CTOS Context Manager II

Programming Guide

UNISYS

UNISYS

CTOS® Context Manager II Programming Guide

Release 5.0

Priced Item

November 1993
Printed in US America
4393 4660-000

UNISYS

Product Information Announcement

○ New Release [✓ ● Revision] ○ Update ○ Errata

Title:

CTOS® Context Manager II 5.0 Programming Guide

This Product Information Announcement (PIA) announces the release and availability of the *CTOS Context Manager II Programming Guide,* release 5.0, part number 4393 4660-000.

This Guide documents the new features in the 5.0 release of CTOS Context Manager II. It also contains corrections to the 4.1 release of the *CTOS Context/Window Manager Programming Guide.*

To order additional copies of this document:

- Customers in the United States can call Unisys Direct at 1-800-448-1424
- All other customers can contact their Unisys representative
- Unisys personnel can order through the Electronic Literature Ordering (ELO) system

Please address all technical and documentation comments relative to this release to:

Unisys Corporation
Product Information
5155 Camino Ruiz
Camarillo, CA 93012

Call our feedback line at :	(800) 729-0451
Or, fax your comments to:	(805) 389-4483

Initial Announcement only:
BT082

All Announcements:
BT081

System: CTOS
Release: Context Manager II 5.0 (November 1993)
Part Number: 4393 4660-000

Page Status

Page	Issue
iii	Original
iv	Blank
v through ix	Original
x	Blank
xi	Original
xii	Blank
xiii through xiv	Original
1-1 through 1-2	Original
2-1 through 2-9	Original
2-10	Blank
3-1 through 3-100	Original
A-1 through A-8	Original
B-1 through B-19	Original
B-20	Blank
C-1 through C-2	Original
Glossary-1 through 7	Original
Glossary 8	Blank
Index 1 through 2	Original

Contents

Figures

Tables

About This Guide

This guide contains programming and reference information for CTOS Context Manager II.

Note: *Throughout the rest of this guide, Context Manager II is abbreviated to Context Manager.*

Who Should Use This Guide

To use this guide, you should be familiar with:

- the internals of the CTOS operating system
- configuring and using Context Manager
- the design, code, and functionality of the applications in which you want to include Context Manager API calls

How This Guide Is Arranged

The information in this guide is organized as follows:

- Section 1 provides an overview of Context Manager and its configuration.
- Section 2 explains how Context Manager works with applications that use its API calls.
- Section 3 contains the Context Manager API calls in alphabetical order.

How to Use This Guide

If you are writing Context Manager API calls for the first time, you should read sections 1 and 2 to gain an understanding of how Context Manager works. Then you can go on to section 3 for the particular API functions you need. You may find it helpful to look over the table of contents before you begin.

Reference Material

This guide includes an index as well as three appendixes. Appendix A lists and explains Status Codes and messages (these Status Codes and messages are also listed in the *CTOS Status Codes Reference Manual*). Appendix B provides information on creating your own Context Manager user interface. Appendix C discusses techniques for rebuilding your operating system to allow more contexts (partitions) than standard.

Related Product Information

Table 1 lists related product information you may find helpful.

Table 1. Related Product Information

Type of Information	Document Title
CTOS Executive commands	*CTOS Executive Reference Manual*
	CTOS Executive User's Guide
CTOS internals and reference information, VAM information	*CTOS Operating System Concepts Manual*
	CTOS Programming Guide
	CTOS Procedural Interface Reference Manual
Installing and configuring Context Manager	*CTOS Context Manager Installation and Configuration Guide*
Using Context Manager	*CTOS Context Manager Operations Training Guide*
Information about applications you plan to run under Context Manager	The appropriate application manuals

Section 1
Overview of Context Manager

CTOS Context Manager is an application users or other applications can use to manage multiple applications running concurrently on a workstation. To an application running under it, Context Manager functions as an extension of the operating system.

Context Manager requires the CTOS III 1.1 (or higher) operating system.

Components of Context Manager

Context Manager consists of the following components:

- Context Manager Service (CM Service)

  The CM Service works with CTOS to provide an environment which allows several interactive applications to run simultaneously.

- Context Manager Screen (CM Screen)

  The CM Screen is the default Context Manager user interface. The CM Screen monitors video and keyboard activity.

- CM Editor

  The CM Editor is a configuration utility that enables you to choose the applications to run under Context Manager.

- InterContext Message Service (ICMS)

  ICMS is a system service that allows applications within Context Manager partitions to communicate with each other through messages. ICMS handles the storing, queuing, and dequeuing of these messages.

Table 1-1 lists files associated with Context Manager.

Table 1-1. Context Manager Files

File Name	Description
CmInstall.run	Run file that installs Context Manager
CmInstallMsg.bin	Message file for the installation run file
CmVM.run	Context Manager run file
CmVMMsg.bin	Message file for Context Manager
CmNull.run	Exit run file for contexts under Context Manager
CmScreen.run	Run file for the default user interface, CM Screen
CmScreenMsg.bin	Message file for the default user interface, CM Screen
CmEditor.run	CM Editor run file. You use CM Editor to edit Context Manager configuration files
CmConfigMsg.bin	Message file for the CM Editor
CmConfigFrm.Lib	Library of forms used by CM Editor
CmInvoker.run	Clears the display and creates frames like the Executive
ICMS.run	InterContext Message Service (ICMS) run file
Request.Cm.Sys	Loadable request file that defines CM Service requests
CmConfig.Sys	Default Context Manager configuration file. You can edit this file to include the applications you want to run under Context Manager or you can create other configuration files If the file *[Sys]<Sys>CmConfig.sys* already exists on your hard disk, the software installation procedure does not overwrite it
Cm.user	Sample user file. If you enter the user name **CM** on the SignOn form, this file loads Context Manager to load automatically If the file *[Sys]<Sys>Cm.user* already exists on your hard disk, the software installation procedure does not overwrite it
CmAPI.Lib	Llibrary of procedural interfaces you can link with applications that use Context Manager

Section 2
How Applications Interact Under Context Manager

This section describes how applications interact under Context Manager. It includes the following information:

- video overview
- application modifications under Context Manager
- suspended background applications
- applications you can swap
- communication with Context Manager
- behavior of CTOS products and system services under Context Manager

The information described in this section presumes your familiarity with CTOS. For information regarding the operating system, refer to the *CTOS Operating System Concepts Manual.*

You should also be familiar with each application that you run under Context Manager. Refer to the appropriate application's documentation for information.

Context Manager requires the CTOS III 1.1 (or higher) operating system. From the application programmer's viewpoint, Context Manager acts as an extension of the operating system and takes over some of its functions. It lets applications run concurrently, each using a separate memory partition.

You can write your application program as if it were running on a single-partition operating system outside of the Context Manager software, but the program must conform to the guidelines given in this section.

Video Overview

Each memory partition has an application character map and a system structure called the video pointer map.

The video pointer map is an array of pointers, one for each line of the screen. These pointers always point to the location of the associated line of the application's screen.

The application character map is an array that stores lines of the screen when an application is running in the background.

When you switch from one context (context A) to another context (context B), the following three steps occur:

- The real screen is copied into context A's character map.
- Context B's character map is copied to the real screen.
- The pointers in both video pointer maps are updated to reflect the appropriate position, either on the real screen or in the application character map.

Applications You Must Modify to Run Under Context Manager

Because Context Manager acts as an extension of the operating system, most applications run without modification.

However, you must modify the following types of applications so they can run under Context Manager:

- applications that write directly to the real screen map in memory
- applications containing busy wait loops
- applications that change a value in the low-memory interrupt vector table directly in memory
- applications that position the cursor using the video controller port
- applications that change the exit run file
- graphics applications released prior to CTOS II 3.3
- applications that control the video and keyboard but do not at the same time control all the lines of the screen

Note: *Two applications cannot share communications ports. For example, you cannot run two versions of the Asynchronous Terminal Emulator at the same time since they both use channel A.*

Applications That Write Directly to the Screen Map

The mapping of video lines ensures that the output from write requests goes to the correct place: either to the real screen or to the application character map. The Video Access Method (VAM) and Video Display Management (VDM) do this mapping invisibly to the application. Therefore, applications that use VAM and VDM calls work without modification under Context Manager.

Context Manager controls the mapping of the video lines to ensure that the output from all write requests goes to the correct place: either to the real screen or to the application character map. However, programs that write directly to the screen cannot be detected; the output from their write requests goes directly to the screen whether they own the lines of the screen or not.

To avoid this problem, you should use VAM to write to the display instead of writing directly to the screen.

For information on using VAM and VDM calls to initialize the screen, refer to the *CTOS / Open Programming Practices and Standards.* For information on manipulating characters and character attributes using VAM calls, refer to the *CTOS Operating System Concepts Manual, Volume I.*

Note: *If you are running an application written in Pascal version 4.0 or earlier, you must indicate Y in the [Needs Exec screen?] field when adding this application to the configuration file.*

Busy Wait Loops

When dealing with the keyboard, you should not include busy wait loops in your application code.

The usual way to read keystrokes from the keyboard is to issue a CTOS ReadKbd call. An application using calls to ReadKbdDirect with mode 1 can generate a busy wait. Since mode 1 means that the operating system returns immediately whether a character is available or not, you can write an application to loop, awaiting a character to arrive.

When an application is running in foreground, Context Manager raises that application's priority (that is, it gives it a lower numeric value). However, when an application runs in background, its priority returns to normal. If an application contains a busy loop and is running in foreground, its priority is higher than the priorities of other applications in background. Because the foreground application always runs, background applications do not get processor time, and stop running.

Changing the Values in the Low-Memory Interrupt Vector Table

If a single application is running, it can change any value in the low-memory interrupt vector table without problems. Because more than one application can run under Context Manager, each application must tell the operating system that it wants to change one of the values in this table. To do this, you use either of the following CTOS calls:

- SetIntHandler
- SetTrapHandler

You use SetIntHandler if your application should not be swapped (for example, if the handler that you are providing is a hardware interrupt service routine).

To allow your application to be swapped out, you use SetTrapHandler. For example, you can swap your application if the handler you are providing is a software interrupt service routine; no interrupts of this type can occur while the application is swapped out. (For more information, refer to the *CTOS Operating System Concepts Manual.*)

Positioning the Cursor

Since an application does not know whether it is running in foreground or background, it must not output values to the video controller port to move the cursor. Instead, use the CTOS procedure PosFrameCursor to position the cursor.

Modifying the Exit Run File

When you choose an application to start, Context Manager chains to the run file and sets the exit run file to *CmNull.run*. Therefore, *CmNull.run* runs when the application is complete. *CmNull.run* sends a message to Context Manager asking it to terminate the context. (Refer to Communication with Context Manager in this section.) Upon receiving this message, Context Manager terminates the appropriate context.

If you write an application that changes the exit run file, Context Manager never gets the message that the application ended. In this case, you must provide a way to run *CmNull.run*.

For example, if the Executive sets itself as the exit run file, the Executive reloads only after its application finishes. The command **Exit Executive** runs *CmNull.run*, enabling you return to Context Manager from the Executive.

Graphics Applications

All graphics applications must be relinked with CTOS Graphics Systems 2.1 if they were released prior to CTOS II 3.3.

Context Manager supports concurrent multiple graphics applications. Graphics output does not go to the character map; it goes to the graphics board.

Modifying the Keyboard Translation Table

Context Manager supports applications that modify the system keyboard translation file through standard CTOS procedural calls. The standard keyboard translation file is *NlsKbd.sys*. For further information, refer to the *CTOS Operating System Concepts Manual.*

Full-Screen Applications on VGA Systems

On VGA systems, Context Manager supports only applications that run full-screen and control all the lines of the video display when they are in the foreground.

Applications Suspended in Background

Applications are classified into four groups, depending on how they communicate with the video display. The four groups are listed in table 2-1. If an application writes directly to the video port, or moves the cursor using the video controller port, or performs other such direct manipulation of the video, Context Manager suspends it if it is in the background, and marks it as either tentatively dirty or absolutely dirty.

Table 2-1. Context States

State	Description
Tentatively clean	Contexts start in this state.
Tentatively dirty	Contexts in the tentatively clean state go to this state if they call ResetVideo or ResetVideoGraphics.
Absolutely clean	Contexts go into this state if they call InitVidFrame or NotifyCM (msgType=8).
Absolutely dirty	Contexts go into this state if they call NotifyCM (msgType=9).

Note: The procedure calls in table 2-1 do not contain complete parameter lists.

To mark an application explicitly as dirty (which prevents it from running while in the background), use the CM Editor to add an asterisk (*) to the end of the command name. (Refer to the *CTOS Context Manager Installation and Configuration Guide*).

For example, to mark the Executive as dirty, you press the **Rename (F7)** function key from the CM Editor display and change the name to **Executive***. Thereafter, Context Manager treats the application as if it sent a message identifying itself as absolutely dirty.

You can also use the NotifyCM call to mark an application as explicitly clean or dirty. For information on the NotifyCM call, refer to Communication with Context Manager in this section.

Note: *The detection of dirty applications is not foolproof; unpredictable results (such as multiple cursors and overlapping text from several applications) can occur unless you change applications that are undetectable to Context Manager.*

Applications You Can Swap

You can only swap those applications that have exactly one outstanding request. For applications that have more than one outstanding request, you must change system services to allow swapping. To respond to CTOS swap requests, they must either finish servicing the application request or return the request to the operating system for requeuing when the application swaps back in. Depending on the timing and the ability of the system service to respond to a swap request, an application can be swappable at one time but not another. For information on handling swap requests, refer to the *CTOS Operating System Concepts Manual.*

Certain calls to CTOS, such as SetCommIsr and SetIntHandler, identify an application as one that cannot be swapped. Real-time and communications applications that include such calls are recognized as unswappable. Therefore, Context Manager never tries to swap them to a swap file.

To ensure that your application is not swapped, you can make a call to the CTOS routine SetSwapDisable. (For further information on this routine, refer to the *CTOS Operating System Concepts Manual.*) To mark an application explicitly as non-swappable, use the CM Editor to add an exclamation point (!) to the end of the command(s) you want to mark.

Communication Between Contexts

Applications running under Context Manager can communicate using CTOS Inter-Process Communication (IPC) or Context Manager's ICMS (InterContext Message Service). The IPC facility manages communication between processes either within a partition or between partitions. ICMS manages storage, queuing, and dequeuing of messages between application partitions.

For more information on IPC, refer to the *CTOS Operating System Concepts Manual*; for more information on ICMS, refer to the *CTOS Context Manager Installation and Configuration Guide.*

Behavior of CTOS Products Under Context Manager

Table 2-2 lists Unisys applications that are dirty or not entirely clean with regard to Context Manager. The following terms and definitions define an application's behavior:

- *Clean* indicates no side effects of the application running under Context Manager.
- *Dirty* means that the application accesses video directly, addresses the cursor through a port, changes color or font directly, and causes unpredictable results when run in background.
- *Busy Wait* refers to applications that poll the keyboard or other devices. It causes all other applications to suspend.

Table 2-2. Product Behavior Under Context Manager

Product	Status
Asynchronous Terminal Emulator	Relatively clean
BTE	Dirty (loads its own fonts)
Enhanced BISYNC 3270 Emulator	Dirty (loads its own fonts)
Forms Designer	Dirty
OFIS Writer	Dirty
PC Emulator	Dirty (loads its own fonts)
CTOS/Vpc	Dirty (loads its own fonts)

Section 3
Programming for Context Manager

This section provides programming information on Context Manager features, including information on the following:

- checking for Context Manager installation
- starting and switching contexts
- context handles
- parent/child relationships of contexts
- specification of context attributes
- invoking programs that require video initialization
- placing information in the Context Manager configuration file
- ICMS (the InterContext Message Service)
- procedural interfaces for using Context Manager requests
- procedural interfaces for using ICMS requests

Checking for Context Manager and ICMS Installation

You can use two calls to determine whether Context Manager is installed:

- CMCurrentVersion

  If Context Manager is installed, CMCurrentVersion returns its version and revision levels. If Context Manager is not installed, CMCurrentVersion returns Erc 12099 (Context Manager is not installed).

- NotifyCM (message 0)

  NotifyCM (message 0) returns Erc 0 if Context Manager is installed, and Erc 33 if Context Manager is not installed.

You can use ICMSCurrentVersion to determine whether ICMS is installed. Like CMCurrentVersion, ICMSCurrentVersion returns the version and revision levels of the ICMS. If ICMS is not installed, ICMSCurrentVersion returns Erc 12108 (ICMS is not installed).

Other Context Manager calls can also return errors if Context Manager or ICMS is not installed or if the the proper loadable request file, Request.CM.sys is not loaded at system initialization. If Context Manager or ICMS is not installed, invoking any Context Manager or ICMS call returns status code 33 (Service not available). If the required request code is not present, the call returns status code 31 (No such request code).

Starting and Switching Contexts

A program running in one context can call Context Manager to start a new application in another context. Similarly, a program in one context can call Context Manager to switch ownership of the keyboard and screen to another context.

When a new application is started, any path information not entered in the configuration file or the data sent in the request for that application is inherited from the parent context.

An outstanding request problem can occur when an application other than a system service or the CM user interface calls Context Manager to start a new context or to switch to an existing context. When a context calls Context Manager to start or swap in a context, the caller may need to be swapped out to create a vacant partition for the new context. However, if a context has an outstanding request, it cannot be swapped.

To avoid this, a program sends two separate requests to Context Manager. The first request supplies the information about the new context to start or the context to switch to. Context Manager responds to the first request, and attempts to start or switch to the new context. The second request is sent so that Context Manager can return any error that was generated while starting the new context.

You can ensure that you correctly receive any errors during a context start or switch by following calls to CMStartApplByBlock, CMStartBkgdApplByBlock, CMStartApplByName, CMStartBkgdApplByName, or CMSwitchToExistingContext with a call to CMQueryErc (except if the call originates from a system service or the user interface). CMQueryErc returns any errors that might have occurred after the other calls.

Alternatively, you can use the CMStartAppl, CMStartApplOptions, and CMSwitchContext routines. They make the calls to start or switch contexts, and then call CMQueryErc to check the status code. Using these routines, you can perform a context start or switch by coding a single call.

Context Handles

When an application is first started, Context Manager assigns it a unique identifier known as a context handle. The context handle is needed for all subsequent calls to Context Manager regarding that context.

A context handle differs from a partition handle in that it allows for differentiation of contexts over time.

For example, if you install Context Manager and start application A from the Context Manager display and then finish application A, the display reappears. If you again start application A, the first and second invocations of application A have the same partition handle because they are loaded into the same physical location of memory. However, they have different context handles so that you can tell them apart.

If a program is to switch to another context, the program must supply the context handle of the target context. To send a message using ICMS, a program must also determine the context handle of the receiver. The following are examples of context handle usage:

- The sender is a parent context that wants to send a message to one of its child contexts. The context handle of the child is returned by Context Manager when the parent starts the child.
- The sender is a child context that wants to send a message to its parent. The child can call CMQueryParent to determine the context handle of its parent.
- The sender and receiver are running concurrently. If context A wants to switch to context B, context A issues a GetPartitionHandle using the name of B. Then, using the returned partition handle, it calls TranslatePhtoCh to find the context handle of B.

Note: *If a program is loaded by a call to Chain or Exit, the context handle remains the same. For example, the context handle for all programs started from the Executive in a partition is the same. The context handle changes only after giving an Exit Executive command and reloading a new Executive.*

There are two reserved context handles. The context handle of Context Manager Service is 0, and 0FFFFh represents the null context handle.

Parent/Child Relationships of Contexts

Every context has a parent, either the Context Manager user interface or another context. A parent context is a context that calls upon the Context Manager Service to create, switch, or terminate other contexts, known as child contexts. When you start a new application from a Context Manager user interface, that context's parent is the Context Manager user interface. When a program calls Context Manager to start a new application, the caller is the parent context and the new application is the child.

Dependent and Independent Child Contexts

A child context can be either dependent or independent. A dependent context is a context that depends on its parent for survival. When the parent context is terminated, the dependent child context is also terminated. An independent child context is a context that does not depend on its parent for survival. If a parent context is terminated, the Context Manager user interface becomes the parent of its independent child context.

Three-Tiered Ancestry

The Context Manager Service supports a three-generational relationship (parent, child, grandchild) among contexts under the Context Manager user interface.

- If a parent context terminates, then the dependent children and their dependent children are also terminated. The independent children become children of the Context Manager user interface.

  Note: *If your application cannot or should not function after its parent context terminates, you should ensure that your application is started as a dependent child context.*

- If a child context terminates, the parent context becomes the parent of any independent grandchild context. The parent context becomes the foreground context only if the child context had been the foreground context.

Note: *If your application should function after its parent context terminates, you should ensure that your application is started as an independent child context.*

Child contexts of a Context Manager user interface may be dependent or independent. This status is determined by the original parent at the time the child context is started, and the lBitDependent bit in the context attribute byte tracks this information (Refer to Specification of Context Attributes for more information).

Figure 3-1 is an example of the dependency relationships of three generations of contexts started from a CM user interface. Figure 3-2 shows the results of the termination of the parent context shown in figure 3-1.

Figure 3-1. Three-tiered Ancestry

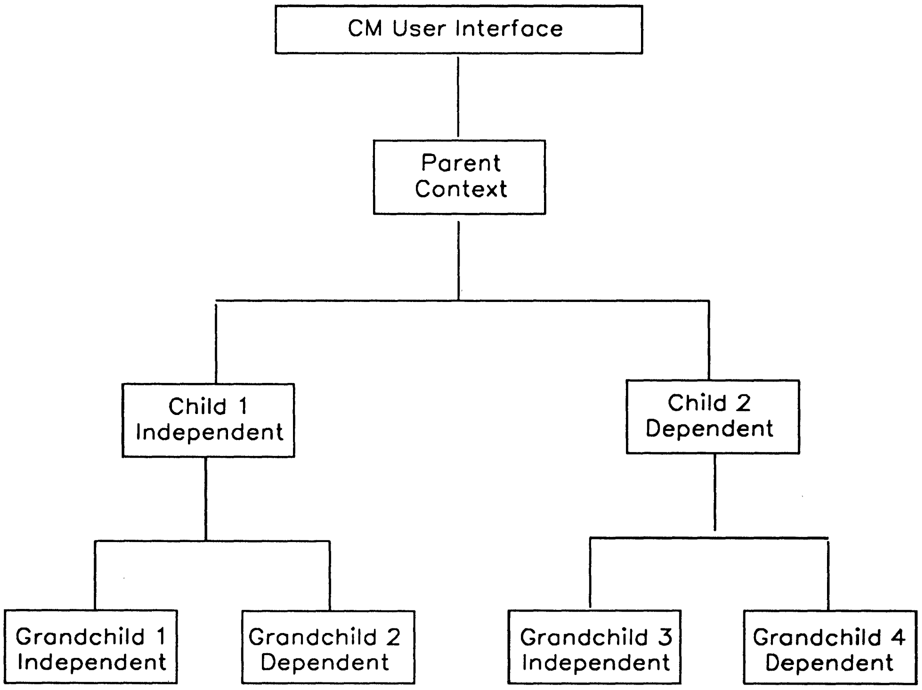
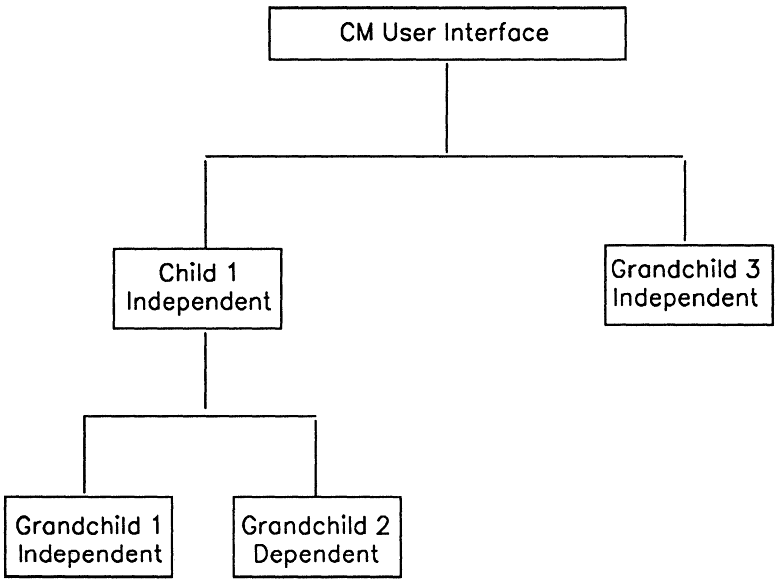


Figure 3-2. Results of Parent Context Termination



When the parent context terminates, independent child context Child 1 becomes the child of the CM User interface, and both the dependent and independent children of Child 1 (Grandchild 1 and Grandchild 2) remain intact.

When the parent context terminates, dependent child context Child 2 also terminates, along with its dependent child context, Grandchild 4. However, the independent context Grandchild 3 survives and becomes a child of the CM user interface.

Context Termination and the CM User Interface

The CM Screen does not read the dependency attribute. When it terminates while acting as the CM user interface, it terminates all contexts regardless of whether they are dependent or independent.

You can create a Context Manager user interface to read the lBitDependent status and treat its child contexts as either dependent or independent with respect to the user interface itself. This would allow all independent contexts to continue even after the Context Manager user interface has been terminated, and become children of the application specified as the exit run file for the CM user interface.

Note: *You must specify an exit runfile other than the default* CMNull.run. CMNull.run *terminates the Context Manager Service, which terminates all contexts regardless of their attributes.*

You can also create a CM user interface that allows you to selectively determine which independent child contexts to continue to execute after the user interface terminates. Refer to Long-lived Contexts for details.

Specification of Context Attributes

Context Manager uses a context attribute byte to specify certain types of child contexts: dependent, long-lived, invisible, or shared video. The Context Manager call CMStartApplOptions allows this attribute byte to be passed as one of the parameters.

This attribute can also be passed using the CMStartApplByName, CMStartApplByBlock, CMStartBkgdApplByName, and CMStartBkgdApplByBlock calls by using offset 13 within the request block. However, to provide compatibility with existing programs using any one of the CMStart calls, the fDependent flag must be set to value 0Ah in order to tell Context Manager Service that the attribute byte being passed is valid. For details on the context attribute byte, refer to CMStartApplOptions.

Long-lived Contexts

Long-lived contexts are independent child contexts that continue to exist even after the Context Manager user interface has chained to its exit run file.

The CM Screen does not recognize long-lived contexts. If CM Screen is the Context Manager user interface and Context Manager terminates, then all child contexts terminate, regardless of whether they are designated as long-lived.

You can create a user interface that recognizes long-lived contexts by reading the long-lived context attribute. This gives the CM user interface the option of allowing selected contexts to continue executing even after the interface itself terminates.

To designate a context as long-lived:

- specify long-lived as a context attribute, using a CMStartAppl call.

  Refer to Specification of Context Attributes for details.

- configure the exit run file to something other than the default exit run file, *CMNull.run*.

  CMNull.run terminates the Context Manager Service, which terminates all contexts regardless of their attributes.

A Context Manager user interface may query for information on long-lived contexts (which may exist prior to running the Context Manager user interface) using the CMQueryOtherContexts call. Refer to appendix B for details.

Invisible Contexts

An invisible context is a context that does not report its existence to the Context Manager user interface. This context type allows a task to run without displaying this activity to the user.

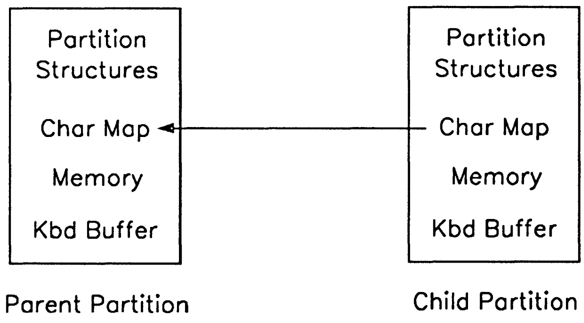
To designate a context as invisible, specify invisible as a context attribute in a CMStartAppl call. Refer to Specification of Context Attributes for details.

Shared Video Contexts

A shared video context is a context that maps its character map to its parent context character map, allowing the child context to direct video output to the screen even while it is running in the background. In this case, the parent and child contexts are responsible for managing video output to the screen to avoid overwriting each other's video output. Figure 3-3 shows the relationship between shared video contexts.

If the parent context terminates, then the Context Manager Service maps the independent child context partition character map back to its own character map.

Figure 3-3. Shared Video Contexts



To designate a context as shared video, specify shared video as a context attribute in a CMStartAppl call. Refer to Specification of Context Attributes for details.

This feature has the following restrictions:

- Neither the parent nor child contexts can use graphics or be considered *video dirty*.
- The parent of a shared video child context must itself be a Context Manager context.
- If a shared video child context calls any CTOS procedure that will initialize video structures, then the screen will blank.
- The parent and child context must be memory-resident.
- The parent and child contexts are responsible for managing screen size and screen attributes to ensure that they are identical between the contexts sharing video.
- Parent-child is the only relationship supported; three-tiered ancestry is not supported

Behavior of Shared Video Contexts

A shared video child context has the following characteristics:

- The child context inherits the color palette and control information from its parent context.
- The child context displays in the same resolution as the parent context.
- If any one of the shared video child siblings or the parent context is the foreground context, they all are considered to be in the foreground since their output is to the real screen.
- If a shared video child context or its parent is not the foreground context, then the parent context and all its shared video children write to the parent's partition character map.
- If the parent of a shared video child context is in the background, the child context cannot be started in the foreground.
- If the parent of a shared video child context terminates or chains, then all independent children will be treated as normal contexts.

Invoking Programs Requiring Video Initialization

Programs written in some programming languages, notably Pascal and C, cannot be invoked directly from Context Manager because they require video initialization that is normally provided when the program is loaded from the Executive. You can invoke these programs by responding **Yes** to the *[Needs Exec screen?]* field in the CM Editor. (For information regarding use of the CM Editor, refer to the *CTOS Context Manager II Installation and Configuration Guide*)

Alternatively, you can write a small program that clears the screen and then chains to your program.

Placing Information in the Context Manager Configuration File

You can place any information in the Context Manager configuration file that your program needs to read when it is running. To do this, you press the **More (F10)** key from the CM Editor, which overlays the Command Editing Area with the More Information Area. (For information regarding use of the CM Editor, refer to the *CTOS Context Manager II Installation and Configuration Guide.*) You provide information using the following syntax: *:Field:Value*.

Context Manager does not use this information unless you use a field that is meaningful to it, such as *:Params:*.

ICMS (InterContext Message Service)

Applications that are running in separate contexts under Context Manager can communicate with each other by passing messages using ICMS (the InterContext Message Service). You use ICMS procedural calls to send and receive messages. Refer to ICMS Procedural Interfaces for details. ICMS is a separate system service that can be installed prior to or during Context Manager installation.

A context communicates with another context by sending a message to ICMS, specifying the context handle of the receiver. Each context is uniquely identified by a context handle that Context Manager assigns. The receiver either waits for a message to arrive or periodically checks for the arrival of a message. This makes ICMS similar to Interprocess Communication (IPC).

ICMS differs from IPC in one important aspect: under IPC, only the address of the message gets transferred. If the sender of the message swaps to disk before the message is received, this procedure may not work. Even though IPC has the address of the message, another context that was swapped into the same physical partition can overwrite the memory space that the message occupies.

ICMS copies the message into the ICMS memory space, where the message waits for the receiver to claim it. Because ICMS copies the actual message, you can write programs and use ICMS to pass messages without worrying about swapping problems.

ICMS queues messages for each context and sends them out on a first-in, first-out basis. When the ICMS memory space for messages is exhausted, it dumps the messages to a disk file.

You use the CM Editor to configure the amount of memory allocated for saving ICMS messages. The CM Editor assigns function key **(F9)** to ICMS. When you press this key, a menu for the ICMS run file and the number and sizes of messages overlays the command editing area. The default is 1 message of 200 bytes.

If you plan to use ICMS extensively, you can install it at system initialization through an entry in the JCL file. Refer to the *CTOS Context Manager II Installation and Configuration Guide* for more information about setting up Context Manager.

ICMS Message Sizing

ICMS message sizes must be a multiple of 128. Because each message includes 10 bytes of header information, the user-specified portion of the message cannot exceed 118 bytes. If the message size is larger than 128 bytes, ICMS rounds it up to the nearest multiple of 128.

ICMS Message Buffer Sizing

The size of the message buffer allocated by ICMS to hold the messages in memory is calculated by the following formula:

Message Buffer Size = Size of Message * Number of Messages

This buffer must be a multiple of 512. If the size of the message is a multiple of 512, then any number of messages is allowed. If the size of the message is a multiple of 256, then the number of messages must be a multiple of 2. Otherwise, the number of messages must be a multiple of 4.

Increasing the number of messages improves performance. However, the message buffer cannot exceed 32KB. When the number of messages sent through ICMS exceeds the size of the message buffer, ICMS expands a file on disk to accomodate additional messages. It writes the current message buffer from memory to the disk and then reads in the newest part of the file from disk to memory. Therefore, if queued messages cause ICMS to access the disk file often, Context Manager performance decreases.

Context Manager Procedural Interfaces

Context Manager provides programmatic interfaces to its internal functions of starting, switching, and terminating contexts. Anything you can do from the Context Manager Screen is available as a procedural call. Together with ICMS, which allows programs in different partitions to communicate, these calls form tools to build a system of integrated programs.

OFIS Designer makes extensive use of the functionality described here. It allows a user to prepare a document that may contain text, graphics, spreadsheets, and/or voice by automatically invoking programs such as OFIS Graphics or OFIS Spreadsheet through the Context Manager Service and passing messages through ICMS.

The following text describes procedural interfaces for the Context Manager operations. For descriptions of other operations, refer to the *CTOS Procedural Interface Reference Manual.*

To write code that uses any of the following procedural interfaces, you must link your run file with the library file *CmAPI.Lib.*

This file resolves references to Context Manager requests and object module procedures within your program. You install this file on your hard disk when you first install the Context Manager software.

CMCurrentVersion

CMCurrentVersion (pbVersion, pbRevision): ErcType

Description

CMCurrentVersion lets a program determine the current version and revision level of Context Manager. It first makes the following checks:

- Context Manager is installed
- Request.CM.sys is installed

Parameters

pbVersion is the memory address of a byte into which Context Manager returns its version level.

pbRevision is the memory address of a byte into which Context Manager returns its revision level.

Errors

12099	Context Manager is not installed.

Request Block

CmCurrentVersion is an object module procedure.

CMCutData

CMCutData (chFrom, nLinestoPaste): ErcType

Description

This procedure is required for Context Manager's data transfer feature. CMCutData allows the Context Manager Service to save the appropriate partition keyboard table and number of lines to paste. The caller then calls CMTransferData.

Parameters

chFrom is the context handle of the context from which the data originates.

nLinestoPaste describes the number of screen lines to be transferred to the receiving context. The maximum number of lines is 38.

Errors

12003	No such context handle.
12022	Paste already in progress.
12023	Invalid data transfer parameters.
12035	Data transfer is not supported from this application.

CMCutData (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	80CFh
12	chFrom	2	
14	nLinestoPaste	2	

CMQueryActiveContext

CMQueryActiveContext (pchRet): ErcType

Description

This call returns the context handle of the active context at the time this request was processed by the Context Manager Service.

Parameters

pchRet is the memory address of a word to which the context handle of the active context is returned.

Errors

None

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntlnfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	65D8h
12	reserved	6	
18	pchRet	4	
22	schRet	2	2

CMQueryAutoStartAppl

CMQueryAutoStartAppl (pInfo, sInfo, pAutoStartInfo, sAutoStartInfo): ErcType

Description

This call returns to the calling application:

- a block of information about the array of autostart information
- a list of applications to be autostarted

Parameters

pInfo is the memory address of a structure returned by the Context Manager Service that contains information about the AutoStartInfo array.

sInfo is a word value that contains the size of the Info structure.

pAutoStartInfo is the memory address of an array of data structures to which autostart info is returned.

sAutoStartInfo is a word containing the size of the array to be returned.

Errors

12027	Not a registered CM user interface.

CMQueryAutoStartAppl (continued)

Info Structure Definition

Offset	Field	Size
0	nAutoStart	2
2	sAutoStart	2

where:

nAutoStart is the number of applications listed in the AutoStartInfo array.

sAutoStart is the size of each autostart record returned in the AutoStartInfo array.

AutoStartInfo Array Definition

AutoStartInfo is an array of structures in which each structure contains the following:

Offset	Field	Size
0	sbCommandName	80
80	sbRunFile	80

where:

sbCommandName is an array of bytes in which the first byte contains the size of the context command name and the remaining bytes contain the command name.

sbRunFile contains the fully qualified file specification to load the run file. The first byte contains the size of the file specification in bytes.

CMQueryAutoStartAppl (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	666Ch
12	reserved	6	
18	pInfo	4	
22	sInfo	2	
24	pAutoStartInfo	4	
28	sAutoStartInfo	2	

CMQueryConfigFile

CMQueryConfigFile (pbBuf, cbBuf, pcbBufRet): ErcType

Description

The CMQueryConfigFile call returns the full file specification of the current Context Manager configuration file.

Parameters

pbBuf and **cbBuf** describe the buffer where the name of the configuration file is returned.

pcbBufRet is the memory address of a word value where the actual size of the configuration file name is placed.

Errors

12002	Configuration buffer too small.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8077h
12	reserved	6	
18	pbBuf	4	
22	cbBuf	2	
24	pbBufRet	4	
28	cbBufRet	2	2

CMQueryContextHandle

CMQueryContextHandle (pchRet): ErcType

Description

The CMQueryContextHandle call lets a program determine its own context handle. This program can then identify itself to another program by including its context handle in a message.

Parameters

pchRet is the memory address of a word value where the context handle returns.

Errors

12001	No such partition handle.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8076h
12	reserved	6	
18	pchRet	4	
22	schRet	2	2

CMQueryErc

CMQueryErc (pErcRet): ErcType

Description

A program uses the CMQueryErc call to retrieve the status code generated during processing by one of the following:

- CMStartApplByBlock
- CMStartApplByName
- CMStartBkgdApplByBlock
- CMStartBkgdApplByName
- CMSwitchToExistingContext

A call to one of these should be followed by a call to CMQueryErc.

A system service or application not started through the CM Service should not call CMQueryErc.

Parameters

pErcRet is the memory address of a word value where the last error code is returned.

Errors

None

CMQueryErc (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	807Ch
12	reserved	6	
18	pErcRet	4	
22	sErcRet	2	2

CMQueryIfGraphics

CMQueryIfGraphics (ch, pfGraphics): ErcType

Description

This procedure allows an application to determine whether a context has registered itself as a graphics context.

Parameters

ch is the context handle of the context.

pfGraphics is the memory address of a flag to which the graphics information is returned. A value of TRUE indicates that the context has registered itself to as a graphics context.

Errors

12003	No such context handle.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	80CEh
12	ch	2	
14	reserved	4	
18	pfGraphics	4	
22	sfGraphics	2	1

CMQueryOtherContexts

```
CMQueryOtherContexts (code, pHeaderBlock, sHeaderBlock,
                      pExistingContexts, sExistingContexts):
                      ErcType
```

Description

This call returns to the calling application:

- a header block of information about the array of existing context information (returned in pHeaderBlock)
- a list of existing context handles and information about them (returned in pExistingContexts)

The context information returned in the array will vary depending on the value passed for code. Applications may use this array to obtain information about other contexts running within the CM session.

Parameters

Code is a word value that specifies which type of context information is returned. The following codes are supported:

0	For use by the CM interface application only to obtain information on any long-lived contexts.
1	Return information on all peers of the calling context (those contexts which have the same parent as the caller, and including the caller itself).
2	Return information on all child contexts (those contexts which have the caller as their parent).
3	Return information on all contexts (including the caller, if applicable).

pHeaderBlock is the memory address of a structure provided by the Context Manager Service that contains information about the returned array (see pExistingContexts).

sHeaderBlock is a word value that contains the size of the header block to be returned.

pExistingContexts is the memory address of an array of data structures to which context-specific information is returned.

sExistingContexts is a word containing the size of the array to be returned.

CMQueryOtherContexts (continued)

Header Block Definition

Offset	Field	Size
0	iVersion	2
2	sHeader	2
4	sContextRecord	2
6	nContextRecords	2
8	chCurrent	2

where:

iVersion is a counter supplied by Context Manager for the CM user interface to use in its next call to CMReadContextEvent.

sHeader is the size of the header.

sContextRecord is the size of each record in the ExistingContexts array.

nContextRecord is the number of records in the ExistingContexts array.

chCurrent is the context handle of the currently active context.

CMQueryOtherContexts (continued)

ExistingContexts Array Definition

ExistingContexts is an array of structures in which each structure contains the following:

Offset	Field	Size
0	ch	2
2	chParent	2
4	sbCommandName	80
84	bFKey	1
85	sbAbbrev	7
92	bContextAttrs	1
93	bStatus	1

Note: *CmQueryOtherContexts determines the size of data to return based on the size of the array passed by the caller.*

CMQueryOtherContexts (continued)

ch is a field containing the context handle.

chParent is a field containing the context handle of the parent of the context.

sbCommandName is an array of bytes in which the first byte contains the size of the context command name and the remaining bytes contain the command name.

bFKey is a field that contains one of the following:

0-9	The context's associated function key (0 corresponds to F1)
0FFh	No function key associated with context

sbAbbrev is an array of bytes in which the first byte contains the size of the context function key abbreviation and the remaining bytes contain the function key abbreviation.

bContextAttrs is a byte which describes special attributes associated with child contexts. Any combination of the following types may be specified within the attribute byte:

1h	lBitDependent	A dependent child context is terminated if its parent is terminated.
2h	lBitInvisible	An invisible child context is not visible to the end user from the CM interface unless it is programatically switched to the foreground.
4h	lBitSharedVid	A shared video child context has its character map mapped to its parent context's character map.
8h	lBitLonglived	A long-lived child context may continue to exist even after the CM interface chains to a different runfile.
10h	lBitDebug	A context started with this bit set to 1 will start in the debugger.

CMQueryOtherContexts (continued)

bStatus is a field containing the current status of the context:

0 = Running

1 = Waiting

2 = Done

3 = Locked

4 = Stopped or Halted

Errors

12000	A specified parameter does not exist.
12002	Configuration buffer too small.
12027	Not a registered CM user interface.

CMQueryOtherContexts (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	3
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	6551h
12	code	2	
14	reserved	4	
18	pHeaderBlock	4	
22	sHeaderBlock	2	
24	pExistingContexts	4	
28	sExistingContexts	2	

CMQueryParent

CMQueryParent (ch, pchRet): ErcType

Description

A program uses the CMQueryParent call to find out the context handle of the parent of any context.

Parameters

ch is a word value indicating the context handle of the context whose parent is desired. If ch is 0, the context handle of the calling context is used.

pchRet is the memory address of a word value where the context handle returns. A returned context handle of 1 means that the parent is the Context Manager user interface. If a parent context is terminated for any reason, the parent of the terminated context becomes the parent of the orphaned child context.

Errors

12003	No such context handle.

CMQueryParent (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8078h
12	ch	2	
14	reserved	4	
18	pchRet	4	
22	schRet	2	2

CMReadContextEvent

CMReadContextEvent (iVersion, pEventBlock, sEventBlock): ErcType

Description

This request allows a CM interface to obtain specific information about the Context Manager environment. CM interfaces should be able to handle context events in a timely manner. Only applications which call CMRegisterUIMS can take advantage of this call.

Parameters

iVersion is a word that acts as a counter that the Context Manager user interface should increment each time it receives a response to this request.

The Context Manager Service uses this parameter to determine whether the interface has missed an event. If so, the interface should call CMQueryOtherContexts in order to catch up on the state of the system and to obtain the current value of iVersion. For further information on using iVersion, refer to apppendix B.

pEventBlock is a memory address of a data structure into which information pertaining to a specific event is returned to the caller.

sEventBlock is the size of the data structure, defined by the caller, into which the information is to be returned. The amount of information returned depends on the size of the event block passed by the user.

CMReadContextEvent (continued)

EventBlock Structure Definition

Offset	Field	Size
0	bEvent	1
1	ch	2
3	chParent	2
5	rgData	80
85	bContextAttrs	1
86	chRequestor	2
88	bFkey	1
89	sbAbbrev	7
96	bStatus	1

CMReadContextEvent (continued)

bEvent is the type of Context Manager event. Event types are the following:

Event	Description
1	A programmatic context start has occurred in the foreground. The information returned includes: • the context handle of the new context • the context handle of the parent context • a string containing the context name • a byte specifying any special child context attributes • the context handle of the context which requested the context start • a byte containing the function key of the new context • a string containing the context's abbreviation • a byte containing the status of the new context
2	A programmatic context start has occurred in the background. The information returned includes: • the context handle of the new context • the context handle of the parent context • a string containing the context name • a byte specifying any special child context attributes • the context handle of the context which requested the context start • a byte containing the function key of the new context • a string containing the context's abbreviation • a byte containing the status of the new context
3	A programmatic context switch has occurred. The information returned includes: • the context handle of the new foreground context • the context handle of the context that requested the switch

continued

CMReadContextEvent (continued)

Event	Description
5	The CM Service has just updated the current CM config file. This event is only sent if the CM interface passed fConfigUpdate = TRUE in the CMRegisterUIMS call. The CM interface should re-read the current CM config file to be consistent with the CM Service.
4	A context termination has occurred. The information returned includes: • the context handle of the terminating context • the termination error code
6	A context has been locked from user access. The information returned includes the context handle of the locked context.
7	A context is now unlocked from user access. The information returned includes the context handle of the unlocked context.
8	A context has been stopped. The information returned includes the context handle of the stopped context.
9	A context is now running. The information returned includes the context handle of the running context.
10	A context is now waiting. The information returned includes the context handle of the waiting context.
11	An Executive context now has the status of Done. The information returned includes the context handle of the Executive context.
12	A context has chained (Executive) and the command name is being updated. The information returned includes: • the context handle • a string containing the new command name
13	A context has a new abbreviation (function key label). The information returned includes: • the context handle • a string containing the new abbreviation

continued

CMReadContextEvent (continued)

Event	Description
14	A context termination has occurred. The information returned includes: • the context handle of the terminating context • the termination message contained in rgData
15	A context has a new parent. The information returned includes: • the context handle of the affected context • the context handle of the new parent context
16	An attempt is being made to log out of the system.

ch is a word containing the context handle of the context involved in the event.

chParent is a word containing the context handle of the parent of the context.

rgData is a string containing information pertaining to the context event.

bContextAttr is a byte which describes special attributes associated with child contexts. Any combination of the following types may be specified within the attribute byte:

1h	lBitDependent	A dependent child context will be terminated if its parent is terminated.
2h	lBitInvisible	An invisible child context will not be visible to the end user from the CM interface unless it is programatically switched to the foreground.
4h	lBitSharedVid	A shared video child context will have its character map mapped to its parent context's character map.
8h	lBitLonglived	A long-lived child context may continue to exist even after the CM interface chains to a different runfile.
10h	lBitDebug	A context started with this bit set to 1 will start in the debugger.

CMReadContextEvent (continued)

chRequestor is a word containing the context handle of the context whose request caused the event.

bFKey a byte containing the function key of the child contexts. If there is no function key associated with the child contexts, this value is 0FFh.

sbAbbrev is a string containing the child context's abbreviation (function key label).

bStatus is a byte containing the current status of the context:

0 = Running

1 = Waiting

2 = Done

3 = Locked

4 = Stopped or Halted

Errors

12027	Not a registered CM user interface.
12031	Missed event.

CMReadContextEvent (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	65D7h
12	iVersion	2	
14	reserved	4	
18	pEventBlock	4	
22	sEventBlock	2	

CMRegisterUIMS

CMRegisterUIMS (fConfigUpdate, pHeaderBlock, sHeaderBlock,
pExistingContexts, sExistingContexts,): ErcType

Description

Applications which act as replacements for the CM Screen (CM interfaces) must make this call to coexist properly in a Context Manager environment. Only one CM interface application is allowed within any Context Manager session. The first application to call CMRegisterUIMS becomes the Context Manager user interface.

Parameters

fConfigUpdate is a FLAG which is set to TRUE if the caller wants to be notified if the CM config file is updated. If it is set to something other than TRUE the CM Service will not do real-time updating of the CM config file.

pHeaderBlock is the memory address of a structure provided by the Context Manager Service that contains information about the returned array (see pExistingContexts).

sHeaderBlock is a word value that contains the size of the header block to be returned.

pExistingContexts is the memory address of an array of data structures to which context-specific information is returned.

sExistingContexts is a word containing the size of the array to be returned.

CMRegisterUIMS (continued)

Header Block Definition

Offset	Field	Size
0	iVersion	2
2	sHeader	2
4	sContextRecord	2
6	nContextRecords	2
8	chCurrent	2

where:

iVersion is a counter supplied by Context Manager for the CM user interface to use in its next call to CMReadContextEvent.

sHeader is the size of the header.

sContextRecord is the size of each record in the ExistingContexts array.

nContextRecord is the number of records in the ExistingContexts array.

chCurrent is the context handle of the currently active context.

CMRegisterUIMS (continued)

ExistingContexts Array Definition

ExistingContexts is an array of structures in which each structure contains the following:

Offset	Field	Size
0	ch	2
2	chParent	2
4	sbCommandName	80
84	bFKey	1
85	sbAbbrev	7
92	bContextAttrs	1
93	bStatus	1

Note: *CMRegisterUIMS determines the size of data to return based on the size of the array passed by the caller.*

CMRegisterUIMS (continued)

ch is a field containing the context handle.

chParent is a field containing the context handle of the parent of the context.

sbCommandName is an array of bytes in which the first byte contains the size of the context command name and the remaining bytes contain the command name.

bFKey is a field that contains one of the following:

0-9	The context's associated function key (0 corresponds to F1)
0FFh	No function key associated with context

sbAbbrev is an array of bytes in which the first byte contains the size of the context function key abbreviation and the remaining bytes contain the function key abbreviation.

bContextAttrs is a byte which describes special attributes associated with child contexts. Any combination of the following types may be specified within the attribute byte:

1h	lBitDependent	A dependent child context will be terminated if its parent is terminated.
2h	lBitInvisible	An invisible child context will not be visible to the end user from the CM interface unless it is programmatically switched to the foreground.
4h	lBitSharedVid	A shared video child context will have its character map mapped to its parent context's character map.
8h	lBitLonglived	A long-lived child context may continue to exist even after the CM interface chains to a different runfile.
10h	lBitDebug	A context started with this bit set to 1 will start in the debugger.

CMRegisterUIMS (continued)

bStatus is a field containing the current status of the context:

0 = Running

1 = Waiting

2 = Done

3 = Locked

4 = Stopped or Halted

Errors

12000	A specified parameter does not exist.
12002	Configuration buffer too small.
12026	A user interface for CM has already been registered.
12027	Not a registered CM user interface.

Request Block

This is an object module procedure in cmAPI.Lib.

CMResumeContext

CMResumeContext (ch): ErcType

Description

CMResumeContext allows a parent context to restart one of its child contexts. If the child context is swapped out, its processes resume when it is swapped back into memory. To reactivate a swapped-out child context, calls to CMSwitchContext and CMResumeContext respectively are necessary.

Parameters

ch is the context handle of the child context to be restarted.

Errors

811	Partition is already swapped

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	629Eh
12	ch	2	
14	reserved	4	

CMSetParent

CMSetParent (chToSet, chNewParent): ErcType

Description

A program uses the CMSetParent call to change the parent of a context. CMSetParent does not allow a context to become the parent of its own parent or grandparent contexts.

Parameters

chToSet is the context handle of the context whose parent is to be changed. If chToSet is 0, the context handle of the calling context is used.

chNewParent is the context handle of the new parent context. Note that if 0 is specified for this variable, the new parent is the Context Manager user interface.

Errors

12003	No such context handle.
12012	No such parent context handle.
12013	Cannot change parent to self.
12015	Cannot adopt parent or grandparent context.

CMSetParent (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8082h
12	chToSet	2	
14	chNewParent	2	
16	reserved	2	

CMStartAppl

CMStartAppl (fName, pbStartData, cbStartData, pVLPB,
fDependentChild, pchRet): ErcType

Description

CMStartAppl is called to start a new application. An application is started in two ways: by name or by block. If an application is started by name, the name supplied must exactly match a name in the current Context Manager configuration file. If an application is started by block, a block of information must be supplied to the Context Manager. This block provides the same information normally supplied by the CM configuration file.

If you want to start a context from a system service or application not started through the CM Service, use one of the following calls instead of CMStartAppl:

- CMStartApplByBlock
- CMStartApplByName
- CMStartBkgdApplByBlock
- CMStartBkgdApplByName

CMStartAppl (continued)

Parameters

fName is TRUE to start an application by name, or FALSE to start an application by a block of information.

If fName is TRUE, **pbStartData** and **cbStartData** describe the name of an application that matches one specified in the Context Manager's configuration file. If fName is FALSE, they describe a block whose definition is given in the StartData Block Definition.

pVLPB is the memory address of a variable length parameter block (VLPB) that Context Manager is to use as the new context's VLPB. If pVLPB is non-NIL, then the command case specified in the StartData block is ignored.

fDependentChild is a flag that determines whether the context is dependent on its parent context for survival. If this parameter is set to TRUE, the child context is marked as dependent, and if the parent context is terminated, the child is also terminated.

pchRet is the memory address of a word to which the context handle of the new context is returned.

CMStartAppl (continued)

StartData Block Definition

Offset	Field	Size
0	sbApplName	80
80	sbRunFile	80
160	sbAbbrev	7
167	wMemorySize	2
169	sbCase	3
172	bPresetFKey	1
173	fPresetFKey	1
174	bAutoStart	1
175	sbVolume	13
188	sbDirectory	13
201	sbPrefix	13
214	sbPassword	13
227	sbNode	13
240	fDirty	1
241	fNeedsExecScreen	1
242	defaultColor	2
244	nPixelsWide	2
246	nPixelsHigh	2
248	sbExitRunFileSpec	79
327	sbExitRunFilePswd	13

CMStartAppl (continued)

where:

sbApplName contains the name of the application to start. The first byte contains the size of the name in bytes.

sbRunFile contains the fully qualified file specification to load the run file. The first byte contains the size of the file specification in bytes.

sbAbbrev contains the abbreviation that appears in the Context Manager display for the associated function key. The first byte contains the size of the abbreviation in bytes.

wMemorySize is the amount of memory needed to run this run file. To specify a flexible partition sizing, set the high bit of this word value. To specify all available memory, set this value to 0.

sbCase is the case value to use when invoking the run file.

bPresetFKey designates the function key (if any) to be assigned to the application:

0-9	The context's associated function key (0 corresponds to F1)
0FFh	No function key associated with context

fPresetFKey is a flag that determines whether Context Manager will use the value specified in bPresetFkey. If fPresetFKey is TRUE, Context Manager assigns the specified function key to the application. If fPresetFKey is FALSE, Context Manager ignores bPresetFKey.

CMStartAppl (continued)

bAutoStart is ignored.

sbVolume contains the default volume to use for the new application. The first byte contains the size of the volume specification in bytes.

sbDirectory contains the default directory to use for the new application. The first byte contains the size of the default directory specification in bytes.

sbPrefix contains the default prefix to use for the new application. The first byte contains the size of the prefix specification in bytes.

sbPassword contains the default password to use for the new application. The first byte contains the size of the password in bytes.

sbNode contains the default node to use for the new application. The first byte contains the size of the node name in bytes.

fDirty is a flag that is TRUE if the application should only run when it is in foreground.

fNeedsExecScreen is a flag that is TRUE if the application requires the video to be initialized with frames as though it had been invoked from the Executive.

defaultColor is a value from 0-12 that determines the default color for the application. Color values correspond to the following colors:

Value	Color	Value	Color
0	Green	7	Pink
1	Blue(Cyan)	8	Aqua
2	Amber	9	Magenta
3	Yellow	10	Lavender
4	White	11	Coral
5	Darkblue	12	Red
6	Purple		

CMStartAppl (continued)

nPixelsWide is a word value indicating the width of the screen. Specify 1024 for high resolution, 720 for low resolution.

nPixelsHigh is a word value indicating the height of the screen. Specify 768 for high resolution, 348 for low resolution.

Note: *If an invalid combination of screen width and height is passed, the started application inherits the resolution of the parent context. If the resolution of the parent context is not available, the started application inherits the resolution of the CM Service.*

sbExitRunFileSpec contains the name of the exit run file. The first byte contains the length of the string and the remainder contains the run file name.

sbExitRunFilePswd contains the password of the exit run file. The first byte contains the length of the string and the remainder contains the password.

Note: *Any path information that is not specified is inherited from the parent context.*

CMStartAppl (continued)

Errors

12004	No such command.
12005	Size of memory partition too large.
12007	Missing Command name.
12008	Command name too long.
12009	Missing run file.
12010	Run file name too long.
12011	Missing memory size.

Request Block

CMStartAppl is an object module procedure. Depending on the value of fName, CMStartAppl calls either CMStartApplByName or CMStartApplByBlock, and then calls CMQueryErc to return the proper status code.

CMStartApplByBlock

CMStartApplByBlock (pbBlock, cbBlock, pVLPB, sVLPB,
fDependentChild, pchRet): ErcType

Description

A program uses the CMStartApplByBlock call to request Context Manager to start a new application in another partition.

A call to CMStartApplByBlock should be followed immediately by a call to CMQueryErc, except if the call originates from:

- a system service
- the CM user interface
- an application not started through the CM Service

(You can refer to CMStartAppl for further details).

Parameters

pbBlock and **cbBlock** describe a block of information about the new application to be started. For the definition of the start block, refer to CMStartAppl.

pVLPB and **sVLPB**: describe a variable length parameter block (VLPB). NIL implies that Context Manager should create a VLPB as though the user had selected this application from the Context Manager display. Otherwise, Context Manager copies the VLPB described by pVLPB and sVLPB and uses it as the VLPB for the new application.

fDependentChild is TRUE if the child context is marked as dependent on its parent context for survival. If the parent is terminated, the child is also terminated.

pchRet is the memory address of a word value to which the context handle of the new application is to be returned.

CMStartApplByBlock (continued)

Errors

12005	Size of memory partition too large.
12007	Missing command name.
12008	Command name too long.
12009	Missing run file.
12010	Run file name too long.
12011	Missing memory size.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8079h
12	fDependentChild	1	
13	reserved	5	
18	pCMInfo	4	
22	sCMInfo	2	
24	pVLPB	4	
28	sVLPB	2	
30	pchRet	4	
34	schRet	2	2

CMStartApplByName

CMStartApplByName (pbName, cbName, pVLPB, sVLPB,
fDependentChild, pchRet): ErcType

Description

A program uses the CMStartApplByName call to request Context Manager to start a new application in another partition.

A call to CMStartApplByName should be followed immediately by a call to CMQueryErc, except if the call originates from:

- a system service
- the CM user interface
- an application not started through the CM Service

(You can refer to CMStartAppl for further details).

Parameters

pbName and **cbName** describe the name of the application to be started. The name must match the name of an application in Context Manager's known list of applications supplied in its configuration file.

pVLPB and **sVLPB** describe a variable length parameter block (VLPB). NIL implies that Context Manager should create a VLPB as though the user had selected this application from the Context Manager display. Otherwise, Context Manager copies the VLPB described by pVLPB and sVLPB and uses it as the VLPB for the new application.

fDependentChild is TRUE if the child context is to be marked as dependent on its parent context for survival. If the parent is terminated, the child also is terminated.

pchRet is the memory address of a word value to which the context handle of the new application is to be returned.

CMStartApplByName (continued)

Errors

12004 No such command.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	807Ah
12	fDependentChild	1	
13	reserved	5	
18	pCMInfo	4	
22	sCMInfo	2	
24	pVLPB	4	
28	sVLPB	2	
30	pchRet	4	
34	schRet	2	2

CMStartApplOptions

CMStartApplOptions (fName, fBackground, pbStartData, cbStartData, pVLPB, sVLPB, bContextAttrs, pchRet): ErcType

Description

This call starts any type of child context, either in the foreground or in a background partition. The context attribute values are specified in the bContextAttrs parameter. This procedure calls one of the following procedures, depending on fName and fBackground:

- CMStartApplByName
- CMStartBkgdApplbyName
- CMStartBkgdApplByBlock
- CMStartApplByBlock

CmStartApplOptions then calls CMQueryErc.

A system service or an application not started through the CM Service should not call CMStartApplOptions.

Parameters

fName is a flag set to TRUE to start an application by name, or FALSE to start an application by a block of information.

fBackground is a flag set to TRUE to start an application in a background partition and FALSE to start an application in the foreground.

pbStartData and **cbStartData** vary depending on the value of fName. If fName is TRUE these parameters give the location and size of a string containing the name of the application that matches one specified in the Context Manager configuration file. If fName is FALSE, these parameters give the location and size of the start block used to start a context. For a definition of the start block, refer to CMStartAppl.

pVLPB and **sVLPB** describe a variable length parameter block (VLPB) which is used as the new context's VLPB.

CMStartApplOptions (continued)

bContextAttrs is a byte which describes special attributes associated with child contexts. Any combination of the following types may be specified within the attribute byte:

Context Attributes Definition

1h	lBitDependent	A dependent child context will be terminated when its parent is terminated.
2h	lBitInvisible	An invisible child context will not be visible to the end user from the interface application unless it is programatically switched to the foreground.
4h	lBitSharedVid	A shared video child context will have its character map mapped to its parent context's character map.
8h	lBitLonglived	A long-lived child context may continue to exist even after the CM interface chains to a different run file.
10h	lBitDebug	A context started with this bit set to 1 will start in the debugger

pchRet is the memory address of a word to which the context handle of the new context is returned.

The long-lived and dependent attibutes cannot conflict; a context cannot be both dependent and long-lived.

CMStartApplOptions (continued)

Errors

12004	No such command.
12005	Size of memory partition too large.
12007	Missing command name.
12008	Command name too long.
12009	Missing run file.
12010	Run file name too long.
12011	Missing memory size.

This is an object module procedure in cmAPI.Lib

Note: *Applications that directly call CMStartApplByName, CMStartBkgdApplbyName, CMStartApplByBlock, or CMStartBkgdApplByBlock, and specify context attributes, can pass the bContextAttrs parameter at offset 13 in the appropriate request block, provided that fDependent is set to 0Ah.*

If an application starts a shared video child context without using CMStartApplOptions, the application should use CMStartBkgdApplByName or CMStartBkgdApplByBlock. Otherwise, an erc will be returned if the shared video child context is in the foreground and its parent is in the background.

CMStartBkgdApplByBlock (continued)

CMStartBkgdApplByBlock (pbBlock, cbBlock, pVLPB, sVLPB,
fDependentChild, pchRet): ErcType

Description

A program uses the CMStartBkgdApplByBlock call to ask Context Manager to start a new application in another partition. This partition is a background partition and is not immediately visible on the screen; subsequently switching to this new application causes it to be visible on the real screen.

A call to CMStartBkgdApplByBlock should be followed immediately by a call to CMQueryErc, except if the call originates from:

- a system service
- the CM user interface
- an application not started through the CM Service

(You can refer to CMStartAppl for further details).

Context Manager locks the parent in memory and then unlocks it when the child context has started in the background.

Parameters

pbBlock and **cbBlock** describe a block of information about the new application to be started. For the definition of the start block, refer to CMStartAppl.

pVLPB and **sVLPB** describe a variable length parameter block (VLPB). NIL implies that Context Manager should create a VLPB as though the user had selected this application from the Context Manager display. Otherwise, Context Manager copies the VLPB described by pVLPB and sVLPB and uses it as the VLPB for the new application.

fDependentChild is TRUE if the child context is marked as dependent on its parent context for survival. If the parent is terminated, the child also is terminated.

pchRet is the memory address of a word value to which the context handle of the new application is to be returned.

CMStartBkgdApplByBlock (continued)

Errors

12005	Size of memory partition too large.
12007	Missing Command Name.
12008	Command name too long.
12009	Missing run file.
12010	Run file name too long.
12011	Missing memory size.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	629Bh
12	fDependentChild	1	
13	reserved	5	
18	pCMInfo	4	
22	sCMInfo	2	
24	pVLPB	4	
28	sVLPB	2	
30	pchRet	4	
34	schRet	2	2

CMStartBkgdApplByName

CMStartBkgdApplByName (pbName, cbName, pVLPB, sVLPB,
fDependentChild, pchRet): ErcType

Description

A program uses the CMStartBkgdApplByName call to ask Context Manager to start a new application in another partition. This partition is a background partition and is not immediately visible on the screen; subsequently switching to this new application causes it to be visible on the real screen.

A call to CMStartBkgdApplByName should be followed immediately by a call to CMQueryErc, except if the call originates from:

- a system service
- the CM user interface
- an application not started through the CM Service

(You can refer to CMStartAppl for further details).

Context Manager locks the parent in memory and then unlocks it when the child context has started in the background.

Parameters

pbName and **cbName** describe the name of the application to be started. The name must match the name of an application in Context Manager's known list of applications supplied in its configuration file.

pVLPB and **sVLPB**: pVLPB is either NIL or points to a variable length parameter block (VLPB). NIL implies that Context Manager should create a VLPB as though the user had selected this application from the Context Manager display. Otherwise, Context Manager copies the VLPB described by pVLPB and sVLPB and uses it as the VLPB for the new application.

fDependentChild is TRUE if the child context is marked as dependent on its parent context for survival. If the parent is terminated, the child also is terminated.

pchRet is the memory address of a word value to which the context handle of the new application is to be returned.

CMStartBkgdApplByName (continued)

Errors

12004	No such command

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	629Ch
12	fDependentChild	1	
13	reserved	5	
18	pCMInfo	4	
22	sCMInfo	2	
24	pVLPB	4	
28	sVLPB	2	
30	pchRet	4	
34	schRet	2	2

CMSuspendContext

CMSuspendContext (ch): ErcType

Description

CMSuspendContext allows a parent context to suspend one of its child contexts. To restart the processes of the child context, an application must call CMResumeContext.

If the child context is swapped out, an error is returned.

Parameters

ch is the context handle of the child context to be suspended.

Errors

811	Partition is already swapped
12030	Cannot suspend or resume child

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	629Dh
12	ch	2	
14	reserved	4	

CMSwitchContext

CMSwitchContext (ch): ErcType

Description

CMSwitchContext switches the screen and keyboard to the context specified by the given context handle.

Parameters

ch is the context handle of the context to switch to.

Errors

12003	No such context handle
12057	The swap file is full -- cannot swap any more contexts

Request Block

CMSwitchContext is an object module procedure. CMSwitchContext calls CMSwitchToExistingContext, then calls CMQueryErc to return the proper status code.

CMSwitchToExistingContext

CMSwitchToExistingContext (ch): ErcType

Description

A program uses the CMSwitchToExistingContext call to ask Context Manager to switch the screen and keyboard to an existing context. This call should be immediately followed by a call to CMQueryErc.

Parameters

ch is the context handle of the context to switch to.

Errors

12003	No such context handle.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	807Bh
12	ch	2	
14	reserved	4	

CMTellMeWhenMyChildTerms

CMTellMeWhenMyChildTerms (ch, pExch) ErcType

Description

The CMTellMeWhenMyChildTerms call notifies Context Manager to send back the termination error when the child context finishes.

The parent application must make this call immediately after starting the child context. The parent obtains the child's erc by means of a message sent by Context Manager to the exchange provided.

Context Manager sends the following structure:

ch	WORD	The child context's context handle
erc	WORD	The child context's termination erc

Context Manager sends the above message to the caller using the CTOS function call Send. A four-byte field of information, not a pointer, is sent to the caller's exchange. The low word contains the ch and the high word contains the erc.

If the child context terminates because of operator intervention (for example, because the user pressed **ACTION+FINISH**), Context Manager returns erc 0.

Note: *The NotifyCM procedure message 22 is similar to CMTellMeWhenMyChildTerms. NotifyCM obtains termination information for all children terminated after the NotifyCM call is made.*

Parameters

ch is the child application's context handle.

pExch is the memory address of an exchange where Context Manager sends the termination message.

Errors

None.

CMTellMeWhenMyChildTerms (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	61B9h
12	ch	2	
14	reserved	4	
18	pExch	4	
22	sExch	2	2

CMTerminateContext

CMTerminateContext (ch): ErcType

Description

The CMTerminateContext call lets a program terminate a context in another partition. A program cannot terminate itself or Context Manager in this way.

Parameters

ch is the context handle of the context to be terminated.

Errors

12003	No such context handle.
12006	Cannot kill self.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	807Dh
12	ch	2	
14	reserved	4	

CMTransferData

CMTransferData (chTo, pbBuffer, cbBuffer, mode): ErcType

Description

This procedure allows contexts to paste up to one line of video characters into another context that is reading the keyboard. The Context Manager Service translates the video characters into keystrokes and sends these keystrokes to the receiving context.

Because a different translation table can be associated with each partition, the Context Manager Service handles data transfer as a two step process.

First, so that the video characters can be properly untranslated, the CMCutData function stores the keyboard table associated with the source application (the context where the data is coming from) and number of lines to paste.

Second, after switching to the target (receiving) context, a call to CMTransferData is made so that the Context Manager Service can transfer the translated data into the target partition's keyboard buffer.

Parameters

chTo is the context handle of the context which is to receive the transferred data.

pbBuffer and **cbBuffer** describe the video data which is to be translated and sent to the receiving context. The maximum length of this buffer is 146 characters.

mode specifies the mode which should be used to transfer the data. The five supported modes are:

1 = Line

2 = Word

3 = Block

4 = Multiplan

5 = Spreadsheet

CMTransferData

Errors

811	Partition is swapped
619	No buffer space
12003	No such context handle
12020	This application is not reading the keyboard
12023	Invalid data transfer parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	6334h
12	chTo	2	
14	mode	2	
16	reserved	2	
18	pbBuffer	4	
22	cbBuffer	2	

CMTranslateChToFnKey

CMTranslateChToFnKey (ch,pbFnKeyRet): ErcType

Description

The CMTranslateChToFnKey procedure allows a program to obtain the function key associated with a Context Manager context handle. The caller provides a context handle and the address of a byte where Context Manager returns a value (0-9) representing the function key **(F1-F10)** assigned to that context. If you specify 0 for ch, the procedure returns the calling application's function key. CMTranslateChtoFnKey checks for NULL pointers passed as parameters.

Parameters

ch is the context handle to be translated.

pbFnKeyRet is the memory address of a byte where the translated function key is to be returned.

Errors

12003	No such context handle
12025	No function key for this context

CMTranslateChToFnKey (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C0C0h
12	ch	2	
14	reserved	4	
18	pbFnKeyRet	4	
22	sbFnKeyRet	2	1

CMTranslateChToInfo

CMTranslateChToInfo(ch, pInfoRet): ErcType

Description

This procedure allows an application to query Context Manager for information on other applications. The information returned includes function key information, command name information, and whether the context is in memory.

Parameters

ch is the context handle of the context about which information is to be returned.

pInfoRet is the memory address of a data structure in which context information is returned.

InfoRet Structure Definition

Offset	Field	Size
0	bFkey	1
1	fInMemory	1
2	sbCmdName	80

bFKey is a field containing:

0-9	The context's associated function key (0 corresponds to F1)
0FFh	No function key associated with context

CMTranslateChToInfo (continued)

fInMemory is a flag indicating whether the partition associated with the function key is currently in memory.

sbCmdName is an array of bytes in which the first byte contains the size of the context command name and the remaining bytes contain the command name.

Errors

None

Request Block

This is an object module procedure in cmAPI.Lib.

CMTranslateChToPh

CMTranslateChToPh (ch, pphRet): ErcType

Description

The CMTranslateChToPh allows a program to translate a context handle into a partition handle.

Parameters

ch is the context handle to be translated. A context handle of 0 indicates the CM Service.

pphRet is the memory address of a word value where the translated partition handle is to be returned.

Errors

12003	No such context handle

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	807Eh
12	ch	2	
14	reserved	4	
18	pphRet	4	
22	sphRet	2	2

CMTranslateFnKeyToInfo

CMTranslateFnKeyToInfo (bFKey, pInfoRet): ErcType

Description

The CMTranslateFnKeyToInfo procedure allows a program to query for information about another application running under Context Manager associated with a function key.

Parameters

bFKey is the function key number from 0 to 9 (0 corresponds to **F1**)

pInfoRet is the memory address of a data structure in which context information is returned.

InfoRet Structure Definition

Offset	Field	Size
0	ch	2
2	fSwapped	1
3	sbCommandName	80

ch is the context handle associated with the given function key

fSwapped is a flag indicating if the partition associated with the function key is currently swapped

sbCommandName is the command name of the context associated with the given function key

Errors

12000	A specified parameter does not exist
12003	No such context handle

CMTranslateFnKeyToInfo (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C13Bh
12	bfkey	1	
13	reserved	5	
18	pInfoRet	4	
22	sInfoRet	2	83

CMTranslatePhToCh

CMTranslatePhToCh (ph, pchRet): ErcType

Description

The CMTranslatePhToCh procedure allows a program to translate a partition handle into a context handle.

Parameters

ph is the partition handle to be translated. A partition handle of 0 indicates the CM Service.

pchRet is the memory address of a word value where the translated context handle is to be returned.

Errors

12001	No such partition handle

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	807Fh
12	ph	2	
14	reserved	4	
18	pchRet	4	
22	schRet	2	2

CMUpdateCurrentConfig

CMUpdateCurrentConfig: ErcType

Description

Applications which modify the current Context Manager configuration file can make this call to have the Context Manager Service update the configuration information on a real time basis. The **CM Editor, CM Add Application**, and **CM Remove Application** commands use this function. Modifying the Context Manager configuration file with the Executive Editor does not result in Context Manager automatically reading the new information and updating itself.

Parameters

None.

Errors

None.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	65D6h

Note: *Upon a successful update of the current CM config information, the CM Service will send out a CMReadContextEvent indicating to the CM interface that an update has taken place.*

NotifyCM

NotifyCM (msgType, pbMsg, cbMsg): ErcType

Description

NotifyCM passes information from a context to Context Manager, stating a condition of that context or asking for action by Context Manager.

Parameters

msgType is a word that contains a code for one of the following messages:

Message	Description
0	Test to see if Context Manager is installed. Returns status code 0 (ercOK) if Context Manager is installed and status code 33 if Context Manager is not installed.
1	Terminate this context and pass an erc (sent by CmNull)
2	New command (sent by the Executive)
3	Logout (sent by SignOn)
4	Terminate this context if you wish (sent by the Executive)
5	Change the parent of the calling context to be the Context Manager user interface
6	Graphics application (sent automatically by all graphics applications)
7	Terminate this context and pass an error message (sent by CmNull)
8	This context can run in background (clean) (sent by any application)
9	This context cannot run in background (dirty) (sent by any application)
10	This context is no longer using graphics
11-12	Reserved

continued

NotifyCM (continued)

Parameters

Message	Description
13	Lock this context from user access
14	Unlock this context from user access
15	Terminate this context when its parent terminates
16	Do not terminate this context when its parent terminates
17	Logout unconditionally
18	Change this context's function key abbreviation
19-21	Reserved
22	Notify caller when any child contexts terminate
23-27	Reserved

pbMsg and **cbMsg** describe the appropriate message. These are only meaningful to Context Manager for messages of types 1, 2, 4, 7, 18, and 22.

Values **8** and **9**: Depending on the sequence of calls that a program makes to video routines, Context Manager marks that context as clean or dirty (able or unable to run in background). Any program can override this marking by including an explicit call to NotifyCM with the values 8 or 9.

Value **22**, when you are waiting at an allocated exchange, wait with the address of the message, not a pointer to the address. Use cbMessage to pass the exchange to which notification is to be sent. Refer to TellMeWhenMyChildTerms for a description of the data structure.

Errors

12000 A specified parameter does not exist.

NotifyCM (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	102h
12	msgType	2	
14	reserved	4	
18	pbMsg	4	
22	cbMsg	2	

ReadCMConfigFile

ReadCMConfigFile (prgbFileName, cbFileName, prgCommandInfoRet,
pnCommandsRet, pActionChars, psbInterfaceExitRf):
ErcType

Description

A program uses this procedure to obtain command information within the specified CM configuration file.

Note: *This procedure allocates 1024 bytes of short-lived memory. You should take this into account when sizing your run file. For more information on sizing run files, refer to appendix B.*

Parameters

prgbFileName is the name of the CM configuration file. This is supplied by the calling application.

cbFileName is the length of the CM configuration file name. This is supplied by the calling application.

prgCommandInfoRet is the memory address of an array of structures in which command information will be returned. The commands will be arranged in alphabetical order, unless the file has been changed through the use of the Executive Editor.

pnCommandsRet is the memory address of a word to which the total number of commands in the specified configuration file will be returned.

pActionCharsRet is the memory address of an array which contains the values of the action keys which are to be used to activate the data transfer and the suspend and resume functions. This field can be set to 0 (zero) if the caller does not need this information.

psbInterfaceExitRF is the memory address of an array of bytes containing information on the exit run file for the interface application. The first byte of the array contains the size of the file name and the remaining bytes contain the name of the run file. This field can be set to zero if the caller does not need this information.

ReadCMConfigFile (continued)

rgCommandInfoRet Structure Definition

Offset	Field	Size
0	sbCommandName	80
80	sbAbbrev	7
87	bPresetFKey	1
88	fPresetFKey	1
89	bMode	1

sbCommandName is an array of bytes in which the first byte contains the size of the command name and the remaining bytes contain the command name.

sbAbbrev is an array of bytes in which the first byte contains the size of the command name abbreviation and the remaining bytes contain the command name abbreviation.

bPresetFKey is a field containing:

0-9	The context's associated function key (0 corresponds to F1)
0FFh	No function key associated with context

fPresetFKey is a flag that is set to TRUE if a preset function key value has been passed to bPresetFKey, and FALSE if no value has been passed to bPresetFKey.

bMode is a byte that contains the context's default data transfer mode.

ReadCMConfigFile (continued)

ActionCharsRet Structure Definition

Offset	Field	Size
0	ActionCut	1
1	ActionPaste	1
2	ActionHalt	1

Errors

None.

Request Block

This is an object module procedure in cmAPI.Lib.

ICMS Procedural Interfaces

The following pages describe the procedural interfaces for ICMS operations.

ICMSCheck

ICMSCheck (pbMsg, cbMsg, pcbMsgRet): ErcType

Description

The ICMSCheck procedure allows a program to check for and possibly receive a message sent by another context using ICMS. If a message is queued waiting for this context, the message is returned.

Parameters

pbMsg and **cbMsg** describe a message buffer supplied by the calling program.

pcbMsgRet is the memory address of a word value where the actual size of the message is placed.

Errors

12103	No message available for this context
12106	Message too long

ICMSCheck (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8011h
12	reserved	6	
18	pbMsg	4	
22	cbMsg	2	
24	pcbMsgRet	4	
28	cbMsgRet	2	2

ICMSCurrentVersion

ICMSCurrentVersion (pbVersion, pbRevision): ErcType

Description

The ICMSCurrentVersion procedure allows a program to determine the current version and revision levels of ICMS. It checks to ensure that Request.CM.sys is installed.

Parameters

pbVersion is the memory address of a byte into which ICMS returns the current version.

pbRevision is the memory address of a byte into which the ICMS returns the current revision.

Errors

12108	ICMS is not installed.

Request Block

ICMSCurrentVersion is an object module procedure.

ICMSFlush

ICMSFlush (ch): ErcType

Description

The ICMSFlush procedure allows a program to flush any messages that may be waiting in the ICMS for a given context. This call can be used, for example, when program A in a given partition chains to program B. Program B may initially flush waiting messages intended for A.

Parameters

ch is the context handle of the context whose messages are to be flushed.

Errors

12003	No such context handle

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8012h
12	ch	2	
14	reserved	4	

ICMSSend

ICMSSend (ch, pbMsg, cbMsg): ErcType

Description

The ICMSSend procedure allows a program to send a message to another context.

Parameters

ch is the context handle of the context where the message is to be sent.

pbMsg and **cbMsg** describe the message to be sent.

Errors

12003	No such context handle

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	800Fh
12	ch	2	
14	reserved	4	
18	pbMsg	4	
22	cbMsg	2	

ICMSWait

ICMSWait (pbMsg, cbMsg, pcbMsgRet): ErcType

Description

The ICMSWait procedure allows a program to wait for a message sent by another context.

Parameters

pbMsg and **cbMsg** describe a buffer for the message supplied by the calling program.

pcbMsgRet is the memory address of a word value where the actual size of the message is placed.

Errors

12106	Message too long

ICMSWait (continued)

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	8010h
12	reserved	6	
18	pbMsg	4	
22	cbMsg	2	
24	pcbMsgRet	4	
28	cbMsgRet	2	2

Appendix A
Status Codes and Messages

This section describes status codes and messages for CTOS Context Manager. You can use this information for troubleshooting.

Status Codes

2440	This application must be invoked through the Executive program; edit the configuration file. *Note:* *Status code 2440 has a slightly different meaning for Context Manager than for operating system errors.*
2441	A parameter contains an invalid value.
12000	Unknown message.
12001	No such partition handle.
12002	Configuration buffer too small.
12003	No such context handle.
12004	No such command.
12005	Information block size too large.
12006	A context cannot call CMTerminateContext to terminate itself.
12007	Using start by block, missing command name.
12008	Using start by block, command name too long.
12009	Using start by block, missing run file.
12010	Using start by block, run file name too long.
12011	Using start by block, missing memory size.

12012	No such parent context handle.
12013	Cannot change parent to self.
12014	Cannot switch to locked context.
12015	Cannot adopt parent or grandparent context.
12020	This application is not reading the keyboard. This status indicates that the application selected to receive the pasted data has been suspended or halted. Applications in this state cannot read the keyboard, and therefore cannot receive pasted data.
12021	Cannot paste to a submit program.
12022	Paste is already in progress.
12023	Invalid data transfer parameters.
12024	This context cannot terminate the child context.
12025	No function key for this context.
12026	A user interface for Context Manager has already been registered.
12027	Not a registered Context Manager user interface.
12029	Cannot start shared video child context.
12030	Cannot suspend or resume child context.
12031	The caller has missed an event it should have received from CMReadContext Event. To obtain current context information, the caller should issue a call to CMQueryOtherContexts.
12035	Data transfer is not supported from this application. The user attempted to use the Data Transfer feature from an application which does not have the necessary keyboard tables to support Data Transfer.
12057	The swap file is full -- cannot swap any more contexts.

12084	The run file needed for this application does not exist.
12085	You cannot start any more contexts; maximum is 20.
12086	There is no context to return to.
12087	The swap file is full -- cannot swap any more contexts.
12088	The run file is too large to run in any partition.
12089	You cannot logout; there are contexts which must be finished.
12090	Warning: There are unfinished contexts. Press GO to logout or CANCEL to deny.
12091	This context cannot be finished from the Context Manager.
12092	The file specified as your swap file does not exist.
12093	You can run only one graphics application at a time.
12094	Cannot start an additional video-controlling application.
12095	There is no current context.
12096	An existing context cannot be swapped out to start a new application.
12097	There is not enough room in the swap file to swap the highlighted context.
12099	Context Manager is not installed.
12100	ICMS is already installed.
12101	ICMS internal error.
12102	Cannot open the ICMS Disk Message file.
12103	ICMS: No message available.
12104	ICMS: No free messages.
12105	ICMS: Not implemented.

12106	ICMS: The message sent was too long.
12107	ICMS: This context is already waiting for messages.
12108	ICMS is not installed.
12109	Incorrect version or missing Request.CM.sys.
40001	Context Manager inconsistency, suggest you save all contexts.
40002	Internal error: Wrong exchange.
40003	Internal error: Region status inconsistency.
40004	Internal error: Swap file inconsistency.
40005	Internal error: Too many swaps.
40006	Internal error: Invalid line indices specified for a map switch.
40007	Internal error: Swap count invalid.
40008	Internal error: Unknown context status.

Status Messages

Status messages that may appear during use of Context Manager are described below. The error code, if any, is shown in parentheses.

```
Not enough memory in this partition to run this application.
(Error 400)
```

When you select an application to start and press **GO**, the Context Manager message area says `Loading ...` then `Finishing ...` and then may give this error.

This error occurs when Context Manager tries to start an application in a partition that is too small for the application. The number specified in the *Memory Required* field of the Context Manager configuration file is too small for that application. You edit the Context Manager configuration file to change the entry in the *Memory Required* field for that application.

When you have finished editing the configuration file, save the configuration file changes. If you are using the default Context Manager interface, the configuration is updated automatically.

```
Not enough memory.  (Error 400)
```

This error can occur if you enter a command from the Executive and the partition running the Executive is too small to support the particular program started by the command. For example, you may have entered the **OFIS Mail** command, which starts the CTOS electronic mail application. To avoid this error, instead of increasing the memory allocated for the Executive, you can add OFIS Mail as a separate application to your Context Manager configuration file and start it through Context Manager rather than the Executive.

When you have finished editing the configuration file, save the changes you have made to it. If you are using the default Context Manager interface, the configuration is updated automatically.

```
This version of the OS cannot support any more contexts.
(Error 801)
```

When you try to start a new application from the Context Manager screen, this message may be displayed.

Refer to the *CTOS Context Manager II Programming Guide* for an explanation of how to generate a new version of the operating system that supports more contexts.

```
A context in memory cannot be swapped out.  (Error 813)
```

A context in memory cannot be swapped out because it cannot be quieted; that is, the program has requests outstanding after swapping requests have been issued by the operating system. This error is usually caused by system services that do not handle swapping correctly.

Either wait until the context in memory is finished, or finish the context that is in memory, and then try to switch your context again.

Note: *This message and the next are the same, but have different error codes and slightly different meanings.*

```
A context in memory cannot be swapped out.  (Error 815)
```

A context in memory cannot be swapped out because it has served a request, served interrupts, or is communicating with a serial or parallel port.

Either wait until the context in memory is finished, or finish the context that is in memory, and then try to switch your context again.

```
This application must be invoked through the Exec; edit the
Config File.
```

The application you have chosen requires that you enter a parameter or parameters that should be supplied from the Executive, by means of an Executive command form. Use the following procedure to correct this error:

1. Use the CM Editor to edit your Context Manager configuration file.
2. Display the configuration information for the application.
3. Change the entry in the *Run file* to ***[Sys]<Sys>Exec.run***.
4. Change the entry in the *[Command case]* field to **CM**.
5. Save the configuration file changes.

```
An existing context cannot be swapped out to start a new
application.
```

If you select a new application and press **GO**, the above message may appear. This means that all memory partitions into which the new application would fit are already occupied by contexts that are not allowed to swap out.

Either wait until the context(s) in memory are finished, or finish a context that is in memory, and then try to start your application again.

```
You cannot activate a data transfer session from a graphics
context.
```

If you press the action character configured to start a data transfer session from a context that has registered itself to the Context Manager Service as a graphics context, the above message will appear.

You must select data from a context that does not register itself as a graphics context.

```
The data transfer feature has been disabled.
```

One or both action characters used to transfer data are disabled.

To configure the action key characters, use the CM Editor to edit your Context Manager configuration file. Press **F8** to display the action key character form, and then enter a new character in the character field.

Appendix B
Creating a Context Manager User Interface

You can create a customized user interface for the Context Manager Service as a replacement for CM Screen. This appendix provides information on creating a user interface to the Context Manager Service.

This appendix includes:

- a list of requirements and recommendations for the interface
- a table of procedural calls and their corresponding functions
- information for creating the features equivalent to various existing CM Screen features
- a sample main routine for a Context Manager user interface

Requirements

A CM user interface must:

- be a protected mode compatible V6 run file
- be Video-clean
- be Swap-disabled
- call CMRegisterUIMS
- link with **CmAPI.Lib**

Recommendations

A CM user interface should:

- have a sized run file

  Unisys recommends that a user interface have a sized run file so that it does not use any unnecessary memory.

- run in the 21h - 50h priority range

  A user interface should run in the 21h to 50h priority range so that it receives events in a timely manner without interfering with system services that require a higher priority (for example, the Mouse Service).

- be error-resilient

  A user interface should be error-resilient so that it does not terminate if it receives an error it cannot handle. If the exit run file of the user interface is set to *[Sys]<Sys>CMNull.run* and the user interface exits upon error, the Context Manager Service terminates all contexts and deinstalls itself.

- be responsible for AutoStarting applications

  A user interface should be responsible for autostarting applications. The interface can use the CMQueryAutoStartAppl call to obtain a list of applications marked for autostarting and then use any of the CMStart requests to start them.

- set the exit run file

  A user interface should set the exit run file for itself using the CTOS function SetExitRunFile. For more information, refer to the *CTOS Procedural Interface Reference Manual.*

Initializing a User Interface through the CM Service

When you use the **Install Context Manager** command to install a CM user interface, the Context Manager Service:

- starts the user interface specified in the *[CM Interface Run File Name]* parameter
- creates a partition for the user interface of the size specified in the *[Partition Size (K)]* parameter of fixed size, unless the first character is a less-than sign (<) or, if no size is specified, a flexible partition of the default size (95K).
- names the user interface partition **CMUI**
- initializes the video if the *[Initialize Screen?]* parameter is set to *Yes*. Interfaces written in the C programming language require that this parameter be set.

Creating Context Manager Screen Features

This section contains information on creating features that exist in CM Screen that you may want to include in your user interface. None of these features is required.

Information for creating features equivalent to CM Screen is included for the following features:

- using action keys
- listing available applications
- obtaining function key information
- starting contexts
- switching contexts
- terminating contexts
- obtaining information about autostarted and long-lived contexts
- obtaining information about context starts, switches, and termination
- reporting context status information
- allowing context halting
- using a configured exit run file
- supporting data transfer
- updating the screen
- reporting mail

Table B-1 provides a quick reference to the calls associated with a specified CM Screen feature.

Table B-1. Calls Used to Implement User Interface

Feature	Applicable Calls
Action keys	ReadActionKbd (OS request)
Available applications	CMQueryConfigFile ReadCMConfigFile
Function key information	ReadCMConfigFile CMTranslateChToFnKey CMTranslateChToInfo CMTranslateFnKeyToInfo CMQueryOtherContexts
Starting contexts	CMStartAppl CMStartApplByName CMStartApplByBlock CMStartBkgdApplByName CMStartBkgdApplByBlock CMStartApplOptions
Switching contexts	CMSwitchContext CMSwitchToExistingContext
Terminating contexts	CMTerminateContext
Long-lived contexts	CMRegisterUIMS CMQueryOtherContexts
Contexts to be Autostarted	CMQueryAutoStartAppl
Context starts, switches, and termination	CMQueryOtherContexts CMReadContextEvent CMQueryActiveContext
Context status information	CMQueryOtherContexts CMReadContextEvent CMTranslateChToInfo CMTranslateFnKeyToInfo GetPStructure(OS request)
Context halting	CMSuspendContext CMResumeContext ReadCMConfigFile

continued

Table B-1. Calls Used to Implement User Interface (cont.)

Feature	Applicable Calls
Exit run file	ReadCMConfigFile
Data transfer	CMCutData CMTransferData CMQueryIfGraphics ReadCMConfigFile
Interactive updates	CMRegisterUIMS CMReadContextEvent CMUpdateCurrentConfig ReadCMConfigFile
Mail reporting	MailCheck(OS request)

Using Action Keys

A user can bring the user interface to the foreground by using Action keys. Action keys are used because they are global to all contexts. Normal keystroke and code key combinations cannot be used because the active context receives those keystrokes. Your user interface must issue a ReadActionKbd request in order to use Action keys.

Listing Available Applications

The CM Screen lists the applications available for the user to start in the 'Applications you can start' area of the screen. A user interface can execute the following calls to obtain information on available applications:

- CMQueryConfigFile
- ReadCMConfigFile

CMQueryConfigFile returns the full file specification of the CM configuration file that is currently in use. This configuration file name can then be used in a call to ReadCMConfigFile so that the user interface application can obtain command information on the applications configured in the CM configuration file. This information can then be used by the user interface to display a list of the available applications.

Obtaining Function Key Information

A CM user interface can use three types of function key information:

- the function key number only
- the function key abbreviation
- the preset value, if any, for a function key

CM Screen displays a function key abbreviation on the function key row when an application is started. It displays any preset function keys on the function key row when CM Screen is invoked, and changes them upon update of the configuration file if applicable.

Function Key Number

After a context has been started, CMTranslateChToFnKey can be used to obtain a function key number. If the application has a preset function key, this number is returned. If a function key is preset to 0 or all 10 function keys are in use at the time the context is started, 0FFh is returned, indicating there is no function key for the context.

Function Key Abbreviations

The following calls return function key abbreviations:

- ReadCMConfigFile
- CMTranslateChToInfo
- CMTranslateFnKeyToInfo
- CMQueryOtherContexts

ReadCMConfigFile retrieves function key abbreviations at the same time it obtains application information. The information can be saved and used as contexts are started.

CMTranslateChToInfo, CMTranslateFnKeyToInfo, or CMQueryOtherContexts can be used to obtain function key abbreviations after contexts have been started. CMTranslateChToInfo and CMTranslateFnKeyToInfo return a function key abbreviation for a single context only, based on the context handle or function key passed; CMQueryOtherContexts includes function key abbreviations for all active contexts.

Preset Function Keys

Preset function key information is returned only from the ReadCMConfigFile call. ReadCMConfigFile returns the preset function key information from the specified CM configuration file. The preset key number is returned as a parameter in each Command Info structure. A function key of 0 (no function key) is returned as 0FFh.

Starting Contexts

Before allowing users to start contexts, a user interface must determine what type of child contexts to start. The following areas should be considered:

- foreground or background
- dependent, independent, or long-lived

Contexts can be started in the foreground or in the background. If contexts are started in the foreground, the user may immediately begin work in that context, but must return to the user interface to start another context. If the contexts are started in the background, the user may start several contexts at one time and then choose the context to work with, but this would require an extra step if only one application is started at a given time. All contexts started from CM Screen are started in the foreground.

Contexts can be dependent, independent, or long lived. If contexts are started as dependent children to the user interface, they are terminated by the Context Manager Service when the user exits the interface. If contexts are started as independent or long-lived, the Context Manager Service does not terminate them when the user interface finishes (providing the exit run file for the user interface is not CMNull.run), but the user interface may terminate them itself as part of its exit routine. All applications started from CM Screen are started as dependent children.

Contexts can also be invisible or have shared video capabilities. No events for an invisible context are reported to a user interface. Shared video children share the video with at least one other context. These are not attributes that would be beneficial to assign to all contexts, but a user interface might give the user the ability to choose to start a context with one of these attributes. CM Screen does not allow these attributes to be used for contexts started from the 'Applications you can start' menu area.

The following calls can be used to start contexts:

- CMStartAppl
- CMStartApplByName
- CMStartApplByBlock
- CMStartBkgdApplByName
- CMStartBkgdApplByBlock
- CMStartApplOptions

You can use all of these calls through the Context Manager procedural interfaces. You can also use most of these calls by building the request block. You cannot build request blocks for CMStartAppl and CMStartApplOptions, which are object module procedures.

In procedural call form, only CMStartApplOptions can be used to start contexts with an invisible, shared video, or long-lived attribute. However, in their request forms all calls can start contexts with these attributes by passing the attribute byte at offset 13 in the request block.

Switching Contexts

The CM Screen allows users to switch contexts by pressing **ACTION+NEXT**, **ACTION+Numeric Minus**, **ACTION+FnKey**, or **GO** from CM Screen. Two calls can be used to switch contexts:

- CMSwitchContext
- CMSwitchToExistingContext

Either call will cause the context passed as a parameter to become the foreground context.

Terminating Contexts

The CM Screen allows users to termintate contexts by pressing **ACTION+FINISH** from the CM Screen. You can terminate contexts by using the CMTerminateContext procedure.

Long-Lived Contexts

Long-lived contexts may exist before a user interface is installed. If a user interface wants to adopt these contexts, or allow the user to access them from the user interface, it can use one of the following calls:

- CMRegisterUIMS
- CMQueryOtherContexts

CMRegisterUIMS can be conveniently used when the user interface registers itself with the Context Manager Service. When called, CMRegisterUIMS returns a list of the existing contexts. Alternatively, CMQueryOtherContexts can be used to obtain the same information at any time during program execution, by providing it with a Code value of 0.

CM Screen uses the CMRegisterUIMS method to obtain information about long-lived contexts that exist before it is installed.

Contexts to be Autostarted

The user interface is responsible for starting contexts marked as autostarting. To obtain a list of the contexts to be autostarted, you can use the CM QueryAutoStartAppl call. CM Screen uses this call to obtain a list of contexts to be autostarted, and then starts them in background except for the last one, which it starts in the foreground.

Context Starts, Switches, and Termination

The following three calls allow a user interface to keep correct information on the state of contexts:

- CMQueryOtherContexts
- CMReadContextEvent
- CMQueryActiveContext

CMQueryOtherContexts returns information on all contexts. This information must then be compared to the information obtained from the previous call to CMQueryOtherContexts to determine which contexts are new to the list and which contexts no longer exist in the list. The user interface can then update its display accordingly.

CMReadContextEvent allows an application to keep up-to-date information on the state of contexts. When a request has been made to CMReadContextEvent, the Context Manager Service reports all programmatic starts and switches, as well as other information, so that an interface application can display these contexts on the screen in addition to its own children. CMReadContextEvent reports this information even if the user interface itself caused a change in the state of a context. The Context Manager Service also reports context termination so that a context can be removed from the screen. Error codes or messages are included with the termination event to be used by the interface application if it is to display this information for the user. CM Screen uses this method to display context information.

Note: *Refer to Details on CMReadContextEvent for further information on using this request.*

You can use CMQueryActiveContext at any time to determine the context handle of the active context.

Reporting Context Status Information

A CM user interface can obtain context status information by using the following calls:

- CMQueryOtherContexts
- CMTranslateChToInfo
- CMTranslateFnKeyToInfo
- CMReadContextEvent
- GetPStructure(OS request)

If a user interface is using CMQueryOtherContexts to learn of changes in the context environment, it can use the status that is returned with this call. CMQueryOtherContexts allows a user interface to report the following states:

- Running
- Waiting
- Done
- Locked
- Stopped.

If a user interface is only required to report to the user when a context is in memory, use CMTranslateChToInfo or CMTranslateFnKeyToInfo.

If a user interface is to report locked, unlocked, stopped, running, waiting, or done status information, you can use the CMReadContextEvent request. A new status is reported to the application program at the time that it changes, eliminating the need for the application to repeatedly make a call to find out if the status has changed from the current one.

Note: *Refer to Details on CMReadContextEvent for further information on using this request.*

To obtain information on whether contexts have been swapped, use the operating system call GetPStructure to obtain the pointer to the partition swap status. To update the swap status of a context, check the partition swap array for that partition entry. Note that this array is indexed by partition handles, not context handles.

Allowing Context Halting

To allow users to configure a halt action keystroke, the user interface must use the ReadCMConfigFile call to retrieve this information from the CM configuration file. To support a user configurable suspend keystroke, the user interface can use the pActionCharsRet parameter to request the information configured in the CM configuration file when it calls ReadCMConfigFile.

To support context halting, a user interface need only use two calls:

- CMSuspendContext
- CMResumeContext

If a user interface is to allow the user to suspend a context by performing a specific action (such as the **ACTION+S** keystroke used by the CM Screen), the user interface should call CMSuspendContext. This suspends the context until the user prompts the user interface to resume the context. The user interface can call CMResumeContext at that time to resume normal context operation.

Using a Configured Exit Run File

If a user interface is to allow its exit run file to be configured by the user, it can use the ReadCMConfigFile call to retrieve the interface exit run file information specified in the CM configuration file. The user interface can specify that it wants this information by using the psbInterfaceExitRF parameter. Once the interface exit run file has been obtained, the user interface can then use the SetExitRunFile operating system call to set this file to be its exit run file.

Supporting Data Transfer

If the user interface is to support user configurable keystrokes for cut and paste actions, it can obtain this information when it calls ReadCMConfigFile by using the pActionCharsRet parameter.

If the user interface is to support user configurable data transfer modes, it also needs to call ReadCMConfigFile for this information. This is returned in the Mode parameter of the Command Info structure.

You use the following calls to perform data transfer:

- CMQueryIfGraphics

  Call CMQueryIfGraphics from the source context before a cut operation to ensure that the user is not trying to cut from a graphics context.

- CMCutData

  CMCutData saves the keyboard translation table and passes the number of lines of data to be transferred.

- CMTransferData

  CMTransferData untranslates the characters in the buffer and sends them to the operating system, which forwards them to the receiving application. CMTransferData allows contexts to paste up to one line of video characters into another context that is reading the keyboard. This information can be stored in a buffer so that blocks of text can be transferred at one time.

The CM Screen provides a data transfer (cut/paste) menu to users.

Updating the Screen

Three calls are involved in handling interactive updating of the screen when the user changes the current CM configuration file:

- CMRegisterUIMS
- CMReadContextEvent
- ReadCMConfigFile

First, when the user interface calls CMRegisterUIMS, it must set the fWantsConfigUpdate flag to TRUE to let the CM Service know that it wants to be notified of CM configuration file updates.

After CMRegisterUIMS has been called, the user interface makes the CMReadContextEvent request. When an application, such as CM Editor, calls CMUpdateCurrentConfig, the Context Manager Service returns Event 5 after updating the current CM configuration file information. If the fWantsConfigUpdate flag is not set to TRUE when the user interface calls CMRegisterUIMS, the Context Manager Service does not update the CM configuration file information, and the event is not returned to the caller.

Finally, a user interface has to obtain the updated configuration file information. To do this, ReadCMConfigFile can be called to obtain the new information in the CM configuration file. With this information, the screen can then be updated appropriately.

Reporting Mail

The operating system call MailCheck is used by a user interface to notify users of incoming mail. This call should be made each time the user interface becomes the active context. It also should be timed while the user interface remains the active context and called after a given time interval. CM Screen uses a time interval of 30 seconds. Refer to the CTOS Procedural Interface Reference Manual for more information.

Details on CMReadContextEvent

When using the CMReadContextEvent request, you must increment iVersion each time you get a response from the CMReadContextEvent request. The Context Manager Service uses the iVersion parameter of the CMReadContextEvent request block to keep track of the event in the event queue you last received. When the Context Manager Service receives a CMReadContextEvent request, it compares its internal iVersion with the iVersion you passed. If the discrepancy between the two is greater than 9, an error is returned informing the requestor that events have been missed because Context Manager Service only queues the last 10 events. If you do not increment iVersion each time you get a response from the CMReadContextEvent request, you will soon get out of synch with the Context Manager Service and this error will be returned continually instead of events.

To recover from missed events, you must call CMQueryOtherContexts to obtain the current context information and the contents of iVersion. This information can then be compared to the last context information you received, and you can then update the interface display to the current state. For example, compare the context handles returned from CMQueryOtherContexts to the known context handles to see if any contexts have been started or terminated since your last event.

Sample CM User Interface Program

Figure B-1 contains a simplified example of a main routine for a Context Manager user interface.

Figure B-1. Sample Context Manager User Interface Main Routine

```
void MainLoop(void)
{
    BYTE Key;
    struct rqHeader *prqHeader;
    struct rqReadKbd *prqReadKbd;
    struct rqReadActionKbd *prqReadActionKbd;
    rqCMReadContextEventType *prqReadContextEvent;
    WORD MyExch, MyUserNum;

    /* set up for UIMS and draw screen */
    UIMSSetup();

    /* get user number and exchange */
    GetUserNumber(&MyUserNum);
    AllocExch(&MyExch);

    /* initialize request blocks */
    prqHeader = (struct rqHeader *) calloc(1, sizeof(struct
rqHeader));

    prqReadKbd = (struct rqReadKbd *)
calloc(1, sizeof(struct rqReadKbd));
    prqReadKbd->sCntInfo = 6;
    prqReadKbd->nRespPbCb = 1;
    prqReadKbd->UserNum = MyUserNum;
    prqReadKbd->ExchResp = MyExch;
    prqReadKbd->rqCode = RQREADKBD;
    prqReadKbd->pCharRet = &Key;
    prqReadKbd->sCharRet = 2;

    prqReadActionKbd = (struct rqReadActionKbd *) calloc(1,
                        sizeof(struct rqReadActionKbd));
    prqReadActionKbd->sCntInfo = 6;
    prqReadActionKbd->nRespPbCb = 1;
    prqReadActionKbd->UserNum = MyUserNum;
    prqReadActionKbd->ExchResp = MyExch;
    prqReadActionKbd->rqCode = RQREADACTIONKBD;
    prqReadActionKbd->fGlobal = TRUE;
    prqReadActionKbd->pActionCharRet = &Key;
    prqReadActionKbd->sActionCharRet = 2;
```

Figure B-1. Sample Context Manager User Interface Main Routine (cont.)

```
    prqReadContextEvent = (rqCMReadContextEventType *)
             calloc(1, sizeof(rqCMReadContextEventType));
    prqReadContextEvent->sCntInfo = 6;
    prqReadContextEvent->nRespPbCb = 1;
    prqReadContextEvent->UserNum = MyUserNum;
    prqReadContextEvent->ExchResp = MyExch;
    prqReadContextEvent->rqCode = RQCMREADCONTEXTEVENT;
    prqReadContextEvent->pEventBlock = (EventInfoType *)
             calloc(1, sizeof(EventInfoType));
    prqReadContextEvent->sEventBlock =
             sizeof(EventInfoType);

    /* issue requests */
    CheckErc(Request(prqReadKbd));
    CheckErc(Request(prqReadActionKbd));
    CheckErc(Request(prqReadContextEvent));

    do {
         Wait(MyExch, &prqHeader);
        ClearMsg();
        switch(prqHeader->rqCode) {

            case RQREADKBD :
                if(!MenuArea)
                    HandleAvailApps(Key);
                else
                    HandleAvailCont(Key);
                Request(prqReadKbd);
                       break;

            case RQREADACTIONKBD :
                HandleActionKey(Key);
                Request(prqReadActionKbd);
                        break;

            case RQCMREADCONTEXTEVENT :
                HandleEvents(prqReadContextEvent->pEventBlock);
                prqReadContextEvent->iVersion++;
                Request(prqReadContextEvent);
                       break;
        }
    } while(FOREVER);
}
```

Appendix C
Rebuilding a System to Allow More Contexts

The number of partitions specified at system build using the *nPartitions* parameter in the sysgen prefix file determines the number of contexts that Context Manager can support simultaneously up to a maximum of 20. The *nPartitions* parameter usually refers to the physical division of memory; however, Context Manager uses *nPartitions* as virtual entities. You specify the physical division of memory when you edit a Context Manager configuration file. In the following discussion, *nPartitions* refers to virtual partitions.

As you create a context, the system assigns a virtual partition to it. If the number of contexts the system supports is exceeded during the operation of Context Manager, the following message appears:

```
This version of the OS cannot support any more contexts.
```

You can rebuild your system to include more virtual partitions, and therefore more contexts (to a maximum of 20), by modifying the prefix file of your operating system. To determine the prefix file for your workstation, refer to the *CTOS Operating System Concepts Manual.*

The prefix file contains a line similar to the following: *%Set(nPartitions, 6).* Only prefix files for multipartition operating systems (the operating system Context Manager requires) have this entry.

To determine the number of partitions needed, you add the number of partitions the operating system requires, the number of partitions Context Manager requires, the number of contexts you want, and the number of installed system services. Table C-1 shows an example of this calculation.

You change the *n* in *nPartitions* to this sum. You must then assemble the sysgen files and link the CTOS version, following the directions in the *CTOS Operating System Concepts Manual.*

Table C-1. Sample Calculation of nPartitions Value

Environment:	User requires enough paritions for five application contexts and three system services. The value to specify for *n* in *nPartitions* is 11, calculated as follows:

Operating system partition	1
System services	+ 3
Context Manager partition	+ 1
CM user interface partition	+ 1
Application partitions	+ 5
Total partitions needed:	= 11

Glossary

A

Absolutely clean context state

Contexts are considered absolutely clean if they call either InitVidFrame or NotifyCM (8).

Absolutely dirty context state

Contexts are considered absolutely dirty if they perform the procedure NotifyCM (9). If the context is not the foreground context and is marked absolutely dirty, Context Manager suspends the application.

Action function key (F8)

When you press this key in the protected mode CM Editor, you can change the default action keys assigned to the Cut, Paste, and Halt features.

Application

In this guide, an application is a general term describing applications, utilities, or programs that you can run using Context Manager. A list of applications available appears on the Context Manager display under the heading Applications you can start.

B

Background

Any context other than the current context that is showing is said to be in the background.

Busy Wait status

Busy Wait status refers to applications that poll the keyboard or other devices. It causes all other contexts to be suspended.

C

Child context

A child context is a context that has been created, or can be switched or terminated by another context referred to as a parent context.

Clean state

Clean state indicates no side effects of the program running under Context Manager.

CM Add Application

This command allows you to add an application to a specified configuration file.

CMAPI.lib

CMAPI.lib is a file that contains procedural interfaces for using Context Manager and ICMS requests.

CmConfig.sys file

CmConfig.sys is a sample Context Manager configuration file. You can edit this file or create other configuration files to conform to the applications you want to access through Context Manager. *CmConfig.sys* is the default configuration file.

CM Editor

The CM Editor is a separate program that allows you to set up Context Manager to meet your needs. You use the CM Editor to edit the Context Manager configuration file and create additional user-specific configuration files.

CmEditor.run file

CmEditor.run is the CM Editor run file. This editor lets you edit a Context Manager configuration file.

CmInstall.run file

CmInstall.run is the run file that installs Context Manager.

CmInvoker.run file

CmInvoker.run clears the screen and creates frames similar to the Executive frame. It also allows pre-configured parameters to be passed to a context.

CmNull.run file

CmNull.run is the exit run file for all Context Manager contexts.

CmScreen.run file

CMScreen.run is the default Context Manager user interface.

CmVm.run file

CmVm.run is the Context Manager run file.

Cm.user file

Cm.user is a sample user file. If you type the user name **CM** at the SignOn form, this file lets Context Manager load automatically. You can edit this file or create other user files for automatic startup of Context Manager.

Config.sys

Config.sys is the CTOS configuration file that can be modified to use *Crashdump.sys* or another user-specified file as the swap file.

Context

A context is an active application. The most recently started context has control of the screen and keyboard.

Context handle

The context handle is a unique identifier that Context Manager assigns to an application when the application is started. The context handle is needed for applications using the CM and ICMS procedural interfaces.

Context Manager

Context Manager is a software utility that allows several applications, utilities, or programs to run concurrently on a CTOS multipartition Unisys operating system.

Context states

There are four context states: tentatively clean, tentatively dirty, absolutely clean, absolutely dirty. (Refer to individual glossary entries.)

D

Default value

A default value gets assigned automatically if you do not specify a value.

Dependent child context

A dependent child context is a child context that depends upon its parent for survival. If the parent of the dependent child context terminates, the dependent child context also terminates.

Dirty state

Dirty state means the program accesses the video directly, addresses the cursor through a port, or changes color or font directly.

Dollar directory

A dollar directory is where the system places temporary files.

E

Executive application

The Executive application refers to the Executive Command level. It lets you enter Executive commands through Context Manager. Applications that do not meet the parameter requirements of Context Manager must be accessed indirectly through the Executive rather than directly through Context Manager.

Exit Run File key (F4)

When you press this key from the CM Editor display, you can specify the exit run file for the CM user interface or the Context Manager Service.

F

Foreground context

The Foreground context is the context that currently controls the screen and keyboard. If you switch contexts, the context you switch to becomes the foreground context.

I

ICMS

The InterContext Message Service is an installable system service that lets contexts communicate with each other.

ICMS key (F9)

You use this key to access the InterContext Message Service menu in the CM Editor.

ICMS.run file

This is the InterContext Message Service run file.

Independent child context

An independent child context is a child context that does not depend upon its parent context for survival. If the parent context terminates, the independent child context does not automatically terminate.

Invisible context

An invisible context is an application partition which is started in the background. It does not have as associated function key and its existence is not reported to the CM user interface.

IPC

Interprocess Communication facility is an operating system facility that uses messages and exchanges to synchronize communication between processes.

L

Long-lived context

A long-lived context is a context that continues to exist even though the CM user interface has chained to another runfile. Long-lived contexts can exist either in the foreground or the background.

M

Memory Area

The Memory Area appears in the lower left portion of the Command Editing Area on the CM Editor display when you press **Memory (F1)**. It contains fields in which you enter memory and partition information.

Memory key (F1)

When you press this key from the CM Editor display, the system moves the highlight to the first field of the Memory Area.

More key (F10)

This key is available after you have entered a name in the *Command Name* field of the CM Editor and pressed **RETURN** or **Create (F5)**.

P

Parent

A parent context calls upon Context Manager to create, switch, or terminate other contexts (known as child contexts).

Partition

A partition is a defined portion of workstation memory. Under Context Manager, the operating system resides in the low end of memory and Context Manager resides in the primary partition. As you begin each application, Context Manager creates the partition and places the application in that partition.

R

Request.CM.sys

This is a loadable request file defining Context Manager Service requests.

S

Shared video context

A shared video context is a context that maps its character map to its parent context character map, allowing the child context to direct video output to the screen even while it is running in the background.

Swap file

A swap file is a file to which Context Manager can swap contexts and thus increase the number of available contexts. You have the option of creating a swap file. However, swapping increases the flexibility of Context Manager by allowing you to open up to twenty applications at the same time.

Swap key (F8)

When you press this key from the CM Editor display, the system prompts you to identify the swap file for this configuration file.

Swapped context

A swapped context refers to a context that Context Manager temporarily puts in your system's swap file, thus leaving room for you to open more applications when your system partitions are full. The swapping capability and the extent of swapping are dependent upon the size of the configured swap file.

Tentatively clean context state

All contexts start in a tentatively clean state.

Tentatively dirty context state

A context enters a Tentatively dirty context state before it can be classified as clean, dirty, absolutely clean, or absolutely dirty.

Video pointer map

The video pointer map is a system structure for each memory partition. It is an array of pointers, one for each line of the video screen, that always points to the location of the associated line of the application's character screen.

Index