Connect:Enterprise®
UNIX

Programmer's Guide

Version 2.3

Sterling Commerce
An IBM Company

Connect:Enterprise UNIX Programmer's Guide
Version 2.3

First Edition

Contents

Chapter 2

User Exits

User Exits in Connect:Enterprise can customize processing of user information and batches as corresponding events occur. Each User Exit is defined as a user modifiable C language subroutine. When an Exit point in the Connect:Enterprise process flow is reached, the corresponding User Exit subroutine is called. Each subroutine may be modified to customize the processing of the corresponding event.

There are thirteen User Exit subroutines provided with Connect:Enterprise. The subroutine entry points are located in the *$CMUHOME/exits/cmuexits.c* source code file. An ANSI compatible C language compiler is necessary to compile the cmuexits.c source code module after modifications have been made.

The use of User Exits is optional. No User Exit subroutines are enabled in Connect:Enterprise by default.

Using Exits in Connect:Enterprise

User Exits in Connect:Enterprise are called when specific events occur. For example, when a batch of data is successfully received by Connect:Enterprise, the Batch Receive Exit function is called with information corresponding to the batch. The User Exit may be customized to automate the processing of the batch.

Enabling User Exits

Enable specific exits by checking the appropriate check box in the **Define Configuration** function of the Site Administration Web interface. Connect:Enterprise supports the following exits:

Argument	Description
API Function Exit	invoked before any API function call. If this exit is enabled, all command line utilities (when executed) result in invocation of this exit function.

Argument	Description
Batch Receive Exit	invoked when a batch of data is added to the repository, either from a remote site (**cmuadd**) or from an API-generated add. Only compatible with batches of less than 2,147,483,647 bytes. If you are working with batches larger than this, use the Batch Receive 64 Exit. You cannot use both Batch Receive Exit and Batch Receive 64 Exit.
Batch Receive 64 Exit	invoked when a batch of data is added to the repository, either from a remote site (**cmuadd**) or from an API-generated add. You cannot use both Batch Receive Exit and Batch Receive 64 Exit.
Batch Send Exit	invoked when a batch of data is sent from the repository. Only compatible with batches of less than 2,147,483,647 bytes. If you are working with batches larger than this, use the Batch Send 64 Exit. You cannot use both Batch Send Exit and Batch Send 64 Exit.
Batch Send 64 Exit	invoked when a batch of data is sent from the repository. You cannot use both Batch Send Exit and Batch Send 64 Exit.
Log Exit	invoked before any Accounting Log Record is written.
Mailbox Initialization Exit	invoked when Connect:Enterprise starts execution.
Mailbox Termination Exit	invoked when Connect:Enterprise terminates execution.
Remote Command Exit	invoked when a remote site issues a command with either the $$ syntax or FTP commands.
Security Exit	invoked before access to the repository is given. Validates the user and password information.
Session Initial Buffer Exit	invoked when a session is started with a remote site using Bisync or non-interactive Async protocols.
Session Initialization Exit	invoked each time a remote site connects to Connect:Enterprise with a online protocol.
Session Termination Exit	invoked when a session is ended with a remote site using any one of the supported communication protocols (FTP, ASYNC, or Bisync). It is not invoked for autoconnect activity.

All Connect:Enterprise user exits are called using standard UNIX C calls. An Exit Parameter List is passed to each of the user exits.

Compiling the User Exit Functions

To use the exit program, follow this procedure:

1. Enable the appropriate user exits check box in the **Define Configuration** function of the Site Administration Web interface.
2. Open the skeleton exit file, *cmuexits.c*, located under the *$CMUHOME/exits* directory. The file contains housekeeping code followed by the exit function definitions. Each function definition is similar to the example shown:

```
long CMUEXIT_MboxInit(char *achMboxName);
{
printf("Initializing Mailbox System.\n");
return 0;
}
```

3. Select the exit call you have enabled in the MCD file.
4. Replace the *printf* line with your code.
5. When you are finished, save and exit *cmuexits.c*.
6. Issue a **ceushutdown** command.
7. At the UNIX prompt, use this command to compile the new code:

```
makeexits
```

The *makeexits* script and *makefile* compile *cmuexits.c* code.

Note: If you are using a compiler other than the XLC Compiler Version 1.3.0.xx on AIX or the Optional C Compiler on HP-UX, the *makeexits* command may fail with an unknown options error.

8. Issue a **ceustartup** command to restart the Connect:Enterprise system.

Header File Locations

The User Exit Function parameter lists consist of values and C structures defined in the *$CMUHOME/include/cmuexit.h, $CMUHOME/include/cmuexits.h* header files, and *$CMUHOME/src/samples.h*.

User Exit System Considerations

The User Exits are called as separate UNIX processes in an asynchronous manner for each invocation. Connect:Enterprise and the host system performance can degrade if the User Exits are implemented so that they slow batch processing or impede general functioning of Connect:Enterprise.

The processes coded for the User Exits should be kept as brief and simple as possible. An example would be to use the Batch Receive Exit to simply notify an external agent that a batch has arrived instead of extracting and processing the batch as part of the Batch Receive Exit function itself. This would keep the time and system requirements of the Batch Receive Exit to a minimum.

Modifying the CMUUSERLOG Utility

The cmuuserlog utility provides an alternative function to the Log Exit (cmulogd). The Log Exit (cmulogd) starts a subprocess for each record transmitted. The cmuuserlog utility allows you to monitor and manipulate the data in real time without starting a separate subprocess for each record.

Note: Modifying the cmuuserlog utility eliminates network level activity between the cmulogd and the cmuexitd programs. Because a new subprocess is not created for each user exit, fewer system resources are required to process your data.

The following steps enable you to modify File Agent to use this utility.

1. If the product is running, issue the **ceushutdown** command.
2. Modify *$CMUHOME/exits/userlog.c* and add your system-specific code.
3. Run the makeexits utility script to compile the User Exit functions and the cmuuserlog utility program.
4. Modify *$CMUHOME/etc/ceustartup* and add "*-S CMUHOME/arch/bin/cmuuserlog*" to the startup script for the cmulogd program.
5. Issue the **ceustartup** command.

API Function Exit

The API Function Exit is called each time an offline command or API command is performed.

The API Function Exit could be used to limit specific users from performing operations using offline commands and API programs.

Function Definition

The function of the API Function Exit follows:

```
long CMUEXIT_APIFunc(achOrigRemoteId, ulFunction)
    char *achOrigRemoteId;
    long ulFunction;
{
    return 0;
}
```

Arguments

Argument	Description
achOrigRemoteId	user ID performing a API or offline command.

Argument	Description
ulFunction	function code corresponding to the command being performed. Possible values are: ◆ APICMD_ADD ◆ APICMD_CONNECT ◆ APICMD_DELETE ◆ APICMD_ERASE ◆ APICMD_EXTRACT ◆ APICMD_LIST ◆ APICMD_REFRESH ◆ APICMD_DAEMON_REFRESH ◆ APICMD_SSLPASS_REFRESH ◆ APICMD_SSHPASS_REFRESH ◆ APICMD_SESSION ◆ APICMD_SHUTDOWN ◆ APICMD_START ◆ APICMD_STATUS ◆ APICMD_STOP ◆ APICMD_TRACE

Return Value

The API Function Exit returns zero for success and non-zero for failure. A return of zero indicates the operation is allowed to proceed. A non-zero return code indicates the operation is not allowed to proceed and a failure code is returned to the context where the command was initiated.

Batch Receive 64 Exit

The Batch Receive 64 Exit is called each time a batch of data is *successfully* deposited in the repository. The Batch Receive 64 Exit is called for all batches regardless of the communication protocol used to deposit the batch. This includes the online protocols and the offline API protocols.

The Batch Receive 64 Exit may be used to automate the processing of batches as they are received by Connect:Enterprise. A typical use of the Batch Receive 64 Exit might be to notify a external program that the batch has arrived. The external program would then process the batch asynchronously. You cannot use both Batch Receive Exit and Batch Receive 64 Exit.

Function Definition

The function of the Batch Receive 64 Exit follows:

```
long CMUEXIT_BatchRecv64(usProtocol, achOrigRemoteId, achMboxId, achBatchId,
                         ulBatchNo, ullBytes, ulStartTime, ulStopTime)
short usProtocol;
char *achOrigRemoteId;
char *achMboxId;
char *achBatchId;
long ulBatchNo;
long long ullBytes;
long ulStartTime;
long ulStopTime;
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. Value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.
achOrigRemoteId	mailbox logon used when the communication session was initiated. Value is passed for reference only.
achMboxId	logical mailbox where the batch was deposited. Value is passed for reference only.
achBatchId	batch ID associated with the batch of data. Value is passed for reference only.
ulBatchNo	batch number associated with the batch of data. Value is passed for reference only.
ullBytes	size of the batch in bytes. Value is passed for reference only.
ulStartTime	timestamp corresponding to the time the batch transmission started. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.
ulStopTime	timestamp corresponding to the time the batch transmission ended. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.

Return Value

Return value has no significance.

Batch Receive Exit

The Batch Receive Exit is called each time a batch of data is *successfully* deposited in the repository. The Batch Receive Exit is called for all batches regardless of the communication protocol used to deposit the batch. This includes the online protocols and the offline API protocols.

The Batch Receive Exit may be used to automate the processing of batches as they are received by Connect:Enterprise. A typical use of the Batch Receive Exit might be to notify a external program that the batch has arrived. The external program would then process the batch asynchronously.

The Batch Receive Exit is the same as the Batch Receive 64 Exit except that the Batch Receive Exit accurately reports only batch sizes less than 2,147,483,647 bytes. Batch sizes larger than 2,147,483,647 bytes are reported as zero. Because of this limitation, it is best to switch to the Batch Receive 64 Exit. You cannot use both Batch Receive Exit and Batch Receive 64 Exit.

Function Definition

The function of the Batch Receive Exit follows:

```
long CMUEXIT_BatchRecv(usProtocol, achOrigRemoteId, achMboxId, achBatchId,
                       ulBatchNo, ulBytes, ulStartTime, ulStopTime)
short usProtocol;
char *achOrigRemoteId;
char *achMboxId;
char *achBatchId;
long ulBatchNo;
long ulBytes;
long ulStartTime;
long ulStopTime;
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. This value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.
achOrigRemoteId	mailbox logon used when the communication session was initiated. Value is passed for reference only.
achMboxId	logical mailbox where the batch was deposited. Value is passed for reference only.
achBatchId	batch ID associated with the batch of data. Value is passed for reference only.
ulBatchNo	batch number associated with the batch of data. Value is passed for reference only.
ulBytes	size of the batch in bytes. Value is passed for reference only.

Argument	Description
ulStartTime	timestamp corresponding to the time the batch transmission started. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.
ulStopTime	timestamp corresponding to the time the batch transmission ended. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.

Return Value

Return value has no significance.

Batch Send 64 Exit

The Batch Send 64 Exit is called whenever a batch is sent from the repository. The Batch Send 64 Exit is called regardless of the communication protocol used.

The Batch Send 64 Exit could be used to notify a administrator that a batch was sent from the repository. You cannot use both Batch Receive Exit and Batch Receive 64 Exit.

Function Definition

The function of the Batch Send 64 Exit follows:

```
long CMUEXIT_BatchSend64(usProtocol, achOrigRemoteId, achMboxId, achBatchId,
ulBatchNo, ullBytes, ulStartTime, ulStopTime)

short usProtocol;
char *achOrigRemoteId;
char *achMboxId;
char *achBatchId;
long ulBatchNo;
long long ullBytes;
long ulStartTime;
long ulStopTime;
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. Value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.
achOrigRemoteId	mailbox logon used when the communication session was initiated. Value is passed for reference only.
achMboxId	logical mailbox where the batch was deposited. Value is passed for reference only.
achBatchId	batch ID associated with the batch of data. Value is passed for reference only.
ulBatchNo	batch number associated with the batch of data. Value is passed for reference only.
ullBytes	size of the batch in bytes. Value is passed for reference only.
ulStartTime	timestamp corresponding to the time the batch transmission started. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.
ulStopTime	timestamp corresponding to the time the batch transmission ended. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.

Return Value

Return value has no significance.

Batch Send Exit

The Batch Send Exit is called whenever a batch is sent from the repository. The Batch Send Exit is called regardless of the communication protocol used.

The Batch Send Exit could be used to notify a administrator that a batch was sent from the repository.

The Batch Send Exit is the same as the Batch Send 64 Exit, except that Batch Send Exit accurately reports only batch sizes less than 2,147,483,647 bytes. Batch sizes larger than 2,147,483,647 bytes are reported as zero. Because of this limitation, it is best to switch to the Batch Send 64 Exit. You cannot use both Batch Send Exit and Batch Send 64 Exit.

Function Definition

The function of the Batch Send Exit follows:

```
long CMUEXIT_BatchSend(usProtocol, achOrigRemoteId, achMboxId, achBatchId, ulBatchNo,
                       ulBytes, ulStartTime, ulStopTime)
short usProtocol;
char *achOrigRemoteId;
char *achMboxId;
char *achBatchId;
long ulBatchNo;
long ulBytes;
long ulStartTime;
long ulStopTime;
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. Value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.
achOrigRemoteId	mailbox logon used when the communication session was initiated. Value is passed for reference only.
achMboxId	logical mailbox where the batch was deposited. Value is passed for reference only.
achBatchId	batch ID associated with the batch of data. Value is passed for reference only.
ulBatchNo	batch number associated with the batch of data. Value is passed for reference only.
ulBytes	size of the batch in bytes. Value is passed for reference only.
ulStartTime	timestamp corresponding to the time the batch transmission started. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.
ulStopTime	timestamp corresponding to the time the batch transmission ended. Standard UNIX time format, may be processed with ctime(3) or localtime(3) for formatting. Value is passed for reference only.

Return Value

Return value has no significance.

Log Exit

The Log Exit is called whenever there is a Connect:Enterprise log record sent from any source to the cmulogd program. The cmulogd program manages writing the product activity log. The product

activity log is used for generating reports with the cmureport program. The Log Exit provides the capability of examining the log records before they are written to the product activity log.

The Log Exit could be used to intercept communication failures and perform notification when these events occur. The Log Exit is called in the following contexts:

- Remote Session Start – Remote Session Startup
- Remote Info – Remote Command Information
- Remote Session End – Remote Session Termination
- Autoconnect Session Start – Communication Session Started
- Autoconnect Remote Start – Transfer Start
- Autoconnect Command Information – Transfer Command Information
- Autoconnect Remote End – Transfer End
- Autoconnect Session End – Communication Session Terminated
- Offline Transaction (API Transactions and Offline Commands)
- Queued Autoconnect Transactions

Function Definition

The function of the Log Exit follows:

```
long CMUEXIT_Log(pAcctLogMsg)
LOG_MSG_T *pAcctLogMsg;
{
    return 0;
}
```

Arguments

Argument	Description
pAcctLogMsg	pointer to a Log Message structure. It is passed for reference only. Only selected elements of the structure are valid for a given Log Message context.

Return Value

A return value of zero (0) indicates success. A non-zero value indicates failure. If a return code of zero is returned the log record is written to the product activity log. If a non-zero value is returned the log record is not written to the product activity log.

Definition of the LOG_MSG_T Information Structure

The parameter **pAcctLogMsg** of the Log Exit points to the LOG_MSG_T structure.

```
typedef struct _LOG_MSG {
        USHORT  ulDeleteFlag;
        USHORT  usOpType;
        ULONG   ulRecordVersion;
        UCHAR   achRmtORLst[9];
        time_t  tStartDateTime;
        ULONG   ulImposedSessionNo;
        USHORT  usMsgType;
        time_t  tEndDateTime;
        USHORT  usSubMsgType;
        USHORT  usStatus;
        USHORT  usFuncCode;
        ULONG   ulProtocol;
        ULONG   ulBatchSize;
        ULLONG  ullBatchSize;
        ULONG   ulBatchNo;
        ULONG   ulCount1;
        ULONG   ulCount2;
        ULONG   ulCount3;
        ULONG   ulCount4;
        ULONG   ulCount5;
        ULONG   ulMsgNo;
        UCHAR   achRmtID[MC_RMTID];
        UCHAR   achBatchID[MC_BATCHID];
        UCHAR   achextraField[105];
        UCHAR   szConnResource[80];
        UCHAR   szbase;
} LOG_MSG_T;
```

The members of the LOG_MSG_T structure are only valid in some contexts. The following sections describe the log messages and the members of the LOG_MSG_T structure used for each.

LOG_MSG_T Values for Remote Session Start Log Messages

Remote Session Start log messages are recorded when a remote user initiates a communication session with Connect:Enterprise.

The following members are valid in the LOG_MSG_T structure used for a Remote Session Start log message. The number in parenthesis preceding the member name indicates the position that the value occupies in the 26-position pipe-delimited output of the cmureport command (**cmureport -d -v**).

Value	Description
usOpType	has a value of decimal 10 (LOG_RC_TYPE).
usMsgType	has a value of decimal 1 (LOG_STRT).
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
achRmtORLst	logon ID used by remote. It must correspond to a Remote Site Definition (RSD).
tStartDateTime	timestamp for session startup. It may be formatted with ctime(3) functions.

Value	Description
ulProtocol	protocol used to establish the connection with the repository. The value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.

LOG_MSG_T Values for Remote Session Information Log Messages

Remote Session Information log messages are recorded when a remote user performs a command to add, request, or delete a batch or when a directory command is performed.

The **add**, **request**, and **delete** remote command functions result in individual log messages for each batch affected by the command. The directory command results in a single log message regardless of how many batches are listed.

The following variables are valid members of the LOG_MST_T structure for Remote Session Information log messages.

Value	Description
usOpType	has a value of decimal 10 (LOG_RC_TYPE).
achRmtORLst	logon ID used by remote. It must correspond to an RSD.
tStartDateTime	timestamp for corresponding to the time the remote command started. It may be formatted with ctime(3) functions.
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
usMsgType	has a value of decimal 3 (LOG_RMTINFO).
tEndDateTime	Timestamp corresponding to the time the remote command ended. It may be formatted with ctime(3) functions.
usStatus	Status code for the remote command. Values are noted in *Appendix A, Error Messages*, in the *Connect:Enterprise Installation and Administration Guide*.
usFuncCode	function code corresponding to the operation performed by the remote command. The value is C_ADD, C_REQ, C_DEL, or C_DIR.
ulBatchSize	size in bytes of the batch affected by the remote command. Accurately reports batch sizes less than 2,147,483,647 bytes. Batch sizes larger than 2,147,483,647 bytes are reported as zero.
ullBatchSize	size in bytes of the batch affected by the remote command. Accurately reports all batch sizes.
ulBatchNo	batch number corresponding to the batch affected by the remote command.
achRmtID	mailbox ID corresponding to mailbox affected by the command issued by the remote user. It may or may not correspond to an RSD.
achBatchID	batch ID corresponding to the batch affected by the remote command.

LOG_MSG_T Values for Remote Session End Log Messages

Remote Session End log messages are recorded when a remote user disconnects a communication session with Connect:Enterprise.

The following variables are valid members of the LOG_MSG_T structure for Remote Session End log messages.

Value	Description
usOpType	has a value of decimal 10 (LOG_RC_TYPE).
achRmtORLst	logon ID used by remote. It must correspond to an RSD.
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
usMsgType	has a value of decimal 6 (LOG_END).
tEndDateTime	timestamp corresponding to the time the remote command ended. It may be formatted with ctime(3) functions.
usStatus	status code for the remote command. Values are noted in *Appendix A, Error Messages*, in the *Connect:Enterprise Installation and Administration Guide*.
long Count1	indicates number of batches added during a remote command session.
long Count2	indicates number of batches requested during a remote command session.
long Count3	indicates number of directory commands during a remote command session.
long Count4	indicates number of delete commands processed during a remote command session.
long Count5	indicates number of add commands performed during a Bisync or non-interactive Async session without a $$ADD card.

LOG_MSG_T Values for Autoconnect Session Start Log Messages

Autoconnect Session Start log messages are recorded when a connection to a remote site is initiated by Connect:Enterprise.

The following variables are valid members of the LOG_MSG_T structure for Autoconnect Session Start log messages.

Value	Description
usOpType	has a value of decimal 20 (LOG_AC_TYPE).
usMsgType	has a value of decimal 1 (LOG_STRT).
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
achRmtORLst	name of Autoconnect Definition (ACD) used for initiating the communication session.
tStartDateTime	timestamp for session startup. It may be formatted with ctime(3) functions.

LOG_MSG_T Values for Autoconnect Remote Start Log Messages

Autoconnect Remote Start log messages are recorded when a remote block defined in an ACD is processed and a communication session is attempted with a remote communication partner.

The following variables are valid members of the LOG_MSG_T structure valid for the Autoconnect Remote Start log messages.

Value	Description
usOpType	has a value of decimal 20 (LOG_AC_TYPE).
usMsgType	has a value of decimal 2 (LOG_RMTSTRT).
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
achRmtORLst	name of ACD used for initiating the communication session.
tStartDateTime	timestamp for session startup. May be formatted with ctime(3) functions.
achRmtID	value of the mailbox ID specified in the remote block used in the current phase of the auto connect attempt. This mailbox ID corresponds to an RSD.
ulProtocol	protocol used to establish the connection with the repository. The value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.

LOG_MSG_T Values for Autoconnect Remote Information Log Messages

Autoconnect Remote Information log messages are recorded each time a protocol command is performed as part of an autoconnect.

The following variables are members of the LOG_MSG_T structure valid for the Autoconnect Remote Information log messages.

Value	Description
usOpType	has a value of decimal 20 (LOG_AC_TYPE).
achRmtORLst	name of ACD used for initiating the communication session.
tStartDateTime	timestamp for corresponding to the time the command started. May be formatted with ctime(3) functions.
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
usMsgType	has a value of decimal 4 (LOG_RMTINFO).
tEndDateTime	timestamp corresponding to the time the command ended. May be formatted with ctime(3) functions.
short sSubMsgType	command performed. The value is C_ADD, C_REQ, C_DEL, or C_DIR.
usStatus	status code for the command. Values are noted in *Appendix A, Error Messages*, in the *Connect:Enterprise Installation and Administration Guide*.
ulBatchSize	size in bytes of the batch affected by the command. Valid for C_ADD and C_REQ. Accurately reports batch sizes less than 2,147,483,647 bytes. Batch sizes larger than 2,147,483,647 bytes are reported as zero.
ullBatchSize	size in bytes of the batch affected by the command. Valid for C_ADD and C_REQ. Accurately reports all file sizes.
ulBatchNo	batch number corresponding to the batch affected by the command. Valid for C_ADD, C_REQ and C_DEL.
achRmtID	mailbox ID corresponding to mailbox affected by the command. May or may not correspond to an RSD.
achBatchID	batch ID corresponding to the batch affected by the command. Valid for C_ADD, C_REQ and C_DEL.

LOG_MSG_T Values for Autoconnect Remote End Log Messages

Autoconnect Remote End log messages are recorded when a communication session defined by a remote block in an ACD is completed. This log message corresponds only to the end of the communication session with the individual remote and does not necessarily mean the end of the autoconnect session.

The following variables are valid members of the LOG_MSG_T structure valid for the Autoconnect Remote End log messages.

Value	Description
usOpType	has a value of decimal 20 (LOG_AC_TYPE).
achRmtORLst	name of ACD used for initiating the communication session.
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
usMsgType	has a value of decimal 5 (LOG_RMTEND).
tEndDateTime	timestamp corresponding to the time the command ended. May be formatted with ctime(3) functions.
short sSubMsgType	set to the overall status of the connection with the remote defined in **achRmtID**.
achRmtID	mailbox ID corresponding to the mailbox affected by the command. May or may not correspond to an RSD.

LOG_MSG_T Values for Autoconnect Session End Log Messages

Autoconnect Session End log messages are recorded when a connection to a remote site is terminated by Connect:Enterprise.

The following variables are valid members of the LOG_MSG_T structure for Autoconnect Session Start log messages.

Value	Description
usOpType	has a value of decimal 20 (LOG_AC_TYPE).
usMsgType	has a value of decimal 6 (LOG_END).
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
achRmtORLst	name of ACD used for initiating the communication session.
tEndDateTime	timestamp for session end. May be formatted with ctime(3) functions.
usStatus	the status of the entire autoconnect session. Values are noted in *Appendix A, Error Messages*, in the *Connect:Enterprise Installation and Administration Guide*.
ulCount1	The number of batches successfully transmitted
ulCount2	The Number of batches that failed to transmit
ulCount3	The number of batches collected from the remote
ulCount4	The number of batches that failed during collection.
achBatchID	batch ID if this was a manual autoconnect by specific batch ID.

LOG_MSG_T Values for Queued Autoconnect Log Messages

The Queued Autoconnect log messages are recorded when a communication session fails and is entered into the Autoconnect Requeue system. This log message is informational and indicates a change in the state of an autoconnect session from active to failed and is only seen if the ACD has been configured to requeue the autoconnect.

The following variables are valid members of the LOG_MSG_T structure for Queued Autoconnect log messages.

Value	Description
usOpType	has a value of decimal 30 (LOG_QAC_TYPE).
achRmtORLst	name of ACD used for initiating the communication session.
tStartDateTime	timestamp corresponding to the time the autoconnect entered Requeue state. May be formatted with ctime(3) functions.
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
usMsgType	has a value of decimal 3 (LOG_RMTINFO).
usStatus	status code for the command. Values are noted in *Appendix A, Error Messages*, in the *Connect:Enterprise Installation and Administration Guide*.
usFuncCode	function code corresponding to the operation performed by the remote command. The value is C_ADD, C_REQ, C_DEL, or C_DIR.
achRmtID	mailbox ID corresponding to the mailbox affected by the command. May or may not correspond to an RSD.
achBatchID	batch ID corresponding to the batch affected by the command. Valid for C_ADD, C_REQ, and C_DEL.

LOG_MSG_T Values for Offline Command Log Messages

The Offline Command log messages are recorded for all TCP/IP communication sessions including the offline batch manipulation commands and custom API Program implementations using the Connect:Enterprise API.

The following variables are valid members of the LOG_MSG_T structure for Offline Command log messages.

Value	Description
usOpType	has a value of decimal 40 (LOG_OFFLINE_TYPE).
achRmtORLst	name of ACD used for initiating the communication session.
tStartDateTime	timestamp for corresponding to the time the remote command started. May be formatted with ctime(3) functions.

Value	Description
ulImposedSessionNo	has a value corresponding to the communication session number used internally by Connect:Enterprise.
usMsgType	has a value of decimal 4 (LOG_INFO).
tEndDateTime	timestamp corresponding to the time the command ended. May be formatted with ctime(3) functions.
Count1	command performed. The value is 0 (add), 1 (extract), 2 (status), 3 (delete), or 4 (erase).
ulBatchSize	size in bytes of the batch affected by the command. Accurately reports batch sizes less than 2,147,483,647 bytes. Batch sizes larger than 2,147,483,647 bytes are reported as zero.
ullBatchSize	size in bytes of the batch affected by the command. Accurately reports all file sizes.
ulBatchNo	batch number corresponding to the batch affected by the command.
achRmtID	mailbox ID corresponding to the mailbox affected by the command. May or may not correspond to an RSD.
achBatchID	batch ID corresponding to the batch affected by the command.

Mailbox Initialization Exit

The Mailbox Initialization Exit is invoked after Connect:Enterprise has been started and initialized, but before any user or session activity is allowed. This exit is called only once.

The Mailbox Initialization Exit could be used to start other tasks needed for processing information while Connect:Enterprise is executing.

Function Definition

The function of the Mailbox Initialization Exit follows:

```
long CMUEXIT_MboxInit(achMboxName)
char *achMboxName;
{
    return 0;
}
```

Arguments

Argument	Description
achMboxName	value of the **System Name** parameter in the **Mailbox Control Definitions** file. Value is passed for reference only.

Return Value

The Mailbox Initialization Exit returns zero (0) for success and non-zero for failure. A return code of zero indicates the product startup should proceed. A non-zero return indicates File Agent startup should abort and no further processing performed.

Mailbox Termination Exit

The Mailbox Termination exit is invoked in the Control Daemon as part of termination processing. The exit is called only once.

The Mailbox Termination Exit could be used to terminate other tasks running in parallel with Connect:Enterprise.

Function Definition

The function of the Mailbox Termination Exit follows:

```
long CMUEXIT_MboxTerm(achMboxName)
char *achMboxName;
{
    return 0;
}
```

Arguments

Argument	Description
achMboxName	value of the **System Name** parameter in the **Mailbox Control Definitions** file. Value is passed for reference only.

Return Value

Return value has no significance.

Remote Command Exit

The Remote Command Exit is called each time a mailbox command is issued in a remote connect communication session.

There are four basic commands performed by remote sites regardless of the protocol. The commands are add, request, list, and delete. Each time one of these operations is requested by a remote site, the Remote Command Exit is called to authenticate the request. All operations are allowed by default.

The Remote Command Exit may be used to implement a high granularity Access Control mechanism within Connect:Enterprise, over and above what is available with the built-in ACL and SECURITY=Batch mechanisms.

A simple use for the Remote Command Exit would be to limit access of remote mailbox users to the mailbox ID they used as a logon when the communication session was initiated.

Function Definition

The function of the Remote Command Exit follows:

```
long CMUEXIT_RmtCmd(usProtocol, usCommand, usRmtConn_Auto_Conn,
                    achOrigRemoteId, pParmCtlBlk, ulDataSize,
                    achDataBuffer)
short usProtocol;
short usCommand;
short usRmtConn_AutoConn;
char *achOrigRemoteId;
PARMCTLBLK_T (*pParmCtlBlk);
long ulDataSize;
char *achDataBuffer;
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. Valid values are P_BSC (Bisync), P_ASYNC (Async), P_FTP (non-secure FTP), and P_FTPS (secure FTP). Value is passed for reference only.
usCommand	command issued. The value is RMTCMD_ADD, RMTCMD_REQ, RMTCMD_DIR, or RMTCMD_DEL. (Defined in *cmuexit.h*)
usRmtConn_AutoConn	0 – autoconnect
1 – remote connect	Currently only set to one (1) for all contexts.

Argument	Description
achOrigRemoteId	File Agent logon used when the communication session was initiated. Value is passed for reference only.
pParmCtlBlk	pointer to a structure of information describing the batch. (Illustrated in *cmuexit.h)*. If a remote user is using FTP and issues an **mget** command, the ParmsCtlBlk->achBchId field will contain the batch number instead of the batch ID of the batch being requested.
ulDataSize	unused.
achDataBuffer	unused.

Return Value

The Remote Command Exit returns zero (0) for success and a non-zero value for failure. If a value of zero is returned the requested operation is allowed to continue. If a non-zero value is returned the requested operation is disallowed.

PARMCTLBLK_T

The parameter **pParmCtlBlk** of the Remote Command Exit points to the PARMCTLBLK_T structure. The PARMCTLBLK_T structure has all of the internal Connect:Enterprise specific information associated with the batch being processed.

```
typedef struct_PARMCTLBLK {
    short sCommand;
    char achId[MC_RMTID];
    char achBchId[MC_BATCHID];
    char achPassword[MC_PASSWORD];
    char achRecvFile[MC_RECVFILE]; /* PI:745101 */
    unsigned short usBlock;
    long lNap;
    long lRetries;
    char chRecSep[1];
    char chMedia[1];
    char chBchSep[1];
    char chCode[1];
    short sConv;
    char achCCList[MC_CCLIST];
    long lFlags;
    char achSearchOrigId[9];
    char achSearchFlags[9];
    time_t tFromTime;
    time_t tEndTime;
    long lFlagsSupplied;
} PARMCTLBLK_T;
```

The following variables are valid members of PARMCTLBLK_T for the Remote Command Exit.

Variable	Description
sCommand	commands issued. Valid values are C_ADD (1), C_REQ (2), C_DIR (3), C_DEL (4), C_NOOP (5).
achId	mailbox ID where batch resides or is being added to.
achBatchId	batch ID associated with the batch being processed.

Security Exit

The Security Exit is called each time a user authentication is performed in Connect:Enterprise. This includes both online communication sessions, offline mailbox commands, and API commands.

For the FTP and Secure FTP protocols, the Security Exit is called after the user responds to the logon and password prompts and before the logon information compared to an RSD (Remote Site Definitions file).

For interactive Async, the Security Exit is called after the user responds to the logon and password prompts and before the logon information is compared to an RSD.

For non-interactive Async and Bisync, the Security Exit is called after the interpretation of the first control card in the data stream and before the information is compared to an RSD.

The Security Exit could be used to authenticate the ID and password parameters against a external database. The Security Exit could also be used to map the given ID and password parameters to valid IDs and passwords for mailbox.

Function Definition

The function of the Security Exit follows:

```
long CMUEXIT_Security(id, passwd, protocol)
char *id;
char *passwd;
unsigned long protocol;
{
    return 0;
}
```

Arguments

Argument	Description
id	logon user ID. Value may be modified in the Security Exit. Maximum allowed length for the replacement is 128 characters.
passwd	password corresponding to ID. Value may be modified in the Security Exit. Maximum length for the replacement is 128 characters.
protocol	protocol used to establish the connection with File Agent. Value is passed for reference only. Following is a list of values. You can also find this list in samples.h in the $CMUHOME/samples directory.

Value	Protocol	Associated Command Type
1	TCPIP	Offline commands + API
2	ASYNC	Async
3	FTP	FTP
4	BSC	Bisync
6	FTPS	Secure FTP
7	HTTP	AS2 HTTP
8	EDIINT	AS2 EDI
9	GIS	GIS BP
10	SSH	SSH

Return Value

The Security Exit returns zero (0) for success and non-zero for failure. A return code of zero indicates success and the logon should be allowed to proceed. This does not indicate that the logon is valid to Connect:Enterprise, just that the logon information presented to the Security Exit has passed the scrutiny of the Exit. A non-zero return code indicates the logon information failed the scrutiny of the Security Exit and the logon attempt is returned to the user as failed.

Session Initial Buffer Exit

The Session Initial Buffer Exit is called only for Bisync and non-interactive Async remote connections. The purpose of the Exit is to allow modification of the data buffer before logon validation is performed.

The Session Initial Buffer Exit is called as soon as the first buffer of data is received for a Bisync or a non-interactive Async remote connection and before any examination of the data for control cards is performed. The Exit is called prior to the Session Initialization Exit. This Exit could be used to map non-File Agent logon cards to Connect:Enterprise control card syntax and would be especially useful in situations where legacy logon information needs to be mapped to Connect:Enterprise control cards.

Function Definition

The function of the Session Initial Buffer Exit follows:

```
long CMUEXIT_SessInitBuff(Protocol, BufSize, Buffer)
unsigned short Protocol;
unsigned long *BufSize;
unsigned char *Buffer;
{
      return 0;
}
```

Arguments

Argument	Description
Protocol	protocol used to establish the connection with File Agent. Any values can be present, but only P_BSC (Bisync) and P_ASYNC (Async) are valid. Value is passed for reference only.
BufSize	size of buffer. Value may be modified if the size of the buffer changes. Maximum value is 4096 bytes.
Buffer	pointer to the storage area containing the first buffer of data. The buffer may be modified. If the size changes, the new size must be reflected in BufSize.

Return Value

A return value of zero (0) indicates success. A non-zero value indicates failure. If a return code of zero is returned the session is allowed to proceed. If a non-zero value is returned the session is disconnected.

Session Initialization Exit

The Session Initialization Exit is called each time a Remote Site connects to Connect:Enterprise with a online protocol. These protocols are Async, Bisync, FTP, and Secure FTP.

The Session Initialization Exit may be used to perform custom remote user authentication. The Exit is called after the first buffer is parsed for control cards. The Session Initialization Exit is called for all remote connects; however, Bisync and non-interactive Async sessions pass the **ulDataSize** and **achDataBuffer** parameters. These two parameters may be modified inside of the Exit and the modified values will be returned to the calling process context. The modified size and buffer contents will be used to replace the first buffer of data sent by the remote user.

Note: The Session Initialization Exit is only called when remotes connect to Connect:Enterprise. The exit is not called for autoconnect.

Function Definition

The function of the Session Initialization Exit follows:

```
long CMUEXIT_SessionInit(usProtocol, achOrigRemoteId, achPassword,
                         ulAddress, ulDataSize, achDataBuffer)
short usProtocol;
char *achOrigRemoteId;
char *achPassword;
long ulAddress;
long ulDataSize;
char *achDataBuffer
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. Value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.
achOrigRemoteId	login ID used to authenticate to Connect:Enterprise. Value is passed for reference only.
achPassword	password used to authenticate to Connect:Enterprise. Value is passed for reference only.
ulAddress	(FTP only) TCP/IP address of remote. Value is passed for reference only.
ulDataSize	(non-interactive Async and Bisync only) size of **achDataBuffer**. Value may be modified if the size of **achDataBuffer** changes.
achDataBuffer	(non-interactive Async and Bisync only) first buffer passed in protocol connection from the remote.

Return Value

A return value of zero (0) indicates success. A non-zero value indicates failure. If a return code of zero is returned the session is allowed to proceed. If a non-zero value is returned the session is disconnected.

Session Termination Exit

The Session Termination Exit is called when a remote connect communication session terminates, regardless of whether the session was successful or not.

The Session Termination Exit could be used to process batches deposited during the communication session.

Syntax

The function of the Session Termination Exit follows:

```
long CMUEXIT_SessionTerm(usProtocol, achOrigRemoteId, usStatus, usReason)
short usProtocol;
char *achOrigRemoteId;
short usStatus;
short usReason;
{
    return 0;
}
```

Arguments

Argument	Description
usProtocol	protocol used to establish the connection with File Agent. Value is passed for reference only. Refer to samples.h in the $CMUHOME/samples directory for a list of valid values.
achOrigRemoteId	logon ID used for the remote connect session.
usStatus	unused.
usReason	unused.

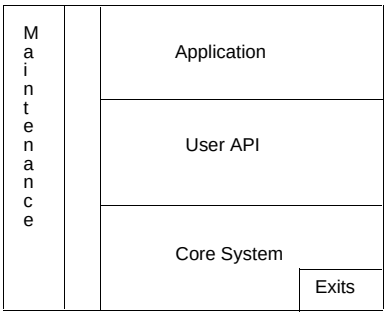
Return Value

Return value has no significance.

Chapter 3

API Calls

Connect:Enterprise Application Program Interfaces (API) are standard C function calls that can be embedded in an application program. The Sterling Commerce-supplied API subroutines are responsible for performing inter-process communication with the appropriate Connect:Enterprise components and report errors or completions back to the calling application. All the command line utilities supplied with Connect:Enterprise (such as **cmuadd** and **cmuextract,** with the exception of **cmureport**) were developed using this API layer.



An application program can communicate with either a host Connect:Enterprise system through a TCP/IP protocol, or with any Connect:Enterprise system in the TCP/IP network. Also, the application program can be distributed to any computer in the TCP/IP network when talking to a host Connect:Enterprise system. A TCP/IP interface is used to communicate between the API subroutines and the online Connect:Enterprise daemons that perform the work.

The Connect:Enterprise APIs enable users to incorporate Connect:Enterprise commands into their own applications. It gives the user the ability to add data to the repository, extract data from the repository, and perform other functions supported by Connect:Enterprise command line utility programs. The utility programs are described in the *File Agent Installation and Administration Guide.*

For example, if you are designing a custom payroll program, you might have to collect weekly payroll reports from various remote sites. To gather the payroll information, you would use Connect:Enterprise APIs. All sample programs are available in the *$CMUHOME/src* directory.

Only three API calls are needed to access Connect:Enterprise and the command utilities. These APIs are:

- CMUAPI_OpenSession
- CMUAPI_Command
- CMUAPI_CloseSession

A prototype for each of these APIs is available in the *samples.h* file residing in the *$CMUHOME/src* directory. Several of the sample C files provided in the *$CMUHOME/src* directory call these functions. Refer to these C files as examples.

To use these APIs:

1. Write your application using C.
2. Use the **CMUAPI_OpenSession** call to connect to the Connect:Enterprise control daemon at the point where you want to insert the Connect:Enterprise instructions. A Connect:Enterprise session handle is returned by this API call which is used with any subsequent calls.
3. Insert the **CMUAPI_Command** call for each Connect:Enterprise utility command you need to issue, supplying the appropriate session handle. Use any of these arguments:
   - APICMD_ADD
   - APICMD_CONNECT
   - APICMD_DELETE
   - APICMD_ERASE
   - APICMD_EXTRACT
   - APICMD_LIST
   - APICMD_REFRESH
   - APICMD_DAEMON_REFRESH
   - APICMD_SSLPASS_REFRESH
   - APICMD_SSHPASS_REFRESH
   - APICMD_SESSION
   - APICMD_SHUTDOWN
   - APICMD_START
   - APICMD_STATUS
   - APICMD_STOP
   - APICMD_TRACE
4. Close your session with Connect:Enterprise with a **CMUAPI_CloseSession** call.
5. Finish and compile your application.

Shared Objects

The File Agent API uses a shared object for communicating with the File Agent server. The API will not work without this object. The shared objects are:

- libcmusips.so (for Solaris, Linux, and AIX)
- libcmusips.sl (for HP-UX)

To use the shared object, set the environment variables to specify the location of the shared object. Use the following table as a guide:

Operating System	Environment Variable	Location of Shared Object
AIX	LIBPATH	$CMUHOME/aix/lib/
HP-UX	LD_LIBRARY_PATH, SHLIB_PATH	$CMUHOME/hpux/lib/
Linux	LD_LIBRARY_PATH	$CMUHOME/linux/lib/
Solaris	LD_LIBRARY_PATH	$CMUHOME/sun/lib/

If you run API programs on a remote computer (any computer other than the computer Connect:Enterprise is running on), copy the shared object to the remote computer and set the environment variables appropriate for the operating system on the remote computer.

Internal Message Encryption

If your API communicates with an instance of File Agent that has internal message encryption enabled, the sipskey file must be available on the computer where your API is running. Refer to the *Encrypting Internal Product Communications* chapter of the *Connect:Enterprise UNIX Installation and Administration Guide* for more information.

Tracing API Activity

When your API program is running, you can trace the activity of CMUUAPI. This allows you to view SIPS activity for purposes of debugging. Use the following procedure:

1. In your program that calls CMUUAPI, define variables for dbgfd and dbglvl as follows:

```
extern int dbgfd, dbglvl;
```

2. Open a file to which the trace output should go, before calling the API.
3. Set dbgfd with the file descriptor of the file you opened in step 2.
4. Set dbglvl to the desired trace level (0-99) into the dbglvl variable. When choosing your debug level, consider that debug levels can affect performance.

Following is an example:

```
if ( IsNum ( optarg) )
      dbglvl = atoi(optarg);
if ( dbglvl >= 0 ) {
       sprintf( debugfn, "cmuadd.out.%d", getpid() );
       dbgfd = open(debugfn, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
   }


extern int dbgfd, dbglvl;/* Connect:Enterprise UNIX API debugging variables */
char debugfn[20];
dbglvl = 9;/* 5 = SIPS summary, 9 = SIPS detail, 99 = all */
sprintf( debugfn, "cmuadd.out.%d", getpid() );
dbgfd = open(debugfn, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
```

CMUAPI_OpenSession

The **CMUAPI_OpenSession** call starts a session with a Connect:Enterprise (host or remote).

The **CMUAPI_OpenSession** API returns a handle of type APISESSION, which must be supplied as input to calls CMUAPI_Command and CMUAPI_CloseSession. Refer to *$CMUHOME/src/samples.h* for details.

Note: An application can have multiple session handles opened by calling open session API multiple times. At the end of the application, all open sessions must be closed.

Function Definition

The function of **CMUAPI_OpenSession** follows:

```
APISESSION  *CMUAPI_OpenSession (

    char    *szSystem,        /* Connect:Enterprise (see *szMailbox).*/
    USHORT  usPort,           /* Port of Connect:Enterprise. */
    char    *szUser,          /* Connect:Enterprise User ID.*/
    char    *szPassword,      /* Connect:Enterprise Password.*/
    char    *Reserved         /* This should always be set to NULL.*/
 )
```

Arguments

Argument	Description
szSystem	string that represents the Internet address of the host on which control daemon is running. (This can include a dot notation or a domain name.)
usPort	port number on which the *cmuctld* daemon is listening.
szUser	pointer to an ASCII null-terminated string that identifies the Connect:Enterprise user ID of the individual executing the API call. This information and the **szPassword** parameter are passed to both the Connect:Enterprise and any user-supplied security routines for validation and logging. The maximum length of **szUser** is 8 bytes plus one null byte.
szPassword	pointer to an 64-character ASCII null-terminated string containing the password for the user specified in the szUser parameter. The length of the string should not exceed 64 bytes because this is the maximum password length supported by Connect:Enterprise.
Reserved	always set to NULL.

Return Value

The return value, a structure of type APISESSION (defined in *$CMUHOME/src/samples.h*) defines an API session handle to the communication channel to a remote or host Connect:Enterprise. External variables such as APIerrno will contain error values which identify the failure of a specific function call and isolate a cause for the failure.

Example Code

The example code for **CMUAPI_OpenSession** follows:

```
/* Open User API session*/

       ApiSession =CMUAPI_OpenSession("mailbox.host.com",
                                      8000,
                                      "mailbox",
                                      "password",
                                      NULL);
       if ( ApiSession == NULL ) {
            printf("ERROR: Unable to open a session. ");
            exit( -1);
       }
```

CMUAPI_CloseSession

The **CMUAPI_CloseSession** call ends a session with a host or remote user.

Function Definition

The function of **CMUAPI_CloseSession** follows:

```
int CMUAPI_CloseSession (

   APISESSION  *ApiSessionHandle   /* Session value returned from
                                      CMUAPI_OpenSession */
);
```

Arguments

Argument	Description
ApiSessionHandle	specifies a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise. It is of type APISESSION (defined in *$CMUHOME/src/samples.h*).

Return Value

After the command is executed, a code is returned to reflect the status of the operation. This value is either APIRC_OK, indicating success, or an error value placed in the external variables CMUErrno and APIerrno.

Example Code

The example code for **CMUAPI_CloseSession** follows:

```
if(( rc = CMUAPI_CloseSession ( ApiSession))  != 0){
          printf("\n Close Session Failed");
          exit ( -1);
}
```

CMUAPI_Command

The **CMUAPI_Command** API call issues Connect:Enterprise commands to the host Connect:Enterprise system or to any Connect:Enterprise system in the TCP/IP network.

Use a new **CMUAPI_Command** call for each Connect:Enterprise command required. For example, to invoke both **cmuadd** and **cmuextract**, include two **CMUAPI_Command** calls and their associated arguments within your application code.

The application program fills in the **szCommand** area and other required parameters, then calls the API. The API first validates the parameters and then the supplied command syntax and values.

Function Definition

The function of **CMUAPI_Command** follows:

```
int CMUAPI_command(
   APISESSION    *ApiSessionHandle,     /* Session Handle */
   ULONG          ulApiCmd,             /* Command Code to execute */
   ...                                  /* Variable argument list */
)
```

Arguments

Argument	Description
ApiSessionHandle	specifies a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise.
ulApiCmd	long value that contains the command you want to execute. Valid values are: ◆ APICMD_ADD ◆ APICMD_CONNECT ◆ APICMD_DELETE ◆ APICMD_ERASE ◆ APICMD_EXTRACT ◆ APICMD_LIST ◆ APICMD_REFRESH ◆ APICMD_DAEMON_REFRESH ◆ APICMD_SSLPASS_REFRESH ◆ APICMD_SSHPASS_REFRESH ◆ APICMD_SESSION ◆ APICMD_SHUTDOWN ◆ APICMD_START ◆ APICMD_STATUS ◆ APICMD_STOP ◆ APICMD_CEUTRACE
...	variable-length argument list where the list depends on the previous parameter, **ulApiCmd**. Each command description follows.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIerrno (an unsigned long). The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample code in your *$CMUHOME/src* directory to see an example of the **CMUAPI_Command**. Various samp*.c files in this directory call **CMUAPI_Command**.

APICMD_ADD

Use the **APICMD_ADD** argument in the **CMUAPI_Command** API call to add batches to the host Connect:Enterprise system or to any Connect:Enterprise system in the TCP/IP network.

Specify batch attributes by providing values for **szMailboxId**, **szBatchId**, **lProcessFlags**, and **lDataFormatFlags**. Data generated by the application is passed through the **cbGetFileData()** callback function. Batch information returned from the Mailbox Engine is provided to the application through the **cbGetBatchInfo()** callback function. The **cbGetFileData()** is required. The **cbGetBatchInfo** callback is optional (its value can be set to null).

The **cbGetBatchInfo()** and **pBatchInfo** structure may appear to be redundant, but are not. The **cbGetBatchInfo()** function is called immediately after Connect:Enterprise allocates a batch slot in its database (each batch number is unique) and before any batch data has been added to Connect:Enterprise. The **pBatchInfo** structure contains all the batch information and is available after the add operation is complete.

Note: It is possible to add batches with batch id fields containing characters that may act as meta characters in the UNIX shell environment and the operating system environments where the batches are to be sent. If non-alphanumeric characters are used in batch IDs, take appropriate precautions in any context where the batch ID will be exposed to a UNIX shell or other scripting type of environment.

Function Definition

The function of **APICMD_ADD** follows:

```
int CMUAPI_Command(
 APISESSION *ApiSessionHandle, /* Session Handle  */
 ULONG      ulApiCmd,          /* Must be equal to APICMD_ADD  */
 CALLBACK   cbGetFileData,     /* a user defined callback function that passes
                                  data to the Mailbox Engine */
 void       *pGetFileDataArg,  /* a void pointer to a user-defined structure
                                  that will pass to the above callback function
                                  as an argument  */
 CALLBACK   cbGetBatchInfo,    /* a user defined callback function that API call
                                  passes batch information as an Argument */
 void       *pGetBatchInfoArg, /* a void pointer to a user-defined structure
                                  that will be passed to the above callback
                                  function as an argument */
 MBOXBATCH_INFO_T *pBatchInfo, /* an address of BATCHINFO structure */
 char      *szMailboxId,       /* First mailbox ID (max size of 8)where the
                                  batch will be added. See also *szCCList. */
 char      *szBatchId,         /* User Batch ID (Max of 64) of the Added batch */
 ULONG     lProcessFlag,       /* Processing Flags Bitmap */
 ULONG     lDataFormatFlag,    /* Data Format Flags Bitmap (ASCII, EBCDIC,
                                  BINARY) */
 USHORT    keepadd,            /* Do not remove $$ADD card from data */
 int       splitcount,         /* divide this batch into multiple batches with
                                  maximum size equal to this count. */
 char      *szUserRecord,      /* A string that points to a file where Macro
                                  Substitution is specified */
 char      *szLink,            /* Batch Number or External file to be linked to */
 ULONG     lUserFlags,         /* Reserved (Unused) */
 USHORT    sEncrypt,           /* If Not Zero perform Encryption.*/
 char      *achKey,            /* User supplied Encryption key. */
 char      *szCCList,          /* Identifies additional recipients(mailbox IDs)
                                  for the batch (max size of 256)*/
 int       iTrigger            /* Boolean variable (must be 1 or 0)to indicate
                                  whether to invoke automatic routing
                                  functionality.*/
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise.
ulApiCmd	long value that contains the command to be executed.

Argument	Description

cbGetFileData	user-defined callback function that passes data to the Mailbox Engine. This function is called multiple times until it returns 0. The return value of 0 indicates to Connect:Enterprise that an application does not have any more data to be written to the repository. The function returns the size of the buffer. Function syntax is as follows:
	`int cbGetFileData(char **buffer,`
	`void *pGetFileDataArg);`
	`char **buffer, /* A pointer to the location of the data */`
	`void *pGetFileDataArg /* A pointer to a user defined structure */`

pGetFileDataArg	void pointer indicating a user-defined structure that is passed to the above callback function as an argument. It enables users to pass information to the callback function instead of using global variables.

cbGetBatchInfo	user-defined callback function that the API call passes batch information to as an argument. This function is called for each added batch. The syntax is shown as follows:
	`int cbGetBatchInfo(MBOXBATCH_INFO_T *pBatchInfo,`
	`void *pGetBatchInfoArg);`
	`MBOXBATCH_INFO_T *pBatchInfo /* A pointer to the batch info */`
	`void *pGetBatchInfoArg, /* A pointer to a user defined structure */`

pGetBatchInfoArg	void pointer indicating a user-defined structure that is passed to the above callback function as an argument.
pBatchInfo	address of a MBOXBATCH_INFO_T structure that contains current added batch information.
szMailboxId	mailbox ID (1–8 characters) of the added batches. Connect:Enterprise searches the repository for all batches that match the submitted Mailbox ID. Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.
szBatchId	user batch ID (1–64 characters) of the added batch. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks.
lProcessFlag	processing flags bitmap (such as FLG_MULTXMIT and FLG_XMITONCE). See *$CMUHOME/src/samples.h* for definitions of flags.
lDataFormatFlag	data format flags bitmap (such as FLG_ASCII, FLG_EBCDIC, and FLG_BINARY). See *$CMUHOME/src/samples.h* for definitions of flags.
keepadd	indicates that the program must not remove **$$ADD** cards from data if the data is not zero.
splitcount	indicates that the program must divide this batch into multiple smaller batches, each containing <splitcount> bytes. The value must be at least 512; otherwise APIRC_SPLIT_TOO_SMALL is returned. Or, specify 0 to disable splitting.
szUserRecord	string that points to a file where Macro substitution is specified.
szLink	batch number or external file to be linked to.
lUserFlags	reserved (unused).

Argument	Description
sEncrypt	perform encryption if not equal to zero.
achKey	user-supplied encryption key.
szCCList	character string, which can be up to 256 characters in length, can contain a comma-separated list of additional mailbox IDs to which the batch will be added. If this string is left blank, then the batch will only be added to the mailbox ID specified in the **szMailboxId** field.
iTrigger	indicates whether triggering is enabled. This integer can only have two values: 0 and 1. If 0, **cmuadd** is performed without -t specified. If 1, **cmuadd** is performed with -t specified. Use this parameter to tell the mailbox daemon to invoke the automatic routing function after the batch has been added. For this feature to work, an ACD file must be configured with the **CONTACT** parameter set to **Forward data to Remote site automatically**, and the **Send ID** parameter should specify an ID that is either **szMailboxId** or one of the IDs specified in the **szCCList**. Refer to the *File Agent Installation and Administration Guide* for more information about automatic routing.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIerrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampadd.c in your *$CMUHOME/src* directory to see an example of the **APICMD_ADD** command.

APICMD_CONNECT

Use the **APICMD_CONNECT** argument in the **CMUAPI_Command** API call to trigger an autoconnect session of a specific Autoconnect Definition (ACD) from the host Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Specify the ACD name as one of the parameters. You can override certain parameters in the ACD such as **Retries**, **Mailbox ID**, and **Batch ID**. Some parameters allow the application to retrieve auto connect messages. Auto connect messages from the auto connect subsystem transfer to the application through the **cbGetInteractMsgs()** callback function. **cbGetInteractMsgs()** is required if the sInteractive option is set.

Function Definition

The function of **APICMD_CONNECT** follows:

```
int CMUAPI_command(
 APISESSION *ApiSessionHandle,  /* Session Handle */
 ULONG      ulApiCmd,           /* Must be set to APICMD_CONNECT */
 CALLBACK   cbGetInteractMsgs,  /* a user defined callback function that
                                   retrieves messages from the Mailbox
                                   AutoConnect subsystem */
 char       *szAcdList,         /* AutoConnect list name (1 - 8) */
 char       *szMailboxId,       /* Mailbox ID (1 - 8) of the selected batches */
 char       *szBatchId,         /* User Batch ID (1 - 64) of the selected
                                   batches */
 USHORT     sACMode,            /* AutoConnect Mode (SendRecv, RecvSend,
                                   SendOnly,RecvOnly) */
 USHORT     sOneBatch,          /* Set the OneBatch option on a $REQ operation */
 USHORT     sRetry,             /* Override the retry parameter  */
 USHORT     sInterval,          /* Waiting period between queries  */
 USHORT     sBatchSep,          /* Batch separation (NO, OPT1,OPT2, OPT3) */
 USHORT     sInteractive,       /* If it is set, allow the messages to be
                                   retrieved */
 USHORT     sBlock,             /* Block value (Bisync only) */
 USHORT     sCompress,          /* Compress white spaces(Bisync only) */
 USHORT     sTransparent,       /* Make batch transparent(Bisync only) */
 char       *szDaemonName,      /* The daemon name (Max of 8), specified when a
                                   Master server is started with a -N option. */
 char       *szPortName,        /* The specific ports of a resource
                                   (ex. /dev/tty1 or port0) */
 USHORT     sType,              /* Specify data type */
 USHORT     sTrunc,             /* Truncate trailing blanks(Bisync only)  */
 USHORT     sConv               /* Indicates whether to autoconvert the data on
                                   transmission */
 char       *bpid,              /* overrides the business process name that the
                                   autoconnect daemon passes to the GIS adapter.*/
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise.
ulApiCmd	long value that contains the command to be executed.
cbGetInteractMsgs	user-defined callback function that retrieves messages from the Connect:Enterprise autoconnect subsystem.
szAcdList	ACD name (1–8 characters).
szMailboxId	mailbox ID (1–8 characters) of the selected batches. Connect:Enterprise searches the repository for all batches that match the submitted mailbox ID. Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.

Argument	Description
szBatchId	user batch ID (1–64 characters) of the deleted batches. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks. Wild card specifications (like an asterisk, *) are also allowed. A specific batch number is preceded by a pound sign, such as #14. One or more hyphenated ranges of batch ID numbers can be specified after the pound sign, separated by commas (for example, #57–59,88,95,100–110,128).
sACMode	autoconnect mode (valid values are SendRecv, RecvSend, SendOnly, and RecvOnly).
sOneBatch	set the OneBatch option on during a **$$REQ** operation.
sRetry	override the retry parameter.
sInterval	waiting period between queries.
sBatchSep	batch separation (valid values are NO, OPT1, OPT2, and OPT3).
sInteractive	(optional) enables the program to retrieve messages.
sBlock	(optional, Bisync only) forces Connect:Enterprise to send multiple records in a single record data block, separated by record separators. This applies to EBCDIC data only.
sCompress	(optional, Bisync only) specifies that Bisync blank compression is performed on the data before transmission.
sTransparent	(optional, Bisync only) specifies that data is transparent.
szDaemonName	daemon name (1–8 characters), specified when a Master server is started with the -N option. If the -N option is not used, default names are associated with each daemon. The default names for daemons are: ◆ Async–asyncd ◆ Bisync–bisyncd ◆ FTP–ftpd
szPortName	specific ports of a resource (ex. /dev/tty1 or port0).
sType	specifies the type of data being sent or received (ASCII, EBCDIC, or Binary).
sTrunc	(optional, Bisync only) specifies that Connect:Enterprise can truncate trailing blanks before records are transmitted.
sConv	indicates whether to autoconvert the data on transmission (NO_CONVERT, TO_ASCII, TO_EBCDIC). See *$CMUHOME/src/samples.h* for definitions.
bpid	overrides the business process that the auto connect daemon passes to the GIS adapter.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampconnect.c in your *$CMUHOME/src* directory to see an example of the **APICMD_CONNECT** command.

APICMD_DELETE

Use the **APICMD_DELETE** argument in the **CMUAPI_Command** API call to delete batches from the local Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Select batches by specifying values for **szMailboxId**, **szBatchId**, **szFrom**, and **szTo** parameters. Batch information from Mailbox Engine transfers to the application through the **cbPutBatchInfo()** callback function. However, if the **chRecvConfirm** is not set to zero (0x00) then the batch information will not be available.

Function Definition

The function of **APICMD_DELETE** follows:

```
int CMUAPI_command(
    APISESSION  *ApiSessionHandle,    /* Session Handle  */
    ULONG     ulApiCmd,               /* Must be equal to
                                         APICMD_DELETE  */
   CALLBACK   cbPutBatchInfo,         /* a user-defined callback
                                         function that API call passes
                                         batch information as an
                                         Argument  */
    void      *pPutBatchInfoArg,      /* a void pointer to
                                         user-defined structure that
                                         will be passed to the above
                                         callback function as an
                                         argument   */
    ULONG     *ulTotal,               /* Total batches deleted */
    char      *szMailboxId,           /* Mailbox ID (Max of 8) of the
                                         deleted batches */
    char      *szBatchId,             /* User Batch ID (Max of 64) of the deleted
                                         batches */
    char      *szFrom,                /* Start Time/Date range */
    char      *szTo,                  /* End Time/Date range */
    char      chRecvConfirm,          /* Receive confirmation for
                                         each deleted batch
                                         (BatchInfo)   */
    ULONG     lUserFlags,             /* Reserved (Unused) */
    char      *szOrig,                /* Originating id    */
    char      *szFlags,               /* Batch flags       */
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or local Connect:Enterprise system.
ulApiCmd	long value containing the command to be executed.

cbPutBatchInfo	user-defined callback function to which the API call passes batch information as an argument. This function is called for each deleted batch. It follows this syntax:
	`int cbPutBatchInfo(MBOXBATCH_INFO_T *pBatchInfo,`
	`void *pPutBatchInfoArg);`
	`MBOXBATCH_INFO_T *pBatchInfo, /* A pointer to the batch info */`
	`void *pPutBatchInfoArg /* A pointer to a user-defined structure */`

pPutBatchInfoArg	void pointer indicating a user-defined structure that is passed to the above callback function as an argument.
ulTotal	address of a long value containing the total number of batches deleted.
szMailboxId	mailbox ID (1–8 characters) of the deleted batches. Connect:Enterprise searches the repository for all batches that match the submitted mailbox ID. Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.
szBatchId	user batch ID (1–64 characters) of the deleted batches. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks. Wild card specifications (like an asterisk, *) are also allowed. A specific batch number is preceded by a pound sign, such as #14. One or more hyphenated ranges of batch ID numbers can be specified after the pound sign, separated by commas (for example, #57–59,88,95,100–110,128).
szFrom	start time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or after the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or after the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or after the date [CC]yymmdd, but on or after the time hhmm each day ◆ nnn–on or after the date nnn days ago ◆ nnn:hhmm–on or after the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or after the date nnn days ago, but on or after the time hhmm each day ◆ hhmm–on or after the time hhmm today

Argument	Description
szTo	end time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or before the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or before the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or before the date [CC]yymmdd, but on or before the time hhmm each day ◆ nnn–on or before the date nnn days ago ◆ nnn:hhmm–on or before the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or before the date nnn days ago, but on or before the time hhmm each day ◆ hhmm–on or before the time hhmm today
chRecvConfirm	receive confirmation for each deleted batch (BatchInfo). If this is set to (0x00) then BatchInfo structure will not be received as a confirmation of the Delete operation.
lUserFlags	reserved (unused). Must be set to zero.
szOrig	originating ID of the deleted batches. Null selects all originating IDs.
szFlags	selects batches with specified flags. Null allows any flag to match.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampdelete.c in your *$CMUHOME/src* directory to see an example of the **APICMD_DELETE** command.

APICMD_ERASE

Use the **APICMD_ERASE** argument in the **CMUAPI_Command** API call to erase batches from the host Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Select batches by specifying values for **szMailboxId**, **szBatchId**, **chOR_AND**, **szFrom**, and **szTo** parameters. Batch information from the Mailbox Engine transfers to the application through the **cbPutBatchInfo()** callback function. However, if the **chRecvConfirm** is not set (0x00) then the batch information will not be available.

Function Definition

The function of **APICMD_ERASE** follows:

```
int CMUAPI_command(
  APISESSION  *ApiSessionHandle, /* Session Handle  */
  ULONG    ulApiCmd,             /* Must be set to APICMD_ERASE  */
  CALLBACK cbPutBatchInfo,       /* A user-defined callback function
                                    that API call passes batch
                                    information as an Argument */
  void     *pPutBatchInfoArg,    /* Void pointer to a user-defined
                                    structure that passes to the
                                    callback function as an
                                    argument */
  ULONG    *ulTotal,             /* Total batches Erased */
  char     *szMailboxId,         /* Mailbox ID (Max of 8) of
                                    the Erased batches */
  char     *szBatchId,           /* User Batch ID (Max of 64
                                    of the Erased batches */
  char     *szFrom,              /* Start Time/Date range */
  char     *szTo,                /* End Time/Date range */
  ULONG    lProcessFlag,         /* Processing Flags Bitmap */
  char     chOR_AND,             /* If logical OR or AND
                                    should be  performed on the
                                    Processing Flags Bitmap
                                    (0 means AND, 1 means OR) */
  char     chRecvConfirm,        /* Receive confirmation for
                                    each erased batch (BatchInfo) */
  ULONG    lUserFlags            /* Reserved (Unused) */
  char     *szOrig,              /* Originating id */
  char     *szFlags              /* Batch flags */
  )
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value containing the command to be executed.

cbPutBatchInfo	user-defined callback function that the API call passes batch information to as an Argument. This function is called for each erased batch. The syntax is as follows:
	`int cbPutBatchInfo(MBOXBATCH_INFO_T *pBatchInfo,`
	`  void *pPutBatchInfoArg);`
	`MBOXBATCH_INFO_T *pBatchInfo, /* A pointer to the batch info */`
	`void *pPutBatchInfoArg /* A pointer to a user-defined structure */`

pPutBatchInfoArg	void pointer to a user-defined structure that is passed to the above callback function as an argument.
ulTotal	address of a long value that contains the total number of batches erased.

Argument	Description
szMailboxId	mailbox ID (1–8 characters) of the erased batches. Connect:Enterprise searches the repository for all batches that match the submitted mailbox ID. Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.
szBatchId	user batch ID (1–64 characters) of the erased batches. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks. Wild card specifications (like an asterisk, *) are also allowed. A specific batch number is preceded by a pound sign, such as #14. One or more hyphenated ranges of batch ID numbers can be specified after the pound sign, separated by commas (for example, #57–59,88,95,100–110,128).
szFrom	start time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or after the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or after the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or after the date [CC]yymmdd, but on or after the time hhmm each day ◆ nnn–on or after the date nnn days ago ◆ nnn:hhmm–on or after the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or after the date nnn days ago, but on or after the time hhmm each day ◆ hhmm–on or after the time hhmm today
szTo	end time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or before the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or before the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or before the date [CC]yymmdd, but on or before the time hhmm each day ◆ nnn–on or before the date nnn days ago ◆ nnn:hhmm–on or before the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or before the date nnn days ago, but on or before the time hhmm each day ◆ hhmm–on or before the time hhmm today
lProcessFlag	processing flags bitmap (such as FLG_MULTXMIT and FLG_XMITONCE). See *$CMUHOME/src/samples.h* for definitions of flags.
chOR_AND	specifies whether a logical OR or AND should be performed on the Processing Flags Bitmap (0 means AND; 1 means OR).
chRecvConfirm	receive confirmation for each erased batch. If this is set to zero (0x00) then the BatchInfo structure will not be received as a confirmation of the erase operation.
lUserFlags	reserved (unused). Must be set to zero.

Argument	Description
szOrig	originating ID of the deleted batches. Null selects all originating IDs.
szFlags	selects batches with specified flags. Null allows any flag to match.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called samperase.c in your *$CMUHOME/src* directory to see an example of the **APICMD_ERASE** command.

APICMD_EXTRACT

Use the **APICMD_EXTRACT** argument in the **CMUAPI_Command** API call to extract batches from the host Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network. Select batches by specifying values for **szMailboxId**, **szBatchId**, **lProcessFlags**, **lCommProtocolFlags**, **SpecialOpFlags**, **lDataFormatFlags**, **szFrom**, **szTo**, **szOrig**, and **szFlags**. Data retrieved from the Mailbox Engine is passed to the application through the **cbPutData()** callback function. Batch information from the Mailbox Engine transfers to the application through the **cbPutBatchInfo()** callback function. The **cbPutData()** is required. The **cbPutBatchInfo** callback is optional and can be set to null. The **cbPutBatchInfo()** function is selected after the Mailbox Engine identifies a batch to be extracted and before any batch data has been extracted.

Function Definition

The function of **APICMD_EXTRACT** follows:

```
int CMUAPI_command(
 APISESSION  *ApiSessionHandle,  /* Session Handle  */
 ULONG  ulApiCmd,                /* Must be equal to
                                    APICMD_EXTRACT  */
 CALLBACK cbPutFileData,         /* a user defined callback
                                    function that retrieves data
                                    from the Mailbox Engine */
 void   *pPutFileDataArg,        /* a void pointer to a
                                    user-defined structure to pass
                                    to the callback function */
 CALLBACK  cbPutBatchInfo,       /* a user defined callback
                                    function that API call passes
                                    batch information as an
                                    Argument*/
 void   *pPutBatchInfoArg,       /* a void pointer to a
                                    user-defined structure that will
                                    be passed to the above callback
                                    function as an argument */
 ULONG  *ulTotal,                /* Total batches extracted */
 char   *szMailboxId,            /* Mailbox ID (Max of 8) of the
                                    extracted batches */
 char   *szBatchId,              /* User Batch ID (Max of 64) of
                                    the extracted batches */
 ULONG  lProcessFlag,            /* Processing Flags Bitmap */
 ULONG  lCommProtocolFlag,       /* Communication Protocol flag
                                    Bitmap (FLG_FTP, FLG_ BSC,
                                    FLG_ASYNC) */
 ULONG  lDataFormatFlag,         /* Data Format Flags Bitmap
                                    (ASCII, EBCDIC, BINARY) */
 ULONG  lSpecialOpFlags,         /* Specify a selection criteria
                                    based on the attributes of the
                                    batch (FLG_OPT3, FLG_ONEBATCH
                                    etc.) */
 ULONG  lUserFlags,              /* Reserved (Unused) */
 struct CMUExtractCounters
       *XtractCnts,              /* A structure containing all
                                    totals (batches skipped, etc.)*/
 USHORT sDecrypt,                /* If non-zero, perform
                                    Decryption.*/
 char   *achKey,                 /* User-supplied decryption key */
 char   *szUserRecord,           /* A string that points to a
                                    file where Macro substitution is
                                    specified */
 USHORT sconv,                   /* Indicates whether to
                                    autoconvert the data on
                                    transmission. */
char   *szFrom,                  /* Start time/date range */
char   *szTo,                    /* End time/date range */
char   *szOrig,                  /* Originating ID */
char   *szFlags                  /* Batch flags */
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.

cbPutFileData	user-defined callback function that retrieves data from the Mailbox Engine. *bufsize* can have the following values on entry:
	>0 = bytes of valid data in buffer
	0 = end of current batch data
	-1= end of current mailbox id (if one id was being extracted)
	-2= end of all mailbox id's (if id1,id2,id3 was being extracted)
	
	This function returns the size of the data buffer that was retrieved correctly. Function syntax is as follows:
	`int cbPutFileData(char *buffer, long bufsize, void *pPutFileDataArg);`
	`char *buffer /* A pointer that points to the location of the data */`
	`long bufsize /* The size of the retrieved data*/`
	`void *pPutFileDataArg /* A pointer to a user-defined structure */`

pPutFileDataArg	void pointer indicating a user-defined structure that is passed to the above callback function as an argument. It enables users to pass information to the callback function instead of using global variables.

cbPutBatchInfo	user-defined callback function that the API call passes batch information to as an argument. This callback function is called for each extracted batch.
	`int cbPutBatchInfo(MBOXBATCH_INFO_T *pBatchInfo, void *pPutBatchInfoArg);`
	`MBOXBATCH_INFO_T *pBatchInfo /* A pointer to the batch info */`
	`void *pPutBatchInfoArg /* A pointer to a user-defined structure */`

pPutBatchInfoArg	void pointer indicating a user-defined structure that is passed to the above callback function as an argument.
ulTotal	address of a long value that contains the total number of batches extracted.
szMailboxId	mailbox ID (1–8 characters) of the extracted batches. Connect:Enterprise searches the repository for all batches that match the submitted mailbox ID. Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.

Argument	Description
szBatchId	user batch ID (1–64 characters) of the extracted batches. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks. Wild card specifications (like an asterisk, *) are also allowed. A specific batch number is preceded by a pound sign, such as #14. One or more hyphenated ranges of batch ID numbers can be specified after the pound sign, separated by commas (for example, #57–59,88,95,100–110,128).
lProcessFlag	processing flags bitmap (such as FLG_MULTXMIT and FLG_XMITONCE). See *$CMUHOME/src/samples.h* for definitions of flags.
lCommProtocolFlag	communication protocol flag bitmap (FLG_FTP, FLG_BSC, FLG_ASYNC). See *$CMUHOME/src/samples.h* for definitions of flags.
lDataFormatFlag	data format flags bitmap (such as FLG_ASCII, FLG_EBCDIC, AND FLG_BINARY). See *$CMUHOME/src/samples.h* for definitions of flags.
lSpecialOpFlags	specifies a selection criteria based on the attributes of the batch (such as FLG_OPT3 and FLG_ONEBATCH).
lUserFlags	reserved (unused).
XtractCnts	structure that contains all totals, such as batches skipped or extracted.
sDecrypt	if non-zero, the program performs a decryption.
achKey	user-supplied decryption key.
szUserRecord	string that points to a file where macro substitution is specified.
sconv	indicates whether to autoconvert the data on transmission (NO_CONVERT, TO_ASCII, TO_EBCDIC). See *$CMUHOME/src/samples.h* for definitions.
szOrig	originating ID of the deleted batches. Null selects all originating IDs.
szFlags	selects batches with specified flags. Null allows any flag to match.
szFrom	start time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or after the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or after the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or after the date [CC]yymmdd, but on or after the time hhmm each day ◆ nnn–on or after the date nnn days ago ◆ nnn:hhmm–on or after the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or after the date nnn days ago, but on or after the time hhmm each day ◆ hhmm–on or after the time hhmm today

Argument	Description
szTo	end time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or before the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or before the date and time [CC]yymmdd and hhmm ◆ nnn–on or before the date nnn days ago ◆ nnn:hhmm–on or before the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or before the date nnn days ago, but on or before the time hhmm each day ◆ hhmm–on or before the time hhmm today

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIerrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

It is possible for the APICMD_EXTRACT call to return a CMURC_OK return code, even if no batches were actually extracted from the repository. In this situation, ulTotal and XtractCnts are set to zero. Also, the USHORT usProcessStatusCode in the MBOXBATCH_INFO for each batch matching the extract criteria reflects the processing status of the associated batch. Values include:

```
#define NOT_EXTRACT             0
#define EXTRACT_OK              99
#define BYPASS_BATCH            98
#define FILTER_BATCH            97
#define BYPASS_TRANSPDATA       11
#define BYPASS_INCOMPLETE       12
#define EXTRCT_INCOMPLETE       13
#define BYPASS_DELETED          14
#define EXTRCT_DELETED          15
#define RE_EXTRACTED            18
```

Example Code

Refer to the sample source file called sampextract.c in your *$CMUHOME/src* directory to see an example of the **APICMD_EXTRACT** command.

APICMD_LIST

Use the **APICMD_LIST** argument in the **CMUAPI_Command** API call to list batches from the host Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Select batches by specifying values for **szMailboxId**, **szBatchId**, **szFrom**, and **szTo** parameters. Batch information from the Mailbox Engine transfers to the application through the **cbPutBatchInfo()** callback function. The **cbPutBatchInfo()** is required.

Function Definition

The function of **APICMD_LIST** follows:

```
int CMUAPI_command(
  APISESSION *ApiSessionHandle, /* Session Handle  */
  ULONG   ulApiCmd,             /* Must be equal to APICMD_LIST */
  CALLBACK cbPutBatchInfo,      /* a user defined callback
                                   function that API call passes
                                   batch information as an Argument */
  void    *pPutBatchInfoArg,    /* a void pointer that points
                                   to a user defined structure that
                                   will be passed to the above
                                   callback function as an argument */
  ULONG   *ulTotal,             /* Total batches listed */
  char    *szMailboxId,         /* Mailbox ID (1-8 char) of the
                                   listed batches */
  char    *szBatchId,           /* User Batch ID (1-64 char) of
                                   the listed batches */
  char    *szFrom,              /* Start Time/Date range */
  char    *szTo,                /* End Time/Date range */
  ULONG   lUserFlags            /* Reserved (Unused) */
 )
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.

cbPutBatchInfo	user-defined callback function that the API call passes batch information to as an argument. This function is called for each listed batch. The syntax of this argument is:
	`int cbPutBatchInfo(MBOXBATCH_INFO_T *pBatchInfo,`
	`  void *pPutBatchInfoArg);`
	`MBOXBATCH_INFO_T *pBatchInfo /* A pointer to the batch info */`
	`void *pPutBatchInfoArg /* A pointer to a user-defined structure */`

Argument	Description
pPutBatchInfoArg	void pointer indicating a user-defined structure that is passed to the above callback function as an argument.
ulTotal	address of a long value containing the total number of batches listed.
szMailboxId	mailbox ID (1–8 characters) of the listed batches. Connect:Enterprise searches the repository for all batches that match the submitted mailbox ID. Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.
szBatchId	user batch ID (1–64 characters) of the listed batches. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks. Wild card specifications (like an asterisk, *) are also allowed. A specific batch number is preceded by a pound sign, such as #14. One or more hyphenated ranges of batch ID numbers can be specified after the pound sign, separated by commas (for example, #57–59,88,95,100–110,128).
szFrom	start time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or after the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or after the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or after the date [CC]yymmdd, but on or after the time hhmm each day ◆ nnn–on or after the date nnn days ago ◆ nnn:hhmm–on or after the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or after the date nnn days ago, but on or after the time hhmm each day ◆ hhmm–on or after the time hhmm today
szTo	end time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or before the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or before the date and time [CC]yymmdd and hhmm ◆ nnn–on or before the date nnn days ago ◆ nnn:hhmm–on or before the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or before the date nnn days ago, but on or before the time hhmm each day ◆ hhmm–on or before the time hhmm today
lUserFlags	reserved (unused). Must be set to zero.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIerrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called samplist.c in your *$CMUHOME/src* directory to see an example of the **APICMD_LIST** command.

APICMD_REFRESH

Use the **APICMD_REFRESH** argument in **CMUAPI_Command** API call to refresh the autoconnect subsystem of the Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network by examining all the ACDs for any new calendar information.

Function Definition

The function of **APICMD_REFRESH** follows:

```
int CMUAPI_command(
  APISESSION  *ApiSessionHandle, /* Session Handle  */
  ULONG   ulApiCmd               /* Must be equal to APICMD_REFRESH  */
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value defining a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called samprefresh.c in your *$CMUHOME/src* directory to see an example of the **APICMD_REFRESH** command.

APICMD_DAEMON_REFRESH

Use the **APICMD_DAEMON_REFRESH** argument in the **CMUAPI_Command** API call to refresh an ACD master daemon or an EDIINT service daemon from any Connect:Enterprise system in the TCP/IP network. Use this argument only in programs you intend to execute on the Connect:Enterprise UNIX repository host.

Function Definition

The function of **APICMD_DAEMON_REFRESH** follows:

```
int CMUAPI_command(
    APISESSION *ApiSessionHandle,    /* Session Handle */
    ULONG ulApiCmd,                  /* Must be equal to APICMD_DAEMON_REFRESH */
    char *DaemonName                 /* Name of the daemon to refresh */
)
```

Arguments

Argument	Description
ulApiCmd	long value that contains the command to be executed.
DaemonName	The resource name of a running daemon to refresh. This call should only be made to the ACD master daemon or to an EDIINT service daemon. In the case of an ACD master daemon, the auto connect database is refreshed. In the case of an EDIINT service daemon, the AS2 configuration file is refreshed.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

APICMD_SSLPASS_REFRESH

Use the **APICMD_SSLPASS_REFRESH** argument in the **CMUAPI_Command** API call to refresh the SSL passphrase for transfers involving Secure FTP and AS2 (HTTP and EDIINT). Use the argument from any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_SSLPASS_REFRESH** follows:

```
int CMUAPI_command(
    APISESSION *ApiSessionHandle,    /* Session Handle */
    ULONG ulApiCmd,                  /* Must be equal to APICMD_SSLPASS_REFRESH */
    char *Pass                       /* The SSL passphrase */
    char *DaemonName                 /* Name of the daemon to refresh */
)
```

Arguments

Argument	Description
ulApiCmd	long value that contains the command to be executed.
Pass	The SSL passphrase. This passphrase can by up to 256 bytes plus a null terminator.
DaemonName	The resource name of the daemon that receives the passphrase. If DaemonName is left out, all daemons of the types SVID, AUTH, FTP, HTTP, and EDIINT receive the Pass argument as the passphrase for their SSL server certificate. This can cause some services to fail or malfunction.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

APICMD_SSHPASS_REFRESH

Use the **APICMD_SSLPASS_REFRESH** argument in the **CMUAPI_Command** API call to refresh the passphrase for an SSH protocol daemon. Use the argument from any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_SSLPASS_REFRESH** follows:

```
int CMUAPI_command(
    APISESSION *ApiSessionHandle,   /* Session Handle */
    ULONG ulApiCmd,                 /* Must be equal to APICMD_SSHPASS_REFRESH */
    char *Pass                      /* The SSH passphrase */
    char *DaemonName                /* Name of the daemon to refresh */
)
```

Arguments

Argument	Description
ulApiCmd	long value that contains the command to be executed.
Pass	The SSH passphrase. This passphrase can by up to 256 bytes plus a null terminator.
DaemonName	The resource name of the daemon that receives the passphrase. If DaemonName is left out, all SSH daemons receive the Pass argument as the passphrase for their SSH server certificate. This can cause some services to fail or malfunction.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

APICMD_IDMBPASS_REFRESH

Use the **APICMD_IDMBPASS_REFRESH** argument in the **CMUAPI_Command** API call to refresh the SSL passphrase for transfers involving Secure FTP and AS2 (HTTP and EDIINT). Use the argument from any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_IDMBPASS_REFRESH** follows:

```
int CMUAPI_command(
    APISESSION *ApiSessionHandle,    /* Session Handle */
    ULONG ulApiCmd,                  /* Must be equal to APICMD_IDMBPASS_REFRESH */
    char *Pass                       /* The SSL passphrase */
)
```

Arguments

Argument	Description
ulApiCmd	long value that contains the command to be executed.
Pass	The SSL passphrase. This passphrase can by up to 256 bytes plus a null terminator.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno.

APICMD_SESSION

Use the **APICMD_SESSION** argument in the **CMUAPI_Command** API call to retrieve session information from the host Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Select sessions by specifying values for the sDaemonType. Session information from Connect:Enterprise control transfers to the application through the cbPutSessionInfo() callback function.

Function Definition

The function of **APICMD_SESSION** follows:

```
int CMUAPI_command(
  APISESSION *ApiSessionHandle,     /* Session Handle */
  ULONG   ulApiCmd,                 /* Must be equal to
                                       APICMD_SESSION  */
  CALLBACK  cbPutSessionInfo,       /* a user defined callback
                                       function that API call passes
                                       session information as an
                                       Argument */
  void  *pPutSessionInfoArg,        /* a void pointer that points to
                                       a user defined structure that
                                       will pass to the above callback
                                       function as an argument   */
  int  *iTotal,                     /* Total Sessions retrieved */
  USHORT sDaemonType,               /* Daemon Type D_FTP, D_MAILBOX,
                                       D_BSC, D_ASYNC */
  char   *achDaemonName             /* Specifies the Daemon to use./
 )
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.

cbPutSessionInfo	user-defined callback function that the API call passes batch information to as an argument. This function is called for each Connect:Enterprise session.
	The syntax of this argument is:
	`int cbPutSessionInfo(SESSION_STATS_T *pSessionInfo, void *pPutSessionInfoArg);`
	`SESSION_STATS_T *pSessionInfo, /* The pointer to the Session info */`
	`void *pPutSessionInfoArg /* A pointer to a user-defined structure */`

pPutSessionInfoArg	void pointer to a user-defined structure that is passed to the above callback function as an argument.
iTotal	address of an integer value containing the total number of sessions retrieved.
sDaemonType	short value identifying the daemon type under Connect:Enterprise (Daemon Type D_FTP, D_MAILBOX, D_BSC, D_ASYNC). See *$CMUHOME/src/samples.h* for definitions.
achDaemonName	specifies the daemon to use. “” returns all.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampsession.c in your *$CMUHOME/src* directory to see an example of the **APICMD_SESSION** command.

APICMD_SHUTDOWN

Use the **APICMD_SHUTDOWN** argument in the **CMUAPI_Command** API call to shut down the Connect:Enterprise system or any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_SHUTDOWN** follows:

```
int CMUAPI_command(
 APISESSION  *ApiSessionHandle,    /* Session Handle  */
  ULONG  ulApiCmd,                 /* Must be equal to
                                      APICMD_SHUTDOWN  */
  char   *szUser,                  /* Connect:Enterprise User ID.*/
  char   *szPassword,              /* Connect:Enterprise Password.*/
  USHORT sUrgency                  /* Level of urgency for stopping
                                      this session */
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value defining a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.
szUser	pointer to an 8-character ASCII null-terminated string identifying the Connect:Enterprise User ID of the individual executing the API call. This information, and the **szPassword** parameter, are both passed to Connect:Enterprise and user-supplied security routines for validation and logging.
szPassword	pointer to an 64-character ASCII null-terminated string containing the password for the user specified in the szUser parameter. The length of the string should not exceed 64 bytes because this is the maximum password length supported by Connect:Enterprise.
sUrgency	level of urgency for shutting down the system where zero means immediate.

Return Value

The API returns APIRC_SYSTEM_DOWN(5) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampshutdown.c in your *$CMUHOME/src* directory to see an example of the **APICMD_SHUTDOWN** command.

APICMD_START

Use the **APICMD_START** argument in the **CMUAPI_Command** API call to start a session that was stopped by the **APICMD_STOP** Command in the Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_STAR**T follows:

```
int CMUAPI_command(
   APISESSION  *ApiSessionHandle, /* Session Handle  */
   ULONG  ulApiCmd,               /* Must be equal to APICMD_START */
   ULONG ulSessionId              /* Session ID to start */
 )
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.
ulSessionId	ID for a session to start up.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampstart.c in your *$CMUHOME/src* directory to see an example of the **APICMD_START** command.

APICMD_STATUS

Use the **APICMD_STATUS** argument in the **CMUAPI_Command** API call to update flags associated with batches from the host Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Select batches by specifying values for **szMailboxId**, **szBatchId**, **szFrom**, and **szTo** parameters. Batch information from the Mailbox Engine transfers to the application through the **cbPutBatchInfo()** callback function. However, if the **chRecvConfirm** is not set (0x00) then the batch information will not be available. The selected batches are updated with the values specified in **szNewMailboxId**, **szNewBatchId**, **lProcessFlagsOn**, **lProcessFlagOff**, **lDataFormatOn**, and **lDataFormatOff**.

Function Definition

The function of **APICMD_STATUS** follows:

```
int CMUAPI_command(
      APISESSION  *ApiSessionHandle,   /* Session Handle  */
       ULONG  ulApiCmd,                /* Must be equal to
                                          APICMD_STATUS  */
      CALLBACK  cbPutBatchInfo,        /* a user defined callback
                                          function that API call
                                          passes batch information as
                                          an Argument */
      void  *pPutBatchInfoArg,         /* a void pointer to a
                                          user-defined structure that
                                          will be passed to the above
                                          callback function as an
                                          argument */
       ULONG  *ulTotal,                /* Total batches updated */
       char   *szMailboxId,            /* Mailbox ID (Max of 8) of
                                          the updated batches */
       char   *szBatchId,              /* User Batch ID (Max of 64)
                                          of the updated batches */
       char  *szNewMailboxId,          /* Mailbox ID (Max of 8) of
                                          the new updated batches */
       char  *szNewBatchId,            /* User Batch ID (Max of 64)
                                          of new updated batches */
       char   *szFrom,                 /* Start Time/Date range */
       char   *szTo,                   /* End Time/Date range */
       ULONG  lProcessFlagOn,          /* Processing Flags to be
                                          turned On */
       ULONG  lProcessFlagOff,         /* Processing Flags to be
                                          turned Off */
       ULONG  lDataFormatFlagOn,       /* Data Format Flags to be
                                          turned On (ASCII, EBCDIC,
                                          BINARY) */
       ULONG  lDataFormatFlagOff,      /* Data Format Flags to be
                                          turned Off (ASCII, EBCDIC,
                                          BINARY) */
       char   chRecvConfirm,           /* Receive confirmation for
                                          each updated batch
                                          (BatchInfo) */
       ULONG  lUserFlags,              /* Reserved (Unused) */
       char   *szOrig,                 /* Originating id    */
       char   *szFlags                 /* Batch flags       */
  )
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value that defines a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.

cbPutBatchInfo	user-defined callback function to which the API call passes batch information as an Argument. This function is called for each updated batch. Syntax of this argument is as follows:
	`int cbPutBatchInfo(MBOXBATCH_INFO_T *pBatchInfo,`
	`void *pPutBatchInfoArg);`
	`MBOXBATCH_INFO_T *pBatchInf, /* A pointer to the batch info */`
	`void *pPutBatchInfoArg /* A pointer to a user-defined structure */`

pPutBatchInfoArg	void pointer to a user-defined structure that is passed to the above callback function as an argument.
ulTotal	address of a long value containing the total number of batches updated.
szMailboxId	mailbox ID (1–8 characters) of the updated batches.Wildcard specifications are supported. For more information, refer to the *Connect:Enterprise UNIX Remote User's Guide*.
szBatchId	user batch ID (1–64 characters) of the updated batches. The batch ID can specify either a number for a specific batch or a 1–64 character literal. The string must be enclosed in quotes and can include embedded blanks. Wild card specifications (like an asterisk, *) are also allowed. A specific batch number is preceded by a pound sign, such as #14. One or more hyphenated ranges of batch ID numbers can be specified after the pound sign, separated by commas (for example, #57–59,88,95,100–110,128).
szNewMailboxId	mailbox ID (1–8 characters) of the new updated batches.
szNewBatchId	user batch ID (1–64 characters) of the new updated batches.
szFrom	start time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or after the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or after the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or after the date [CC]yymmdd, but on or after the time hhmm each day ◆ nnn–on or after the date nnn days ago ◆ nnn:hhmm–on or after the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or after the date nnn days ago, but on or after the time hhmm each day ◆ hhmm–on or after the time hhmm today

Argument	Description
szTo	end time/date range specified as an ASCII string with this syntax: [[CC]yymmdd|nnn[:hhmm|/hhmm]]|[hhmm] The following options are available: ◆ [CC]yymmdd–on or before the date [CC]yymmdd ◆ [CC]yymmdd:hhmm–on or before the date and time [CC]yymmdd and hhmm ◆ [CC]yymmdd/hhmm–on or before the date [CC]yymmdd, but on or before the time hhmm each day ◆ nnn–on or before the date nnn days ago ◆ nnn:hhmm–on or before the date and time nnn days ago and hhmm ◆ nnn/hhmm–on or before the date nnn days ago, but on or before the time hhmm each day ◆ hhmm–on or before the time hhmm today
lProcessFlagOn	processing flags to be turned On such as FLG_MULTXMIT and FLG_XMITONCE. See *$CMUHOME/src/samples.h* for definitions of flags.
lProcessFlagOff	processing flags to be turned Off such as FLG_MULTXMIT and FLG_XMITONCE. See *$CMUHOME/src/samples.h* for definitions of flags.
lDataFormatFlagOn	data format flags to be turned On (such as FLG_ASCII, FLG_EBCDIC, AND FLG_BINARY). See *$CMUHOME/src/samples.h* for definitions of flags.
lDataFormatFlagOff	data format flags to be turned Off (such as FLG_ASCII, FLG_EBCDIC, AND FLG_BINARY). See *$CMUHOME/src/samples.h* for definitions of flags.
chRecvConfirm	receive confirmation for each updated batch (BatchInfo). If this is set to (0x00), the BatchInfo structure will not be received as a confirmation of the update operation.
lUserFlags	reserved (unused). Must be set to zero.
szOrig	selects batches with the specified originating ID. NULL selects all originating IDs.
szFlags	selects batches with the specified flags.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampstatus.c in your *$CMUHOME/src* directory to see an example of the **APICMD_STATU**S command.

APICMD_STOP

Use the **APICMD_STOP** argument in the **CMUAPI_Command** API call to stop a session from executing in the Connect:Enterprise system or from any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_STOP** follows:

```
int CMUAPI_command(
   APISESSION  *ApiSessionHandle,   /* Session Handle  */
   ULONG   ulApiCmd,                /* Must be equal to APICMD_STOP */
   ULONG   ulSessionId,             /* Session ID to stop */
   USHORT  sUrgency                 /* Level of urgency for stopping
                                       this session */
)
```

Arguments

Argument	Description
ApiSessionHandle	holds a pointer to the value defining a communication channel to a remote or host Connect:Enterprise system.
ulApiCmd	long value that contains the command to be executed.
ulSessionId	ID for a session to be stopped/terminated. Obtain this ID by using **CMUAPI_Command**() with **ulApiCmd** equal to **APICMD_SESSION**.
sUrgency	level of urgency for stopping this session where zero means immediate.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *File Agent Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampstop.c in your *$CMUHOME/src* directory to see an example of the **APICMD_STOP** command.

APICMD_CEUTRACE

Use the **APICMD_CEUTRACE** argument in the **CMUAPI_Command** API call to dynamically inquire about or change trace settings for the master daemons or the Connect:Enterprise system from any Connect:Enterprise system in the TCP/IP network.

Function Definition

The function of **APICMD_CEUTRACE** follows:

```
int CMUAPI_command(
  APISESSION  *ApiSessionHandle,   /* Session Handle  */

  CALLBACK  cbPutTraceInfo,        /* a user defined callback
  function that API call
                                          passes daemon information as
                                          an Argument */
      void  *pPutTraceInfoArg, /* a void pointer to a
  user-defined structure that
  will be passed to the above
  callback function as an
       argument */
  int    traceState /* Turn trace on or off. 1=On, 2=Off,
  0=No change
  char   *traceFilePrefix /* Specifies the prefix of the tracefile.
  Null to leave unchanged
  int    traceLevel, /* Level of trace, 1-99
  -1= don't change*/
  char   *daemonName, /* Daemons to include
  Null=all daemons */
  char   *accountName, /* Accounts to include. For future use
      Must pass NULL pointer*/
  char   *resourceName, /* Resources to include. For future use
      Must pass NULL pointer*/
  char   *autoconnectName, /* Autoconnects to include. For future use
      Must pass NULL pointer*/
  int    mailboxAlso, /* Includes related mailbox daemon
  activity. For future use
      Must pass NULL pointer*/
  )
```

Arguments

Argument	Description
ApiSessionHandle	Holds a pointer to the value defining a communication channel to a remote or local Connect:Enterprise system.
cbPutTraceInfo	User-defined callback function to which the API call passes daemon information as an Argument. This function is called for each daemon. Trace information is returned only for each master daemon. Child and slaves daemons are not included in the trace.
pPutTraceInfoArg	Void pointer to a user-defined structure that is passed to the above callback function as an argument.
traceState	Specifies the state of the trace. 1=On, 2=Off, 0=No change
traceFilePrefix	Specifies the prefix of the tracefile. A null value indicates no change to the current prefix. This only takes effect if exactly one master daemon is provided in the daemonName argument.
traceLevel	Specifies the level of the trace. When selecting a trace level, consider that debug levels can affect performance. Valid values are 1-99. Use -1 to leave the trace level unchanged.
daemonName	Specifies the daemons to include in the trace.
accountName	Specifies the accounts to include in the trace. This feature is not yet available. It must be included with a null value.
resourceName	Specifies the resource to include in the trace. This feature is not yet available. It must be included with a null value.
autoconnectName	Specifies the autoconnect to include in the trace. This feature is not yet available. It must be included with a null value.
mailboxAlso	Includes related mailbox daemon activity. This feature is not yet available. It must be included with a null value.

Return Value

The API returns APIRC_OK (0) on success. On failure, it returns an APIErrno. The reason for the error is indicated by an externally defined variable, CMUErrno. Values are noted in *Appendix A, Error Messages*, in the *Connect:Enterprise UNIX Installation and Administration Guide*.

Example Code

Refer to the sample source file called sampceutrace.c in your *$CMUHOME/src* directory to see an example of the **APICMD_CEUTRACE** command.

Index

A

B

C

E

F

K

L

M

P

R

S

T

U

X