADDR (2) GIVING RADDRESS (2).
```

```
036700     MOVE CMC-TYPE-INDIVIDUAL TO NAME-TYPE (2).
036800
036900* This one (entry 1) goes on the "CC:" list... *
037000     MOVE CMC-ROLE-CC TO ROLE (2).
037100     MOVE 0 TO RECIP-FLAGS (2).
037200
037300* Entry 2, same story (almost)... *
037400     STRING "CAPTAIN KLUTZ" NULL-TERMINATOR DELIMITED BY SIZE
037500       INTO U-NAME (3).
037600     STRING "BOSS@HERE.COM" NULL-TERMINATOR DELIMITED BY SIZE
037700       INTO U-ADDR (3).
037800     CALL INTRINSIC ".LOC." USING U-NAME (3) GIVING RNAME (3).
037900     CALL INTRINSIC ".LOC." USING U-ADDR (3) GIVING RADDRESS (3).
038000     MOVE CMC-TYPE-INDIVIDUAL TO NAME-TYPE (3).
038100
038200* Entry 2 is goes on the "BCC:" list... *
038300     MOVE CMC-ROLE-BCC TO ROLE (3).
038400* Last recipient, set the flag... *
038500     MOVE CMC-RECIP-LAST-ELEMENT TO RECIP-FLAGS (3).
038600
038700* 2nd message...
038800
038900* NO ATTACHMENTS used in this message...
039000     MOVE 0 TO ATTACHMENTS (2).
039100
039200* Notice that this time around, CMC_MSG_TEXT_NOTE_AS_FILE is *
039300* *NOT* set, but I am setting CMC-MSG-LAST-ELEMENT...        *
039400     MOVE CMC-MSG-LAST-ELEMENT TO MESSAGE-FLAGS (2).
039500     CALL INTRINSIC ".LOC." USING M-SUB GIVING SUBJECT (2).
039600
039700* Load the message into text_note, terminated by a \0... *
039800     STRING "NOTHING IN PARTICULAR" CARR-RET NEWLINE
039900       NULL-TERMINATOR DELIMITED BY SIZE INTO N-TEXT.
040000     CALL INTRINSIC ".LOC." USING N-TEXT GIVING TEXT-NOTE (2).
040100
040200* Load recipients with the address of our recipient array... *
040300     CALL INTRINSIC ".LOC." USING RECPT-2 GIVING RECIPIENTS (2).
040400
040500     STRING "CAPTAIN KLUTZ" NULL-TERMINATOR DELIMITED BY SIZE
040600       INTO U2-NAME.
040700     STRING "YOU@THERE.COM" NULL-TERMINATOR DELIMITED BY SIZE
040800       INTO U2-ADDR.
040900     CALL INTRINSIC ".LOC." USING U2-NAME GIVING R2NAME.
041000     CALL INTRINSIC ".LOC." USING U2-ADDR GIVING R2ADDRESS.
041100     MOVE CMC-TYPE-INDIVIDUAL TO R2NAME-TYPE.
041200
041300     MOVE CMC-ROLE-TO TO R2ROLE.
041400     MOVE CMC-RECIP-LAST-ELEMENT TO R2RECIP-FLAGS.
041500
041600     CALL "CMC_SEND" USING \SID\, MSG, \FLGS\, \UI-ID\,
041700                           \EXTENSIONS\
041800                   GIVING RC.
041900
042000* I am just printing the return, you will probably want to stop *
```

```
042100* if you don't get a 0 return...                    *
042200     DISPLAY "SEND(1) = " RC.
042300
042400 1999-EXIT.
042500     EXIT.
042600
042700 2000-CMCSENDDOC.
042800
042900* SET UP ADDRESSES
043000     STRING "YOU@THERE.COM" SDS-DELIMITER "ME@HERE.COM"
043100            SDS-DELIMITER
043200            "BCC:BOSS@HERE.COM" NULL-TERMINATOR DELIMITED BY SIZE
043300            INTO SDS-ADDRESSES.
043400
043500* SET UP TEXT
043600     STRING "Thou art lucky. I have sent thee a document."
043700            NULL-TERMINATOR DELIMITED BY SIZE
043800            INTO SDS-TEXT.
043900* If you have no text, use the next line in place of the above,
044000* and replace SDS-TEXT with \SD-TEXT\ on the CALL statement...
044100*    MOVE 0 TO SD-TEXT.
044200
044300* SET UP SUBJECT
044400     STRING "ALMOST FREE UNLIMITED TIME OFFER!" NULL-TERMINATOR
044500            DELIMITED BY SIZE INTO SDS-SUBJECT.
044600* If you have no subject, use the next line in place of the above,
044700* and replace SDS-SUBJECT with \SD-SUBJECT\ on the CALL statement...
044800*    MOVE 0 TO SD-SUBJECT.
044900
045000* SET UP TITLES
045100     STRING "Some Text" SDS-DELIMITER "A Program" NULL-TERMINATOR
045200            DELIMITED BY SIZE INTO SDS-TITLES.
045300* If you have no titles, use the next line in place of the above,
045400* and replace SDS-TITLES with \SD-TITLES\ on the CALL statement...
045500*    MOVE 0 TO SD-TITLES.
045600
045700* SET UP FILES
045800     STRING "FILE.GRP.ACCT01" SDS-DELIMITER "FILE.GRP.ACCT02"
045900            NULL-TERMINATOR DELIMITED BY SIZE INTO SDS-FILES.
046000* If you have no files, use the next line in place of the above,
046100* and replace SDS-FILES with \SD-FILES\ on the CALL statement...
046200*    MOVE 0 TO SD-FILES.
046300
046400* NOTE: SDS-DELIMITER and SDS-ADDRESS are always required, so they
046500*       can be passed straight. SDS-TEXT, SDS-FILES, and SDS-TITLES
046600*       may be null, and since this is NOT an OPTION VARIABLE
046700*       procedure, the address of 0 must be passed. It is put in a
046800*       variable to make sure there is not confusion: a 32-bit 0
046900*       address is passed. SD-UI-ID and SD-FLAGS are passed by
047000*       value. If you plan to use SDS-TEXT, SDS-FILES and/or
047100*       SDS-TITLES, you may pass them directly.
047200
047300* LEAVE FLAGS AT 0 THIS TIME.
047400
```

```
047500* CALL...
047600     CALL "CMC_SEND_DOCUMENTS" USING SDS-ADDRESSES, SDS-SUBJECT,
047700                              SDS-TEXT, \SD-FLAGS\, SDS-FILES,
047800                              SDS-TITLES, SDS-DELIMITER,
047900                              \SD-UI-ID\
048000                             GIVING RC.
048100
048200     DISPLAY "SEND_DOC = " RC.
048300
048400 2999-EXIT.
048500     EXIT.
048600
048700 3000-CMCLOGOFF.
048800* CMCLOGOFF *
048900* Set FLGS to 0 because we don't support any UIs... *
049000     MOVE 0 TO FLGS.
049100
049200     CALL "CMC_LOGOFF" USING \SID\, \UI-ID\, \FLGS\,
049300                             \EXTENSIONS\
049400                        GIVING RC.
049500
049600* I am just printing the return, you will probably want to stop *
049700* if you don't get a 0 return...                               *
049800     DISPLAY "LOGOFF = " RC.
049900
050000 3999-EXIT.
050100     EXIT.
050200
050300 4000-PRINT-CONFIG.
050400
050500* NOTE: Some items returned by cmc_query_configuration are malloc'ed
050600*       pointers, which COBOL cannot read directly. Therefore, a new
050700*       routine, cmc_cobol_cell_read, is used to move data from these
050800*       pointers into static WORKING-STORAGE areas.
050900
051000* We will use TEMP-ID and its current value, because we aren't
051100* actually logging on...
051200
051300* First, what character sets are available?
051400
051500     MOVE CMC-CONFIG-CHARACTER-SET TO CMC-ITEM.
051800     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
051900                                    CMC-CELLPTR,
052000                                    \EXTENSIONS\
052100                                GIVING RC.
052200     IF RC = 0
052300        CALL "CMC_COBOL_CELL_READ" USING \CMC-CELLPTR\, CELL-ADDR,
052400                                    4, 1
052500        PERFORM VARYING II FROM 1 BY 1 UNTIL CELL-ADDR = 0
052600           CALL "CMC_COBOL_CELL_READ" USING \CELL-ADDR\,
052700                                       CELL-CONTENTS,
052800                                       60, 0
052900           DISPLAY "Char set: " CELL-CONTENTS
053000           CALL "CMC_FREE" USING \CELL-ADDR\
```

```
053100         COMPUTE CMC-CELL = CMC-CELLPTR + (II * 4)
053200         CALL "CMC_COBOL_CELL_READ" USING \CMC-CELL\,
053300                                  CELL-ADDR, 4, 1
053400       END-PERFORM
053500       CALL "CMC_FREE" USING \CMC-CELLPTR\
053600     ELSE
053700       DISPLAY "Char set: ???".
053800
053900* Next, what do we use as a line terminator?
054000
054100     MOVE CMC-CONFIG-LINE-TERM TO CMC-ITEM.
054400     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
054500                                  CMC-ENUMVAL,
054600                                  \EXTENSIONS\
054700                             GIVING RC.
054800     DISPLAY "Line terminator: " WITH NO ADVANCING.
054900     IF RC = 0
055000       EVALUATE CMC-ENUMVAL
055100         WHEN CMC-LINE-TERM-CRLF
055200           DISPLAY "CR/LF"
055300         WHEN CMC-LINE-TERM-LF
055400           DISPLAY "LF"
055500         WHEN CMC-LINE-TERM-CR
055600           DISPLAY "CR"
055700         WHEN OTHER
055800           DISPLAY "BOGUS RETURN!"
055900       END-EVALUATE
056000     ELSE
056100       DISPLAY "???".
056200
056300* Next, what default service?
056400
056500     MOVE CMC-CONFIG-DEFAULT-SERVICE TO CMC-ITEM.
056800     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
056900                                  CMC-CELL,
057000                                  \EXTENSIONS\
057100                             GIVING RC.
057200     IF RC = 0
057300       CALL "CMC_COBOL_CELL_READ" USING \CMC-CELL\,
057400                                  CELL-CONTENTS,
057500                                  60, 0
057600       DISPLAY "Default service: " CELL-CONTENTS
057700       CALL "CMC_FREE" USING \CMC-CELL\
057800     ELSE
057900       DISPLAY "Default service: ???".
058000
058100* Next, what default user?
058200
058300     MOVE CMC-CONFIG-DEFAULT-USER TO CMC-ITEM.
058600     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
058700                                  CMC-CELL,
058800                                  \EXTENSIONS\
058900                             GIVING RC.
059000     IF RC = 0
```

```
059100       CALL "CMC_COBOL_CELL_READ" USING \CMC-CELL\,
059200                                    CELL-CONTENTS,
059300                                    60, 0
059400       DISPLAY "Default user: " CELL-CONTENTS
059500       CALL "CMC_FREE" USING \CMC-CELL\
059600     ELSE
059700       DISPLAY "Default service: ???".
059800
059810* Next, is a password required?
059820
059830     MOVE CMC-CONFIG-REQ-PASSWORD TO CMC-ITEM.
059860     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
059870                                    CMC-ENUMVAL,
059880                                    \EXTENSIONS\
059890                                GIVING RC.
059891     DISPLAY "Password: " WITH NO ADVANCING.
059892     IF RC = 0
059893       EVALUATE CMC-ENUMVAL
059894         WHEN CMC-REQUIRED-NO
059895           DISPLAY "Not required"
059896         WHEN CMC-REQUIRED-YES
059897           DISPLAY "Required"
059898         WHEN CMC-REQUIRED-OPT
059899           DISPLAY "Optional"
059950         WHEN OTHER
059960           DISPLAY "BOGUS RETURN!"
059970       END-EVALUATE
059980     ELSE
059990       DISPLAY "???".
062200
062300* Next, is a service name required?
062310
062410     MOVE CMC-CONFIG-REQ-SERVICE TO CMC-ITEM.
062440     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
062450                                    CMC-ENUMVAL,
062460                                    \EXTENSIONS\
062470                                GIVING RC.
062480     DISPLAY "Service name: " WITH NO ADVANCING.
062490     IF RC = 0
062491       EVALUATE CMC-ENUMVAL
062492         WHEN CMC-REQUIRED-NO
062493           DISPLAY "Not required"
062494         WHEN CMC-REQUIRED-YES
062495           DISPLAY "Required"
062496         WHEN CMC-REQUIRED-OPT
062497           DISPLAY "Optional"
062498         WHEN OTHER
062499           DISPLAY "BOGUS RETURN!"
062570       END-EVALUATE
062580     ELSE
062590       DISPLAY "???".
064600
064710* Next, is a user name required?
064720
```

```
064730     MOVE CMC-CONFIG-REQ-USER TO CMC-ITEM.
064760     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
064770                                  CMC-ENUMVAL,
064780                                  \EXTENSIONS\
064790                             GIVING RC.
064791     DISPLAY "User: " WITH NO ADVANCING.
064792     IF RC = 0
064793        EVALUATE CMC-ENUMVAL
064794           WHEN CMC-REQUIRED-NO
064795              DISPLAY "Not required"
064796           WHEN CMC-REQUIRED-YES
064797              DISPLAY "Required"
064798           WHEN CMC-REQUIRED-OPT
064799              DISPLAY "Optional"
064840           WHEN OTHER
064850              DISPLAY "BOGUS RETURN!"
064860        END-EVALUATE
064870     ELSE
064880        DISPLAY "???".
064890
064900* Next, do we support any user interfaces?
065000
065100     MOVE CMC-CONFIG-UI-AVAIL TO CMC-ITEM.
065200     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
065300                                  CMC-BOOLEAN,
065400                                  \EXTENSIONS\
065500                             GIVING RC.
065600     IF RC = 0
065700        IF CMC-BOOLEAN = CMC-TRUE
065800           DISPLAY "UI: Available"
065900        ELSE
066000           DISPLAY "UI: Not Available"
066100     ELSE
066200        DISPLAY "UI: ???".
068100
068210* Next, is DO_NOT_MARK_AS_READ supported?
068220
068230     MOVE CMC-CONFIG-SUP-NOMKMSGREAD TO CMC-ITEM.
068240     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
068250                                  CMC-BOOLEAN,
068260                                  \EXTENSIONS\
068270                             GIVING RC.
068280     IF RC = 0
068290        IF CMC-BOOLEAN = CMC-TRUE
068291           DISPLAY "DO_NOT_MARK_AS_READ supported: Yes"
068292        ELSE
068293           DISPLAY "DO_NOT_MARK_AS_READ supported: No"
068294     ELSE
068295        DISPLAY "DO_NOT_MARK_AS_READ supported: ???".
069200
069310* Next, is DO_NOT_MARK_AS_READ supported?
069320
069330     MOVE CMC-CONFIG-SUP-COUNTED-STR TO CMC-ITEM.
069340     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
```

```
069350                                  CMC-BOOLEAN,
069360                                  \EXTENSIONS\
069370                              GIVING RC.
069380     IF RC = 0
069390        IF CMC-BOOLEAN = CMC-TRUE
069391           DISPLAY "Counted string: Supported"
069392        ELSE
069393           DISPLAY "Counted string: Not supported"
069394     ELSE
069395        DISPLAY "Counted string: ???".
070300
070310* Next, which version is this implementation?
070320
070330     MOVE CMC-CONFIG-VER-IMPLEM TO CMC-ITEM.
070340     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
070350                                  CMC-UINT,
070360                                  \EXTENSIONS\
070370                              GIVING RC.
070380     IF RC = 0
070390        DISPLAY "Version (implementation): " CMC-UINT
070430     ELSE
070431        DISPLAY "Version (implementation): ???".
071100
071110* Finally, which version is our specification?
071111
071120     MOVE CMC-CONFIG-VER-SPEC TO CMC-ITEM.
071130     CALL "CMC_QUERY_CONFIGURATION" USING \TEMP-ID\, \CMC-ITEM\,
071140                                  CMC-UINT,
071150                                  \EXTENSIONS\
071160                              GIVING RC.
071170     IF RC = 0
071180        DISPLAY "Version (specs): " CMC-UINT
071190     ELSE
071200        DISPLAY "Version (specs): ???".
071300
071900 4999-EXIT.
072000     EXIT.
```

The sample program in C (CMCCEX) that uses various CMC calls:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "xcmc.h"

/**************/
/* PROTOTYPES */
/**************/
void cmclogon(void);
void cmcsend(void);
void cmclogoff(void);
void cmcsenddocs(void);
void cmcprintconfig(void);

/****************************/
/* FIELDS USED BY CMC CALLS */
/****************************/

CMC_return_code rc;
CMC_session_id sessid;
CMC_message msg[2];
CMC_flags flgs, flags;
CMC_ui_id ui_id;
char user[17], pass[9];
CMC_uint16 vers;
CMC_extension *extensions;
CMC_object_identifier charset;
CMC_string service;

/*********************************/
/* FIELDS FOR CMC_SEND_DOCUMENTS */
/*********************************/

char addresses[256], text[256], subject[80], titles[256],
     files[256], delimiter[2];

int main(int nump, char **parmz)
{
   cmcprintconfig();

   if(nump < 2 || strcmp(parmz[1], "SEND") == 0)
   {
      cmclogon();
      cmcsend();
      cmclogoff();
   }
   else
      cmcsenddocs();
}
```

```
void cmclogon(void)
{
/*************************************************************/
/* This won't be needed with cmc_send_documents, but it is */
/* needed with cmc_send.                                     */
/*************************************************************/

/* UI-ID is not used anyway, set to 0 */
   ui_id = 0;

/* USER must be 16 or less characters, terminated with a \0 */
   strcpy(user, "MY_USER");
/* NOTE: If you wish to pass a NULL user, any NULL character */
/*       pointer will work just fine...                      */

/* PASS must be 8 characters, terminated with a \0 */
   strcpy(pass, "PaSsWoRd");
/* NOTE: If you wish to pass a NULL user, any NULL character */
/*       pointer will work just fine here, too...            */

/* SERVICE, CHARACTER SET, AND EXTENSIONS ARE NOT USED, PASS NULLS. */
   service = NULL;
   extensions = NULL;
   charset = NULL;

/* Programmed to specification 1.00 */
   vers = 100;

/* Pass a 0 in FLGS because we don't support LOGON_UI, ERROR_UI, */
/* or COUNTED_STRING_TYPE.                                       */
   flgs = 0;

/* sessid is the session ID that must be used in all the other calls. */
   rc = cmc_logon(service, user, pass, charset, ui_id, vers, flgs,
&sessid,
                  extensions);

/* I am just printing the return, you will probably want to stop */
/* if you don't get a 0 return...                                */
   printf("LOGON = %d\n", rc);
}

void cmcsend(void)
{
/*****************************************************************/
/* There are two ways of sending a text file. First, there is the */
/* sending as a file method. Here is how to do this method:       */
/*****************************************************************/

/* FLGS is 0 because we don't support COUNTED_STRING_TYPE or any */
/* of the UIs...                                                 */
   flgs = 0;

/* To indicate that the message text is a file, we set the */
```

```
/* CMC_MSG_TEXT_NOTE_AS_FILE flag in the CMC_message       */
/* structure.                                              */

/* Set the CMC_MSG_TEXT_NOTE_AS_FILE flag */
   msg[0].message_flags = 0 | CMC_MSG_TEXT_NOTE_AS_FILE;

/* MESSAGE-REFERENCE, MESSAGE-TYPE are ignored. */
/* You may specify them if you wish...          */
   msg[0].message_reference = NULL;
   msg[0].message_extensions = NULL;
   msg[0].message_type = (CMC_string)malloc(9);
   strcpy(msg[0].message_type, "CMC: IPM");
/* NOTE: For the purpose of sending a new message, MESSAGE-REFERENCE */
/*       has no use whatsoever. Also, we currently don't bother to   */
/*       check the MESSAGE-TYPE field, although it may be useful in  */
/*       the future...                                               */

/* TEXT-NOTE is NULL in this case because the text is in a file. */
   msg[0].text_note = NULL;

/* TIME-SENT isn't used, but set it to 0 anyway... */
   msg[0].time_sent.second = 0;
   msg[0].time_sent.minute = 0;
   msg[0].time_sent.hour = 0;
   msg[0].time_sent.day = 0;
   msg[0].time_sent.month = 0;
   msg[0].time_sent.year = 0;
   msg[0].time_sent.isdst = 0;
   msg[0].time_sent.tmzone = 0;

/* SUBJECT is a string terminated by \0. */
   msg[0].subject = (CMC_string)malloc(80);
   strcpy(msg[0].subject, "Re: Geoff's behaviour...");

/* Set up attachment structure... */
   msg[0].attachments =
      (CMC_attachment *)malloc(sizeof(CMC_attachment)*2);

/* attach_title isn't used yet...                              */
/* Actually, you can go ahead and put one in, but currently, the */
/* NETMAIL engine won't use it...                              */
   msg[0].attachments[0].attach_title = (CMC_string)malloc(17);
   strcpy(msg[0].attachments[0].attach_title, "Some Text...");
   msg[0].attachments[1].attach_title = (CMC_string)malloc(17);
   strcpy(msg[0].attachments[1].attach_title, "Binary Data...");

/* attach_type isn't used yet...                               */
/* You can put one in, either CMC-ATT-OID-BINARY or -TEXT, but */
/* it will have no affect...                                   */
   msg[0].attachments[0].attach_type =
(CMC_object_identifier)malloc(17);
   strcpy(msg[0].attachments[0].attach_type, CMC_ATT_OID_TEXT);
   msg[0].attachments[1].attach_type =
(CMC_object_identifier)malloc(17);
```

```
   strcpy(msg[0].attachments[1].attach_type, CMC_ATT_OID_BINARY);

/* Set the ATT_LAST_ELEMENT flag in the second entry... */
   msg[0].attachments[0].attach_flags = 0;
   msg[0].attachments[1].attach_flags = 0 | CMC_ATT_LAST_ELEMENT;

/* attach_filename is a string terminated by a \0... */
   msg[0].attachments[0].attach_filename = (CMC_string)malloc(37);
   strcpy(msg[0].attachments[0].attach_filename, "FILE.GRP.ACCT01");
   msg[0].attachments[1].attach_filename = (CMC_string)malloc(37);
   strcpy(msg[0].attachments[1].attach_filename, "FILE.GRP.ACCT02");

/* We don't use attachment extensions... */
   msg[0].attachments[0].attach_extensions = NULL;
   msg[0].attachments[1].attach_extensions = NULL;

/* Load recipients with the address of our recipient array... */
   msg[0].recipients =
      (CMC_recipient *)malloc(sizeof(CMC_recipient)*3);

/* name and address must be strings terminated with a \0... */
   msg[0].recipients[0].name = (CMC_string)malloc(20);
   strcpy(msg[0].recipients[0].name, "JOHN Q. PUBLIC");
   msg[0].recipients[0].address = (CMC_string)malloc(20);
   strcpy(msg[0].recipients[0].address, "YOU@THERE.COM");
   msg[0].recipients[0].name_type = CMC_TYPE_INDIVIDUAL;

/* This one (entry 0) goes on the "TO:" list... */
   msg[0].recipients[0].role = CMC_ROLE_TO;

/* The recip_flags available are CMC_RECIP_IGNORE,       */
/* CMC_RECIP_LIST_TRUNCATED, and CMC_RECIP_LAST_ELEMENT. */
/* We don't need any of these for this recipient...      */
   msg[0].recipients[0].recip_flags = 0;
   msg[0].recipients[0].recip_extensions = NULL;

/* Same stuff for entry 1... */
   msg[0].recipients[1].name = (CMC_string)malloc(20);
   strcpy(msg[0].recipients[1].name, "JOHN DOE");
   msg[0].recipients[1].address = (CMC_string)malloc(20);
   strcpy(msg[0].recipients[1].address, "ME@HERE.COM");
   msg[0].recipients[1].name_type = CMC_TYPE_INDIVIDUAL;

/* This one (entry 1) goes on the "CC:" list... */
   msg[0].recipients[1].role = CMC_ROLE_CC;
   msg[0].recipients[1].recip_flags = 0;
   msg[0].recipients[1].recip_extensions = NULL;

/* Entry 2, same story (almost)... */
   msg[0].recipients[2].name = (CMC_string)malloc(20);
   strcpy(msg[0].recipients[2].name, "CAPTAIN KLUTZ");
   msg[0].recipients[2].address = (CMC_string)malloc(20);
   strcpy(msg[0].recipients[2].address, "BOSS@HERE.COM");
   msg[0].recipients[2].name_type = CMC_TYPE_INDIVIDUAL;
```

```
/* Entry 2 is goes on the "BCC:" list... */
   msg[0].recipients[2].role = CMC_ROLE_BCC;
   msg[0].recipients[2].recip_extensions = NULL;
/* Last recipient, set the flag... */
   msg[0].recipients[2].recip_flags = 0 | CMC_RECIP_LAST_ELEMENT;

/* 2nd message... */

/* NO ATTACHMENTS used in this message... */
   msg[1].message_reference = NULL;
   msg[1].attachments = NULL;
   msg[1].message_extensions = NULL;
   msg[1].message_type = (CMC_string)malloc(9);
   strcpy(msg[1].message_type, "CMC: IPM");

/* SUBJECT is a string terminated by \0. */
   msg[1].subject = (CMC_string)malloc(80);
   strcpy(msg[1].subject, "Re: Geoff's attitude...");

/* Notice that this time around, CMC_MSG_TEXT_NOTE_AS_FILE is */
/* *NOT* set, but I am setting CMC-MSG-LAST-ELEMENT...        */
   msg[1].message_flags = 0 | CMC_MSG_LAST_ELEMENT;

/* TIME-SENT isn't used, but set it to 0 anyway... */
   msg[1].time_sent.second = 0;
   msg[1].time_sent.minute = 0;
   msg[1].time_sent.hour = 0;
   msg[1].time_sent.day = 0;
   msg[1].time_sent.month = 0;
   msg[1].time_sent.year = 0;
   msg[1].time_sent.isdst = 0;
   msg[1].time_sent.tmzone = 0;

/* Load the message into text_note, terminated by a \0... */
   msg[1].text_note = (CMC_string)malloc(256);
   strcpy(msg[1].text_note, "NOTHING IN PARTICULAR\015\n");

/* Load recipients with the address of our recipient array... */
   msg[1].recipients = (CMC_recipient *)malloc(sizeof(CMC_recipient));
   msg[1].recipients[0].name = (CMC_string)malloc(20);
   strcpy(msg[1].recipients[0].name, "JOHN Q. PUBLIC");
   msg[1].recipients[0].address = (CMC_string)malloc(20);
   strcpy(msg[1].recipients[0].address, "YOU@THERE.COM");
   msg[1].recipients[0].name_type = CMC_TYPE_INDIVIDUAL;

/* This one (entry 0) goes on the "TO:" list... */
   msg[1].recipients[0].role = CMC_ROLE_TO;

/* Last recipient, set the flag... */
   msg[1].recipients[0].recip_flags = 0 | CMC_RECIP_LAST_ELEMENT;

   rc = cmc_send(sessid, msg, flgs, ui_id, extensions);
```

```
/* I am just printing the return, you will probably want to stop */
/* if you don't get a 0 return...                                */
   printf("SEND = %d\n", rc);
}

void cmcsenddocs(void)
{
/* SET UP ADDRESSES */
   strcpy(addresses, "YOU@THERE.COM,ME@HERE.COM,BCC:BOSS@HERE.COM");

/* SET UP TEXT */
   strcpy(text, "Thou art luck. I have sent the a document.");
/* NOTE: If you have no text, any NULL character pointer will do... */

/* SET UP SUBJECT */
   strcpy(subject, "ALMOST FREE UNLIMITED TIME OFFER!");
/* NOTE: If you have no subject, any NULL character pointer will do...
*/

/* SET UP TITLES */
   strcpy(titles, "Some Text,A Program");
/* NOTE: If you have no titles, any NULL character pointer will do... */

/* SET UP FILES */
   strcpy(files, "FILE.GRP.ACCT01,FILE.GRP.ACCT02");
/* NOTE: If you have no titles, any NULL character pointer will do... */

/* DELIMITER is a comma! */
   strcpy(delimiter, ",");

/* LEAVE FLAGS AT 0 THIS TIME. */
   flags = 0;

   rc = cmc_send_documents(addresses, subject, text, flags, files,
                           titles, delimiter, ui_id);

   printf("SEND_DOC = %d\n", rc);
}

void cmclogoff(void)
{
/* Set FLGS to 0 because we don't support any UIs... */
   flgs = 0;

   rc = cmc_logoff(sessid, ui_id, flgs, extensions);

/* I am just printing the return, you will probably want to stop */
/* if you don't get a 0 return...                                */
   printf("LOGOFF = %d\n", rc);
}

void cmcprintconfig(void)
{
   int ii;
```

```
CMC_session_id temp_id;
CMC_enum what, et;
CMC_extension *none;
CMC_string str;
CMC_object_identifier *oi;
CMC_boolean boo;
CMC_uint16 uint;

temp_id = 0; /* Not logged on... */
none = (CMC_extension *)NULL;

/* What character set? */
what = CMC_CONFIG_CHARACTER_SET;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&oi, none);
if(rc == 0)
{
   for(ii = 0; ii < 100 && oi[ii] != NULL; ii++)
   {
      printf("Char set: %s\n", oi[ii]);
      cmc_free(oi[ii]);
   }
   cmc_free(oi);
}
else
   printf("Char set: ???\n");

/* What line terminator? */
what = CMC_CONFIG_LINE_TERM;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&et, none);
printf("Line terminator: ");
if(rc == 0)
{
   switch(et)
   {
      case CMC_LINE_TERM_CRLF:
         printf("CR/LF\n");
         break;
      case CMC_LINE_TERM_LF:
         printf("LF\n");
         break;
      case CMC_LINE_TERM_CR:
         printf("CR\n");
         break;
      default:
         printf("BOGUS RETURN!\n");
   }
}
else
   printf("???\n");

/* What default service? */
what = CMC_CONFIG_DEFAULT_SERVICE;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&str, none);
if(rc == 0)
```

```
{
   printf("Default service: %s\n", str);
   cmc_free(&str);
}
else
   printf("Default service: ???\n");

/* What default user? */
what = CMC_CONFIG_DEFAULT_USER;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&str, none);
if(rc == 0)
{
   printf("Default user: %s\n", str);
   cmc_free(&str);
}
else
   printf("Default user: ???\n");

/* Password required? */
what = CMC_CONFIG_REQ_PASSWORD;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&et, none);
printf("Password: ");
if(rc == 0)
{
   switch(et)
   {
      case CMC_REQUIRED_NO:
         printf("Not required\n");
         break;
      case CMC_REQUIRED_OPT:
         printf("Optional\n");
         break;
      case CMC_REQUIRED_YES:
         printf("Required\n");
         break;
      default:
         printf("BOGUS RETURN!\n");
   }
}
else
   printf("???\n");

/* Service name required? */
what = CMC_CONFIG_REQ_SERVICE;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&et, none);
printf("Service name: ");
if(rc == 0)
{
   switch(et)
   {
      case CMC_REQUIRED_NO:
         printf("Not required\n");
         break;
      case CMC_REQUIRED_OPT:
```

```
         printf("Optional\n");
         break;
      case CMC_REQUIRED_YES:
         printf("Required\n");
         break;
      default:
         printf("BOGUS RETURN!\n");
   }
}
else
   printf("???\n");

/* User required? */
what = CMC_CONFIG_REQ_USER;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&et, none);
printf("User: ");
if(rc == 0)
{
   switch(et)
   {
      case CMC_REQUIRED_NO:
         printf("Not required\n");
         break;
      case CMC_REQUIRED_OPT:
         printf("Optional\n");
         break;
      case CMC_REQUIRED_YES:
         printf("Required\n");
         break;
      default:
         printf("BOGUS RETURN!\n");
   }
}
else
   printf("???\n");

/* UI available? */
what = CMC_CONFIG_UI_AVAIL;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&boo, none);
if(rc == 0)
   if(boo == CMC_TRUE)
      printf("UI: Available\n");
   else
      printf("UI: Not available\n");
else
   printf("UI: ???\n");

/* DO_NOT_MARK_AS_READ supported? */
what = CMC_CONFIG_SUP_NOMKMSGREAD;
rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&boo, none);
if(rc == 0)
   if(boo == CMC_TRUE)
      printf("DO_NOT_MARK_AS_READ supported: Yes\n");
   else
```

```
            printf("DO_NOT_MARK_AS_READ supported: No\n");
      else
         printf("DO_NOT_MARK_AS_READ supported: ???\n");

      /* COUNTED_STRING supported? */
      what = CMC_CONFIG_SUP_COUNTED_STR;
      rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&boo, none);
      if(rc == 0)
         if(boo == CMC_TRUE)
            printf("Counted string: Supported\n");
         else
            printf("Counted string: Not supported\n");
      else
         printf("Counted string: ???\n");

      /* Which version? */
      what = CMC_CONFIG_VER_IMPLEM;
      rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&uint, none);
      if(rc == 0)
         printf("Version (implementation): %d\n", uint);
      else
         printf("Version (implementation): ???\n");

      /* Which spec? */
      what = CMC_CONFIG_VER_SPEC;
      rc = cmc_query_configuration(temp_id, what, (CMC_buffer)&uint, none);
      if(rc == 0)
         printf("Version (specs): %d\n", uint);
      else
         printf("Version (specs): ???\n");
   }
```

The sample program in SPLash! (native mode SPL) (CMCSPLEX) that uses various CMC calls:

```
$CONTROL SEGMENT=CMC'EXAMPLE,NATIVE,SYMLEN=31
<< WARNING: The default SYMLEN is 15, and that will cut off >>
<<          the name of cmc_send_documents.                 >>
BEGIN
<< STORAGE >>
   BYTE ARRAY addresses(0:127), text(0:127), subject(0:80),
                      titles(0:127), files(0:127), delimiters(0:1),
                      buff(0:127);
   VIRTUAL BYTE POINTER n'file, n'title;
   DOUBLE flags, ui'id, rc;
   INTEGER len;
   INTRINSIC PRINT, DASCII;

<< EXTERNALS >>
   DOUBLE PROCEDURE cmc'send'documents(addr, sub, txt, flgs,
                                       fils, titls, dels, ui);
   VALUE                                              flgs,
                                                         ui;
   BYTE ARRAY                          addr, sub, txt,
                                       fils, titls, dels;
   DOUBLE                                             flgs,
                                                         ui;
   OPTION EXTERNAL,NATIVE;

<< MAIN CODE >>
main'code:
<< Addresses are assumed to be "TO:", this example uses one TO >>
<< and one CC. BCC is also available. Addresses are separated  >>
<< by commas, so A@B.C,B@B.C,CC:C@B.C,BCC:D@B.C,E@B.C would    >>
<< end up with A and B on the TO line, C on the CC line, with  >>
<< D and E getting blind carbons. Be sure to NULL terminate.   >>
   MOVE addresses:=("RCB@3K.COM,CC:TFK@3K.COM", 0);

<< IF YOU HAVE NO TEXT, PASS A NULL POINTER INSTEAD >>
   MOVE text:=("I am sending you a document.", 0);

<< THE SUBJECT OF THE MESSAGE >>
   MOVE subject:=("SPAM-O-RAMA!", 0);

<< IF YOU HAVE NO ATTACHMENTS, PASS A NULL STRING >>
<< OTHERWISE, SEPARATE THEM BY COMMAS...          >>
   titles:=files:=0;
<< MOVE titles:=("First attachment,Second attachment", 0);
   MOVE files:=("SPAM1.USGTFK.DEV3K,SPAM2.USGTFK.DEV3K", 0); >>

<< MY DELIMITER IS A COMMA >>
   MOVE delimiters:=(",",0);

<< NO SPECIAL FLAGS, NO UID >>
   flags:=0D;
   ui'id:=0D;
```

```
   rc:=cmc'send'documents(addresses, subject, text, flags, files,
                          titles, delimiters, ui'id);

<< ALTERNATIVE WAY OF PASSING NULLS: A NULL POINTER >>
<< @n'file:=0D;
   @n'title:=0D;
   rc:=cmc'send'documents(addresses, subject, text, flags, n'file,
                          n'title, delimiters, ui'id); >>

<< PRINT OUT THE RESULT >>
   len:=MOVE buff:="SEND_DOC = ";
   len:=len + DASCII(rc, 10, buff(len));
   PRINT(buff, -len, 0);
END.
```