UNISYS

XE500
BTOS
Programming Reference Manual

Relative to Release
Level 7.0

Priced Item

May 1988
Distribution Code SA
Printed in U S America
5029077

UNISYS

XE500
BTOS
Programming Reference Manual

Relative to Release
Level 7.0

Priced Item

May 1988
Distribution Code SA
Printed in U S America
5029077

The names, places and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual living or otherwise, or that of any group or association is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the CLASS specified as 2 (S.S.W.:System Software), the Type specified as 1 (F.T.R.), and the product specified as the 7-digit form number of the manual (for example, 5029077).

About This Manual

This manual describes XE520 BTOS programming. It provides supplemental information concerning Unisys XEBTOS operating system software for systems and applications programming. In conjunction with the BTOS operating system reference manual, this manual will assist the programmer in understanding the concepts of XEBTOS; Inter–CPU Communications (ICC); and system requests, procedures, and services. This manual describes the specific features provided in XEBTOS that are relative to programming an XE520 system.

Who Should Use This Manual

Systems and applications programmers should use this manual to understand XEBTOS concepts. The programmers should be experienced at systems or applications programming and have a basic understanding of BTOS and the XE520.

How to Use This Manual

This manual should be used as a supplement to the BTOS operating system reference manual, which describes general BTOS information. This manual details XE520–specific BTOS operating system features, requests, procedures, and services.

If you are using this manual for the first time, scan the table of contents and review the topics before you start. To find definitions of unfamiliar terms, use the glossary; to locate specific information, use the index.

How This Manual is Arranged

- Section 1 provides a brief overview of XEBTOS, the XE520 operating system.
- Section 2 describes the concepts pertaining to routing information between processors developed for the XE520 environment.
- Section 3 describes the ICC facility, which is an extension of the Interprocess Communications (IPC) facility.
- Section 4 lists workstation request block level requests that have been enhanced to use the XE520.
- Section 5 describes the services and procedures that have been added to BTOS for the XE520 environment.

A glossary and an index follow section 5.

Conventions

The following conventions are used throughout this manual:

- All numbers are decimals unless otherwise noted.
- XE520 or XE520 system refers to the XE520 hardware.
- XEBTOS refers to the BTOS operating system that runs on the XE520 hardware.
- The term BTOS refers to BTOS II.
- Executive commands appear in uppercase.
- Syntactical elements which are not literal (that is, to be specified by the user), are underlined.
- Processor board names refer to both the standard board and the X-board. For example, CP refers to both CP and CP-X.

Related Product Information

For suggested error message responses, refer to your System Status Codes Reference Manual.

For an explanation of BTOS and information about request code processing, refer to your Operating System Reference Manual.

For more information about BTOS commands, refer to your Standard Software Operations Guide.

For more information about BTOS system calls and structures, refer to Volume 2 of the *BTOS Reference Manual* or the *BTOS II System Procedural Interface Reference Manual.*

For information on installing and implementing XEBTOS, refer to the *XE500 BTOS Installation and Implementation Guide.*

For information on administering the XE520 BTOS system, refer to the *XE500 Administration Guide.*

For information on performing those XE520 tasks that are routinely performed by anyone using the XE520, refer to the *XE500 BTOS Operations Guide.*

For information on customizing the operating systems that run on both workstations and the XE520 processors, refer to the *BTOS II Customizer Programming Guide.*

Contents

Figures

Tables

System Description

BTOS Features

The XE520 BTOS operating system is essentially the same operating system that supports distributed processing on Unisys workstations. BTOS provides system–wide file and communications services. Generally, the major features of BTOS include:

- multiprogramming
- event–driven priority scheduling
- Interprocess Communication
- Inter–CPU Communication
- local resource sharing networks
- overlays
- file management system

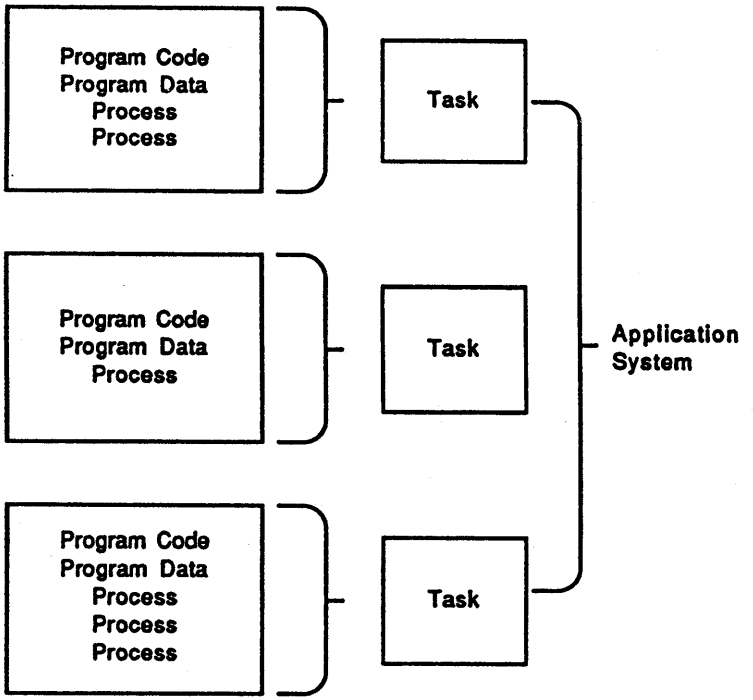
Other BTOS features include video display with multiple split screen and color; an unencoded keyboard (only encoded versions of keyboard operations are available on the XE520); communication line support; and sequential, direct, and indexed sequential access methods.

Multiprogramming

BTOS provides a real–time, multiprogramming environment. Multiprogramming is supported at three levels: application systems, tasks, and processes. (See figure 1–1.)

- Any number of application systems can coexist, each in its own memory partition. An application system is a collection of one or more tasks that access a common set of files and implement a single application.
- Any number of tasks (that is, executable programs) can be loaded into the memory of a partition and be independently executed.
- Any number of processes can independently execute the code (instructions) of each task. A process is the basic element of computation that competes for access to the processor.

Figure 1–1 **Relationship of Processes, Tasks, and an Application System**



Event–Driven Priority Scheduling

To meet the need for high performance, the operating system kernel provides efficient event–driven priority scheduling.

Each process is assigned one of 255 priorities and is scheduled for execution based on that priority. Whenever an event, such as the completions of an I/O operations, makes a higher priority process eligible for execution, rescheduling occurs immediately. This results in a more responsive system than scheduling techniques that are entirely time–based.

Interprocess Communication

The kernel's IPC primitives, such as Request and Wait (or Check) are the primary building blocks for synchronizing process execution and transmitting information between processes.

Inter–CPU Communication (ICC)

The ICC facility, which is an extension of IPC, provides for communication between CPUs among the different processor boards on the XE520.

If the requested system service is on the same XE520 processor board as the client process, the kernel uses IPC. If, however, the service is on a different processor board, the kernel uses ICC. ICC passes request and response messages between processor boards.

The XE520 is compatible with the workstations at the request level. Whether your program runs on an XE520 or a workstation, it can access system services in the same way (either by using the request procedural interface or by calling the kernel primitives).

Messages and Exchanges

When a process needs to send a message to another process, it sends the address of its message to an exchange. Sending only the address, instead of the actual message, reduces system overhead. At the exchange, the first process can either wait for a response to its message from the second process or poll for a response. If it waits for a response, its execution is suspended until the response occurs; this is how the execution of different processes is synchronized. If the first process polls for a response, it resumes execution of the code which sent the message until a response occurs.

A single process can serve several exchanges. It can send messages associated with different priorities of work to different exchanges. This feature can be used to set priorities for the work the process is to perform. In addition, several processes can serve the same exchange, thereby sharing the processing of a single kind of message.

All processes share a finite pool of exchanges. The termination of a process automatically deallocates its exchanges. A process that continues to work but has no further use for an exchange should relinquish it.

System Service Processes

BTOS includes system service processes which manage various classes of system resources, such as files and memory. These processes receive requests for their services from other processes through IPC, and they are scheduled for execution in the same manner as application processes.

A system service process is the only software element that accesses a particular resource, and the interface to a system service process through IPC is formalized.

Local Resource Sharing Networks (Clusters)

The cluster configuration that executes in the XE520 processors that run XEBTOS is similar to that of the BTOS workstations. The XE520 provides services to processes running on workstations such as file management, queue management, 3270 terminal emulator, and database management. Services such as video and keyboard management are performed at the workstations.

Intercluster communications in the XE520 are handled by Cluster Processors (CPs) in the XE520 enclosures. Workstations are connected to CPs with high-speed RS-422 channels.

Overlays

The code in the run file of a program using the Virtual Code management facility either is part of one of several overlays, or is resident. (In this discussion, a program that uses the Virtual Code management facility is called an overlay program.)

When the overlay program begins execution, the resident code is loaded into memory, where it remains for the duration of the program's execution. When a call is made to a procedure in one of the overlays, the Virtual Code management facility reads that overlay into memory into an area of memory called the overlay zone so that the program can continue execution.

The Virtual Code management facility keeps as many overlays as possible in memory at once. When another overlay that would exceed the available space is called into memory, the Virtual Code management facility uses a least-recently-used (LRU) algorithm to determine which currently resident overlay to discard.

XEBTOS supports the segment swapping implementation of virtual memory, allowing overlaid run files to execute on every processor board of the XE520.

Refer to your *BTOS II Language Development Programming Guide* for more information on overlays.

File Management System

The file management system provides a hierarchical organization of disk file by node, volume, directory, and file. Volumes are automatically recognized when placed on line. A file can have a 50-character file name, a 12 character password, and a file protection. You can dynamically expand or contract a file without limit as long as it fits on one disk. You can protect the file by a volume directory, or file password, depending on the level number you assign to a file. You can control concurrent file access by read (shared) and modify (exclusive access) modes.

While providing convenience and security, the file system management also provides the full throughput capability of the disk hardware. This includes reading or writing any 512-byte sector of any open file with one disk access, reading or writing up to 65,024 bytes with one disk access, overlapping input/output with process execution, and optimizing disk arm scheduling.

In a cluster configuration, files can be located at cluster workstations that have local storage, as well as on the XE520 master.

Two File Header Blocks are created for each volume with a parameter that is set when initializing the disk. This defaults to creating the secondary volume structures. The duplication of critical volume control structures protects the integrity of disk file data against hardware malfunction. You can recover a file that has been backed up by using the Executive MRESTORE command, if either of the File Header Blocks is valid.

BTOS Versions

In workstation BTOS, memory can be divided into more than two partitions, each with its own application system, with all the application systems executing simultaneously.

XEBTOS distinguishes between system and application partitions but not between primary and secondary partitions.

System services can be installed only when a single non-system partition exists.

In the XE520 system, there is a tailored version of BTOS on each processor. Each version can be customized to meet the needs of an individual installation. This is discussed in detail in the *BTOS II Customizer Programming Guide.*

General Structure of BTOS

The five basic components of the BTOS operating system are:

- the kernel
- system service processes
- system common procedures
- object module procedures
- device and interrupt handlers

The kernel is the most primitive yet powerful component of the operating system. It provides process management and IPC facilities. The kernel schedules process execution, including the saving and restoring of process context.

A process, the basic element of computation that competes for access to the processor, consists of the following:

- the address of the next instruction to execute on behalf of this process
- a copy of the data to be loaded into the processor registers before control is returned to this process
- a stack
- a default response

When you add a system service process, you assign one of 255 priorities to each process so that BTOS can schedule the process's execution appropriately.

The kernel's IPC primitives are the primary building blocks for synchronizing process execution and transmitting information between processes.

System service processes are standard processes that manage system resources. These are scheduled for execution in the same way as application processes. The five major categories of system services are:

- task management
- network management
- device management
- memory management

A system service can be accessed directly by using the Request and Wait primitives, or indirectly, by using a procedure call from a high-level language. Accessing a system service directly allows an increased degree of concurrence between multiple I/O operations and computation.

System common procedures are standard procedures that perform some common system functions. They are executed in the same context and with the same priority as the invoking process. Examples of system common procedures are EXIT, which terminates the execution of an application system.

Object module procedures are supplied as part of an object module file or library. They are not part of BTOS itself. Most application systems require only a subset, not a full set, of these procedures. An example of an object module procedure is the Sequential Access Method (SAM).

BTOS device handlers and interrupt handlers are accessed indirectly through the convenient interfaces of the system service processes.

You can add your own system processes, system common procedures, device handlers, and interrupt handlers to BTOS at system build. System build is the name for the sequence of actions necessary to construct a customized version of BTOS.

BTOS Concepts for the XE520

This section briefly describes BTOS concepts developed for the XE520 environment, concepts that pertain to the routing of information between processors.

Before reading this material, you should be familiar with BTOS process management. If not, see the discussions on interprocess and interstation communications in the *BTOS II System Reference Manual.*

Inter–CPU Communications

The Inter–CPU Communications (ICC) facility is an upward–compatible extension of the Interprocess Communications (IPC) facility. ICC enables a process on one processor to obtain a service provided by a process on another processor.

When a client process on a particular processor requests a system service, a request block with all the necessary information is constructed. If the requested service resides on a different processor, the request block is queued at the ICC Server exchange on the first processor. The ICC Server converts the request block into a message which is transmitted via the system bus to the ICC Server on the processor where the requested service resides.

The ICC Server on the second processor then converts the message back into a request block. A CPU Description Table (CDT) on the second processor is used to locate the exchange of the system service that is to perform the requested service, and the request block is queued there.

The response to the request follows a similar route from the second processor to the first processor and undergoes similar transformations. All processors have ICC Servers and CDTs for interboard communications.

Inter–CPU Communications are implemented with the following concepts, all of which are new to BTOS for the XE520.

Slot Number

The slot number is the eight-bit binary number that uniquely identifies a particular processor board in an XE520 system. Processor slot numbers range from a high of hex 77 to a low of hex 20. The slot number is used by certain XEBTOS routines to identify a particular processor and by the hardware to accomplish interboard addressing. Table 2-1 shows slot number assignments for multienclosure systems.

You can use the GetProcInfo and the GetSlotInfo operations to retrieve such hardware information and, thereby, explicitly control ICC routing. You would use these operations if using one of the XE520 routing types defined in section 3 is not sufficient.

Doorbell Interrupt

Each XE520 processor can send an interrupt, called a doorbell interrupt, to any other board in the system. During initialization, a doorbell interrupt awakens a processor that has just been loaded with that processor's version of BTOS. During ICC communication, a doorbell interrupt alerts the target processor that a request or response needs processing.

Table 2-1 **Slot Numbering for Multienclosure Systems (shown in hexadecimal numbers)**

Enclosure	P0	P1	P2	P3	P4	P5	P6	P7
Base	70	71	72	73	74	75	76	77
Expansion 1	60	61	62	63	64	65	66	67
Expansion 2	50	51	52	53	54	55	56	57
Expansion 3	40	41	42	43	44	45	46	47
Expansion 4	30	31	32	33	34	35	36	37
Expansion 5	20	21	22	23	24	25	26	27

Linear Format

XEBTOS describes structures to be read by the processor using a linear pointer. A linear pointer is a 4–byte quantity in which the most significant byte is at the lowest address. A linear pointer is absolute, not segment based.

Like a linear pointer, a linear offset has the most significant byte at the lowest address; however, it is a two–byte quantity. The byte ordering is opposite to the 80x86 convention, which puts the most significant byte at the highest address. Linear offsets are used in XE520 BTOS to describe structures that must be read by both 80x86–based processors. A linear offset within a structure is always taken to be the offset relative to the base structure.

CPU Description Table

Each XE520 processor contains a CPU Description Table (CDT). You can obtain the address of the CDT by calling GetpStructure with a structCode value of 27. The table describes the processor to other processors, contains the offsets of the circular buffers used by other processors to send ICC requests and responses, and contains some routing information. The CDT for the master File Processor (FP) contains additional routing information and tables used to translate line and workstation numbers into particular slot number–port number pairs. Table 2–2 presents the structure of the CDT.

Table 2–2 **CPU Description Table (CDT) Structure**

Offset	Name	Size (Bytes)	Contents
0	filler$0	8	Unused
8	bProcessorType	1	Type of processor FP = 10 TP = 11 CP = 12 SP = 13 DP = 14
9	fWatchDog	1	Flag that is set once per second by the master FP and must be reset within one second or the processor is considered dead.
10	ibRqTakePtr	2	Current request linear offset.
12	ibRqPutPtr	2	Next available request linear offset.
14	ibRqStartPtr	2	Base of request circular buffer.
16	ibRqEndPtr	2	End of request circular buffer.
18	ibRespTakePtr	2	Current response linear offset.
20	ibRespPutPtr	2	Next available response linear offset.
22	ibRespStartPtr	2	Base of response circular buffer.
24	ibRespEndPtr	2	End of response circular buffer.
26	fLockByte	1	Used to ensure no race for the request and response circular buffer.

Table 2–2 **CPU Description Table (CDT) Structure** (continued)

Offset	Name	Size (Bytes)	Contents
27	bInitErrorStat	1	If initialization detects an error, the code is placed here.
28	bMemorySize	1	Memory size in kB/128.
29	bBootStruct$FF	1	Signature, value – hex OFF
30	bBootStruct$0	1	Signature, value – hex 0.
31	bBootStruct$A6	1	Signature, value – hex 0A6.
32	bBootCommand	1	Allowable values:
	fRunning		0
	fBootMe		1
	fDumpThenBoot		2
	fDumpThenError		3
	fFailedDiagnostics		Hex 0F0
	fRunningDiagnostics		Hex 0F1
33	bMasterFp	1	Master FP slot number
34	fOsInitialized	1	Hex OFF when OS has initialized.
35	fFiller	5	
40	rgbFPXlate	8	Array of FP slot numbers.
48	rgBusConfig	240	Array of slot number/processor types.
288	filler	100	Reserved for future expansion.
388	oTermTable	2	Linear offset of table that translates workstation number to slot number/port.

Table 2–2 **CPU Description Table (CDT) Structure** (continued)

Offset	Name	Size (Bytes)	Contents
390	sTermTable	2	Size of preceding table.
392	oLineTable	2	Linear offset of table that translates line number to slot number/port.
394	sLineTable	2	Size of preceding table.
396	oRqRoute (17)	34	Array of information for routing requests on each level to other processors.
430	sRqRoute (17)	34	Size of segment each level.

Terminal Output Buffer

The terminal output buffer is the structure used to describe the virtual terminal output device. It consists of a circular buffer and associated information. A linear pointer to the terminal output buffer is returned by both WhereTerminalBuffer and OpenTerminal. There is no request interface for output; instead, the client process manipulates the output buffer data structure described in table 2–3. Note that all operations that involve modification of the put pointer (loPotPut) must be single–threaded; the lock–byte, foLock, ensures this. File system requests to XE520 masters are not single–threaded through the master FP or DP.

Table 2–3 **Output Buffer Data Structure**

Offset	Name	Size (bytes)	Contents
0	loOutTop	2	Offset from header to first data byte in the first in, first out (fifo) buffer. The first byte is the one at the lowest address (nearest hex 0000).
2	loOutBot	2	Offset from the header to the bottom of the fifo. The bottom byte is one past the data byte occupying the highest address.
4	loOutGet	2	Offset from the header to the byte that the output interrupt routine will fetch next. After this byte is fetched, this pointer is incremented. Thus, the discipline is post–incremented.
6	loOutPut	2	Offset from the header to the next client byte. Like the Get pointer, this pointer is post–incremented. Note that if the two pointers are equal, the buffer is empty. If the module increment from the Put pointer equals the Get pointer, the buffer is full. All manipulations of the Put pointer must be done with the lock byte set.
8	foLock	1	A semaphore used to protect the Put pointer from multiple writers (including the Terminal Processor itself). Setting the sign of the byte to nonzero will keep others from using the Put pointer.

Table 2–3 **Output Buffer Data Structure** (continued)

Offset	Name	Size (bytes)	Contents
9	foCarrier	1	A boolean flag describing the sense of the line carrier with hex 000 representing False and hex OFF representing True. For dataset lines, this flag tracks the DCD (data carrier detect) signal. For hardwired lines, foCarrier is always true.
10-17	rgbAP	8	An 8-byte area reserved for the use of the clients of the port.

Channel Number

The channel number is a logical number used to designate a particular port on a Cluster Processor (CP) or Terminal Processor (TP).

Channel numbers for a CP are as follows:

Device	Backpanel Label
8274A	Channel 1
8274B	Channel 2
8251	Channel 3

Channel numbers for a TP are as follows:

Device	Backpanel Label
8274#1A	Channel 1
8274#1B	Channel 2
8274#2A	Channel 3
8274#2B	Channel 4
8251#2	Channel 5
8251#1	Channel 6
8251#4	Channel 7
8251#3	Channel 8
8251#6	Channel 9
8251#5	Channel 10

Inter–CPU Communication

The Inter–CPU Communication (ICC) facility provides for communication between CPUs among the different processor boards on the XE520. ICC is an extension of Inter–Process communication (IPC).

The XE520 is compatible with the workstations at the request level. Messages passed between a client and a system service on the same processor board use IPC. The kernel routes the request to the system service exchange; the system service performs its function and responds to the client's exchange, acknowledging service completion. Requests routed locally on a single XE520 processor board also use IPC.

When a client requests a system service, the kernel examines its request routing table to determine, for example,

- if the request block is correctly formed
- to which system service the request is to be sent

These actions are taken in the case of ICC or IPC. However, the destination to which the request is sent determines if the request is handled as a normal IPC message or if it is to be routed by ICC.

ICC involves interboard routing or the passing of the request and the response message between processor boards. To accomplish this, ICC uses

- processor boards identified by slot numbers
- XE520 routing type information in the operating system's request routing table
- an ICC Server Agent on each processor board, which issues requests on behalf of a client on a different processor board
- communication between processors over a high–speed bus
- linear pointers and linear offsets for interboard addressing

- Y–blocks and Z–blocks for storing copies of request blocks
- a request ring buffer and a response ring buffer in a CPU Description Table (CDT) on each processor board
- a doorbell interrupt

Processor board slot numbers, linear pointers and linear offsets, and doorbell interrupts are discussed in section 2.

XE520 Routing Types

Table 3–1 describes each of the XE520 routing types used to define requests on the XE520. If you are writing a system service for the XE520, you will need to include an XE520 routing type in your system service request definition(s). For more information, refer to the *BTOS II System Reference Manual.*

Table 3–1 **XE520 Request Routing Types**

Field	Description
rLocal*	The request is served on the same board. The service exchange is indicated by the service exchange field in the operating system request routing table. The request is routed remotely, however, if a file specification for a remote board is included in the request block. In this case, a file system filter calls RequestRemote and routes the request to the board specified by a slot number in the Master Processor global slot number table.
rRemote*	Same as rLocal if the request is served locally. If the request is not served locally, it is searched for in the Master Processor global slot number table. When a system service calls ServeRq during installation, ServeRq updates the Master Processor global slot number table to reflect the system service's slot number.
rMasterFP	The request is routed to the Master FP.
rHandle*	The request is routed by an indexed field in the file handle.

Table 3–1 **XE520 Request Routing Types** (continued)

Field	Description
rFileId	The first byte of control information in the request block contains the slot number of the board to which the request is routed.
fMasterCp	(Unused)
rLine#	The request is routed to the Cluster Processor (CP) that handles this line. This routing type is used by the MegaFrameDisableCluster operation. Each CP has two lines. For example, CP00 has lines 1 and 2; CP001 has lines 3 and 4; and so on.

*This type is frequently used.

XE520 Linear Addressing

XE520 linear addressing becomes important if you are writing programs that will run on multiple boards. For example, if a client requires a system service located on a processor board other than the one that the client is on, you cannot use equivalent addresses in your program logic.

Blocks

Blocks are areas of memory allocated for ICC and for cluster communication. Y–blocks and Z–blocks are used for holding ICC messages. A Z–block is used if the message can fit into a small number of bytes; otherwise, a Y–block is used. The size and number of these blocks is determined at system initialization.

Interboard Routing

Each XE520 processor board provides for the sending and receiving of messages. In the following description of interboard routing, the actions for the sending and receiving of messages are described separately.

Sending Requests

In figure 3–1, a client calls a Request located at the address referenced by the pointer (pRg). The kernel uses the request code (Rq) as an index into the routing table to determine the XE520 routing type. The routing type tells the kernel where to route the request.

How A Message Is Sent

Sending a message is summarized in figure 3–1.

Local Routing?

If request routing indicates that the request is to be served locally and a local server exists, ICC is not used. The request is routed using the normal procedures of IPC.

In figure 3–1, the pointer (pRq), when referencing a request served locally is a logical memory address.

Remote XE520 Processor Routing

If request routing indicates that the request is to be served off board (for example, on a different XE520 processor), ICC is used to send the request.

To send the request, the kernel

1 Enters the client's return address into the CDT request ring buffer on the receiving XE520 processor.

  The ring buffer entry consists of 5 bytes that describe the client's return address: 1 byte defines the client board's enclosure and slot number; 4 bytes define the client's request block linear address.

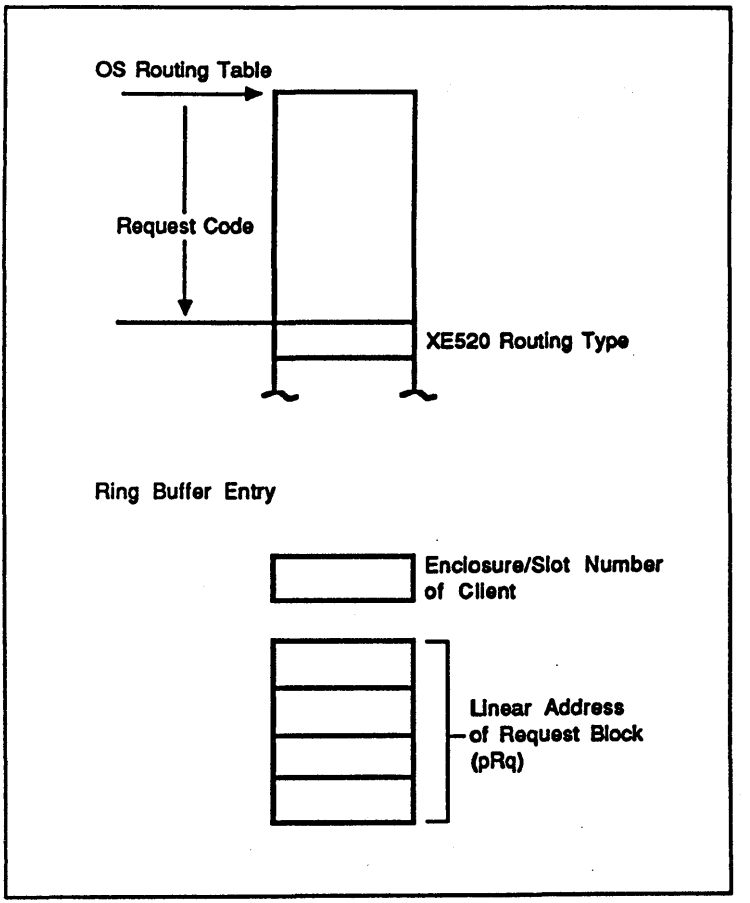
2 Sends a doorbell interrupt to the receiving board.

Sending Responses

Figure 3–1 also shows sending responses.

A response to a request originated remotely must be sent back to the client on the requesting XE520 processor. The kernel recognizes a response to be routed remotely by the request block response exchange number.

Figure 3–1 **How a Message is Sent**



To return the remotely response, the kernel takes the following actions:

1 copies the pb/cb response buffers and a status code to the client's request block memory on the client's board

2 frees the Z–block (or Y–block) holding a copy of the client's request block (in the server's processor memory)

3 places the client's return address in the CDT response ring buffer on the client's board

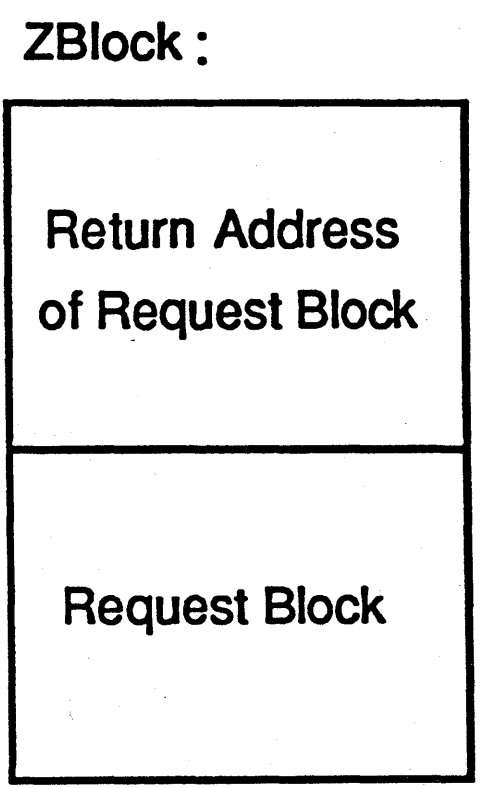
4 sends a doorbell interrupt to the client's XE520 processor

How A Message Is Received

Receiving a message is summarized in figure 3–2.

In figure 3–2, the doorbell interrupt from the sending board alerts the ICC Server Agent on the receiving XE520 processor that it has received a remote message in one of its CDT ring buffers.

Figure 3–2 **How a Message Is Received**



The ICC Server Agent examines the ring buffer entry to see if it is a request or a response.

Request?

If the ring buffer entry is a request, the ICC Server Agent

1 Calculates the size of the request by examining the size of the client's request block memory. The ICC Server Agent uses the size to reserve a Z–block (or a Y–block) in the receiving XE520 processor's memory.
2 Copies the request block contents and the client's return address into the reserved Z–block.
3 Calls Request, providing the memory address of the Z–block.

   In figure 3–2, Request(pZblock) repeats the sending requests procedure in figure 3–1.

The kernel on the receiving board routes the request to the specified service exchange. The message is processed using IPC.

Response?

If the ring buffer entry is a response, the ICC Server Agent calls the Respond primitive (pRq) to alert the kernel on the receiving board to route the response back to the client's local response exchange.

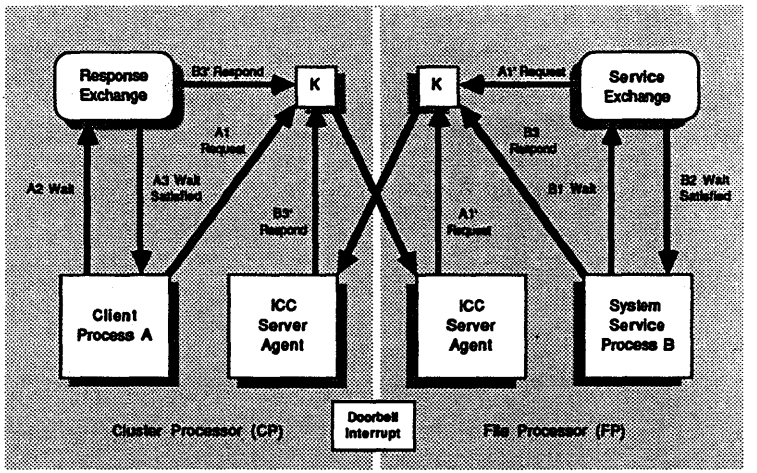
Sending and Receiving Messages

Figure 3–3 shows the interaction of client A on a Cluster Processor (CP) board and system service B on a File Processor (FP) board.

In figure 3–3, client A on the CP board requests (A1) a service provided by system service B on the FP board. The kernel on the CP board places the request block return address in the FP board's CDT request ring buffer and rings the FP's doorbell.

The ICC Server Agent on the FP board copies the request block contents to a Z–block (or Y–block) in the FP processor and calls Request (A1'). The kernel on the FP board examines Request (A1'), and sends it to system

Figure 3–3 **Interaction of Client and System Service Using ICC**



service B's service exchange, satisfying system service B's Wait (B2). System service B processes the request and responds (B3).

The kernel on the FP board acts on the Respond (B3) by copying the response back to client A's request block, placing an entry in the CP's CDT response ring buffer, and ringing the CP's doorbell.

The ICC Server Agent on the CP board examines the response ring buffer and calls Respond (B3'). The kernel on the CP board sends Respond (B3') to client A's response exchange, satisfying the client's Wait (A3).

Note that Request and Respond function in two ways in figure 3–3. One set of Request and Respond send information to another board; the other Request and Respond are queued at an exchange.

Operations

The ICC operations are described below. These operations are not available to BTOS workstations. (Refer to section 5 for a complete description of each operation.)

GetProcInfo returns the name of the processor on which the caller is running.

GetSlotInfo determines the slot numbers of other processors in the XE520.

RemoteBoot causes another dormant processor to be bootstrapped with a specified System Image.

RequestRemote requests a system service from a remote processor by sending the request to the ICC Server Agent of that remote processor.

Requests Enhanced in XEBTOS

XE520 processor boards and workstations execute most of the same BTOS II standard software run files, and the XE520 and workstations support a single version of CTOS.lib. For more information, refer to the *XE500 BTOS Administration Guide*.

In addition, most of the XEBTOS requests have the same names and functions as their workstation counterparts and are compatible at the request block level. This section lists those requests which have been enhanced to make use of the XE520 hardware.

General Differences

The general differences between XEBTOS and workstation BTOS II are described in the following subsections.

Alternate Request Procedural Interface

With XEBTOS, you can use an alternate request procedural interface to issue requests with other than the default user number. This eliminates the requirement for you to manually construct request blocks. You invoke this feature by prefixing the name of an operation with "alt" and supplying the desired user number as the first parameter to the procedure. For example, to issue a CloseFile request with user number 5 and file handle fh, you make the calling sequence AltCloseFile (5, fh).

Mode Parameters

XEBTOS supports the mode peek parameter. Mode peek is a possible value for one of several mode parameters; examples of additional parameters are mode read, mode write, mode access, mode append and mode modify. All of these parameters are 16-bit values representing ASCII constants. In these ASCII constants, the first character (m) is the high-order byte and the second character (r, w, and so forth) is the low-order byte.

Mode peek allows read access to a file without preventing a different user from modifying the file. Mode peek opens a file in the same way mode modify opens a file. The operating system puts existing mode peek handles into a state where subsequent operations return error 235. Then it allows mode modify to open the file. Mode peek is useful with files like the Executive Message file, which you may want to update but find constantly opened.

Programmable Interval Timer

The XE520 programmable interval timer runs at a frequency of 19.5 KHz (providing a 51.3 msec resolution) as it does on workstations. You may have to alter your applications to run them under the 51.3 msec resolution. Refer to the *BTOS II Customizer Programming Guide.*

Direct Printing

XEBTOS does not support direct printing.

Using the Video Control Block

XEBTOS does not maintain the video control block (VCB). The system common pointer which pointed to the VCB in workstation BTOS now points to the CPU description table (CDT). Therefore, programs that make direct use of the video control block must be converted before they can run under XEBTOS. For more information on the CDT, refer to section 2.

Using Certain SysConfigBlk Fields

You must change programs running on workstation BTOS which use the fLocalFileSystem and WsType fields of the System Configuration Block to run under XEBTOS. If the Hardware Type field in the System Configuration Block contains a value outside the range between 10 and 19, assume your system is now running on workstation BTOS and change the fLocalFileSystem and WsType fields.

Using VAM and VDM Operations

The XE520 supports a subset of the Video Access Method (VAM) operations only and one Video Display Method (VDM) operation only. Supported VAM operations include PosFrameCursor, PutFrameAttrs, and ScrollFrame; the supported VDM operation is ResetFrame.

Operations within the VAM subset and the operation within VDM are restricted to provide the functionality of a dumb terminal and do not cause any operation to take place. In addition, PutFrameChars, a VDM operation, outputs characters on the screen but ignores the specified screen coordinates.

Using the Cluster Status Utility

The Cluster Status utility reports cluster errors seen by workstations. Cluster Status utility polling and response to polling occur at interrupt level, and termination requests are sent as a single transaction.

Line 0 is invalid when running the workstation Cluster Status utility from a workstation that is clustered to an XE520. This is the only restriction.

Byte–stream Video

In XEBTOS, byte–stream escape sequences (beginning with hex 0FF) are ignored. The output model is that of an RS232–C serial terminal. Standard byte–stream sequences are supported.

Communications Byte Streams

Communications byte streams support the 8251 chip on both the Cluster Processor and Terminal Processor processor boards. The support of the chips allows you to access port C on the CP board and ports E through J of the TP board in the same way as other RS232–C communication ports.

The interpretation of n as in [Comm]n depends on whether it is executed on a Terminal Processor or on a Cluster Processor. n has to be running in the processor with the port that is being addressed. Table 4–1 illustrates the different interpretations.

Table 4–1 **Processor Interpretation of n**

n	Channel	Terminal Processor	Cluster Processor
A	1	8274#1A	8274A
B	2	8274#1B	8274B
C	3	8274#2A	8251
D	4	8274#2B	—
E	5	8251#2	—
F	6	8251#1	—
G	7	8251#4	—
H	8	8251#3	—
I	9	8251#6	—
J	10	8251#5	—

Specific Request Difference

Changes in specific requests that have been enhanced to make use of the XE520 hardware are listed in this subsection. Refer to the *BTOS II System Reference Manual* for more information.

AllocExch

The AllocExch service allows direct interboard message passing. XEBTOS assigns unique exchange values across all XE520 processor boards to accomplish this. Applications should not expect the values of exchanges to be in any particular range under XEBTOS. Exchanges can have any value up to 65,535.

RequestDirect

The RequestDirect kernel primitive allows direct interboard message passing. Each XE520 processor board has a unique exchange value; the value is returned whenever you call the AllocExch service. RequestDirect works with the Generic Print System (GPS) to install multiple printer drivers. The GPS routing switch uses RequestDirect to issue work to drivers installed on different boards.

SetTrapHandler

The SetTrapHandler operation does not create trap handlers that are part of a partition's context; therefore, trap handlers are not switched when processes run in a different context. Instead, the trap handlers are global to the processor board on which SetTrapHandler executes. Because of this, only one program per processor board can use software traps. The software trap handler's primary client is the Pascal floating-point run-time library.

QueryDCB

The QueryDCB service copies the Device Control Block (DCB) of the specified device to the specified memory area. The information is truncated if it cannot fit in the memory area. There is a DCB for each physical device. For example, the DCB for a disk includes the number of tracks per disk and the number of sectors per track. The DCB points to a chain of I/O blocks.

QueryDCB needs no password. To prevent security violations, a string of zeros is returned in the DCB device (sbPassword) field.

The DCB structure is slightly different under XEBTOS. Table 4-2 lists these changes.

Table 4–2 **DCB Changes in XEBTOS**

Offset	Field	Size (Bytes)	Description
65	gaplength	1	Wd1010/Wd2010 intersector gap parameter.
66	writePrecompt	1	Cylinder after which write precomposition is used.
67	stepRate	1	Wd1010/Wd2010 step rate value.
76	fEccFormat	1	True if disk formatted in ECC mode.
77	reserved	1	
78	fOnLine	1	True if drive is on line (used in polling).
79	fInhibitEcc	1	For SMD drives.

GetVhb

The GetVhb service copies the Volume Home Block (VHB) of the specified device to the specified memory area. If the specified area is not large enough to hold the requested information, XEBTOS truncates the information.

GetVhb requires no password. To prevent security violations, zeros are returned in the volPassword field of the VHB.

The VHB structure is slightly different under XEBTOS, as shown in table 4–3.

Table 4–3 **VHB Changes in XEBTOS**

Offset	Field	(Bytes)	Description
247	SeekStepRate	1	Wd1010/Wd2010 step rate value.
248	gapsize	1	Wd1010/Wd2010 intersector gap parameter.
249	WritePreCompCyl	1	Cylinder after which write precomposition is used.
250	devType	1	00 = fixed disk 81 = EPI tape
251	reserved	5	

ServeRQ

The ServeRQ service is the part of a dynamically installed system service process that serves a specified request code. Future requests containing this request code are queued at a specified exchange.

ServeRQ functions the same way on XEBTOS as on workstation BTOS. The XE520 enhancement involves the visibility of the request. If the routing type of the request is rRemote, the server is globally visible to all processors in the XE520 system. Otherwise, only the processor on which ServeRQ is issued can see the server.

SetCommISR and ResetCommISR

The SetCommISR service establishes the Communications Interrupt Services Routines (CISRs) for the specified communications channel. Separate CISRs are established to process transmit, external/status, receive, and special receive conditions. The ResetCommISR purges the CISRs.

Allowable values for the parameter iLine, which identifies the SIO channel, are extended on a Cluster Processor as follows:

- 0 refers to Channel 1
- 1 refers to Channel 2 (8274 Channels B and A, respectively).

Note: SetCommISR and ResetCommISR are now available for backwards compatibility only. InitCommLine has been implemented as a replacement for SetCommISR. ResetCommLine has been implemented as a replacement for ResetCommISR.

DisableCluster

The DisableCluster service allows an application system on the master terminal to disable polling of the cluster terminals after a specified time period. This service is also used to resume polling of the cluster terminals. DisableCluster optionally disables all cluster lines except for that line where the request was issued.

XE520 Procedures and Services

This section describes procedures and services that have been added to BTOS for the XE520 environment. These procedures and services fall into five categories.

Return Information:

GetProcInfo returns information about the processor on which the application is running.

GetSlotInfo returns the slot numbers of the processors in a particular category.

ExpandSpec returns an expanded file specification that includes information from the caller's path.

Handle terminal input/output:

OpenTerminal prepares a terminal for I/O.

DrainTerminalOutput ensures that a terminal output buffer is empty. Usually used prior to a SetTerminal operation.

SetTerminal ascertains or changes the characteristics, both hardware and software, of a terminal.

ReadTerminal reads one or more characters from a terminal.

WhereTerminalBuffer locates a terminal buffer for output.

CloseTerminal closes an open terminal.

Route interprocessor communications:

RemoteBoot loads a designated processor with an operating system and starts the execution of the image.

RequestRemote sends a request to a particular processor.

ReadRemote receives standard read requests mapped by the ICC if remote Direct Memory Access (DMA) is required.

Route comunications:

SetCommISR establishes the Communications Interrupt Services Routines (CISRs) for the specified communications channel.

ResetCommISR purges the Communications Interrupt Services Routines (CISRs) previously established for the specified communications channel.

InitCommLine allocates a communications channel (serial port) to the user and specifies how interrupts from the channel are to be serviced.

ResetCommLine resets the comm line

Handle half–inch magnetic tape:

CloseTape removes a user's exclusive access to a specified tape drive, thereby making the tape drive available to another user. For half–inch tape, CloseTape rewinds the tape. The tape is not rewound for QIC tape.

OpenTape gives a user exclusive access to a tape drive.

PurgeTapeUser aborts all outstanding tape requests from a user and removes the user's exclusive access to all tape drives.

ReadTapeRecords reads the next n fixed length records from tape into a user buffer.

TapeStatus returns tape status information and clears outstanding error conditions.

TapeOperation sends non–data transfer commands to the tape drive.

WriteTapeRecords writes one or more fixed length records.

The remainder of this section details these procedures and services.

GetProcInfo

Description

The GetProcInfo service fills a caller-supplied array with the name of the processor on which the caller is running. The format of the processor name is ttnn.

tt is the board type-one of the following strings: FP, CP, SP, DP, or TP.

nn is the two-digit relative slot number (in decimal) of the board.

Procedural Interface

GetProcInfo (pMySlotRet, pTypeRet, pNumberRet, pbNameRet, cbNameMax, pbNameLenRet): ercType

pMySlotRet is a pointer to a byte where the slot number of this board is placed. Refer to section 2 for a further description of how slot numbers are sequenced.

pbTypeRet is a pointer to a byte where the processor type is placed. Processor types conform to the hardware type codes that are used in the Processor Type field of the CDT.

pbNumberRet is a pointer to a byte where the ordinal number of the processor is placed. To determine a processor's ordinal number, view the base enclosure from the rear and count processors of the same type from left to right (the slot numbers decrease). If the system has a second enclosure (an expansion enclosure to the left of the main enclosure), continue counting processors from left to right. Continue the count through any further enclosures in the same manner.

pbNameRet and **cbNameMax** is a (pointer, length) pair that defines the location and size of the array to receive the name of the processor.

pbNameLenRet is a pointer to a byte where the length of the name of the processor is placed.

pbType and **pbNumber** point to the binary equivalents of the characters pointed to by **pbNameRet** and **cbNameMax**.

GetProcInfo Examples

If the GetProcInfo service is called in the second Cluster Processor, the results would be:

MySlotRet - hex 75 (an arbitrary value for this example)
TypeRet - hex 011
NumberRet - hex 01
NameRet - CP01

If this routine is called in the third File Processor, the results would be:

MySlotRet - hex 51 (an arbitrary value for this example)
TypeRet - hex 010
NumberRet - hex 02
NameRet - FP02

GetSlotInfo

Description

GetSlotInfo determines the slot numbers of other processors in the system.

The GetSlotInfo primitive fills a caller–supplied array with the slot numbers of all of the processors of a specified type, and it returns the number of processors of the specified type. GetSlotInfo may be used to find the slot number of the master File Processor.

If the caller supplies an array too small to hold all of the slot numbers, as many entries as possible are placed in the array, then status code 211 is returned. This status code tells you that the buffer size is invalid. The number of processors is correctly returned.

Procedural Interface

GetSlotInfo (bCode, pbSlotArrayRet, cbSlotArrayRet, psProcessorCountRet): ercType

where bCode takes on one of the following values:

bCode	Processor Type
1	File Processor
2	Terminal Processor
3	Cluster Processor
4	Application Processor
5	Storage Processor (not part of a DP)
6	Disk Processor
7	Master File Processor

Other values of bCode are reserved for future expansion.

pbSlotArrayRet and **cbSlotArrayRet** is a (pointer, length) pair that defines the location and size of the area to receive the slot numbers of the processors specified by bCode.

psProcessorCountRet is a pointer to a word in which the number of processors of the type specified in bCode will be placed.

Request Block

GetSlotInfo is a system common procedure with the following error codes:

211 – ercBadBufferSize

154 – ercNoSuchCode

ExpandSpec

The ExpandSpec service expands an incomplete file specification to a full specification that contains the node, volume, directory, file and password, if required, of the caller. ExpandSpec accepts the incomplete specification values from pbFilespec and pbPassword. It completes the specification by obtaining information from the caller's default path.

ExpandSpec can only expand specifications for users at the workstation where the request was issued. The returned specification is in canonical form. For example, the volume name returned will always contain the name assigned to the volume when it was initialized rather than [Sys] or [Scr]. For unmounted or uninitialized volumes, the volume name field will be returned as the device name. In the case of the <$> directories, the expanded directory name will always be returned.

Procedural Interface

ExpandSpec (pbFileSpec, cbFileSpec, pbPassword, cbPassword, pSpecRet, sSpecMax, specType): ercType

pbFilespec and **cbFilespec** describe the partial specification to be expanded.

pbPassword and **cbPassword** describe the password, if any.

pSpecRet and **sSpecMax** describe the expanded file specification format returned.

specType indicates whether the specification is a volume, directory, or file type with 1 being the volume, 2 being the directory, and 3 being the file.

Request Block

Offset	Field	Size	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	253
12	specType	2	
14	reserved	2	
18	pbFileSpec	4	
22	cbFileSpec	2	
24	pbPassword	4	
28	cbPassword	2	
32	pSpecRet	4	
36	sSpecMax	2	

OpenTerminal

Description

This request initiates the use of a specified port on either a CP or TP. The returned output buffer address is in linear format and is used for subsequent input/output operations.

On an RS232–C serial terminal, this request resets the terminal characteristics to the initial state specified in the configurations file and asserts DTR.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3012
12	bDestCPU	1	
13	bSourceCPU	1	
14	bPort	1	
15	bWindow	1	
16	ppOutputBufferRet	4	
20	spOutputBufferRet	2	

bDestCpu is the slot number of the CP or TP to which the terminal is connected.

bSourceCpu is the slot number of the client.

bPort is the physical port number of an RS232–C serial terminal.

bWindow is the window number, which must be zero for RS232-C serial terminals.

ppOutputBufferRet and **spOutputBufferRet** is the output buffer for the terminal specified by bDestCpu. This returned pointer is in linear format. (Refer to section 2 for a description of linear format and terminal output buffers.)

DrainTerminalOutput

Description

This request is issued by the client to ensure an empty output buffer. DrainTerminalOutput only returns after all output has been flushed. This request is commonly used prior to a SetTerminal request.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3012
12	bDestCPU	1	
13	bSourceCPU	1	
14	bPort	1	
15	bWindow	1	

bDestCpu is the slot number of the CP or TP that is to handle the request.

bSourceCpu is the client slot number.

bPort refers to the physical port number of an RS232–C serial terminal or the logical port number of a PT 1500.

bWindow is the number of the referenced window of the PT 1500 (must be zero for RS232–C serial terminals).

SetTerminal

Description

This request is used by the client to perform out of band functions on the port, such as changing the baud rate of the port, turning on XON/XOFF recognition, and sending break requests.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	1
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3016
12	bDestCPU	1	
13	bSourceCPU	1	
14	bPort	1	
15	bWindow		
16	bFunctionCode	1	
17	bDummy	1	(for alignment)
18	pParamBlock	4	2
22	sParamBlock	2	
24	sParamReg	4	
28	sParamRet	2	

bDestCpu is the slot number of the CP or TP connected to an RS232-C serial terminal connected directly to a port.

bSourceCpu is the slot number of the client.

bPort is the physical port number of an RS232-C serial terminal.

bWindow is the number of the window, which must be zero for RS232-C serial terminals.

bFunctionCode takes on one of the following values:

Value	Function
0	Read parameter Block. Reads the parameter block described in Table 5-1.
1	Write Parameter Block. Writes the parameter block.
2	Flush Input Buffer. Empties the input buffer.
3	Flush Output Buffer. Empties the output buffer.
4	Send Break. Sends a 1/4 second break.
5	Suspend output. Stops output.
6	Resume Output. Resumes output that has been suspended.

Table 5–1 Parameter Block Format

0	bBaud	1	A value from the baud rate in Table 5–2.
1	bStopBits	1	0, 1, 2, but 0 – 1.5 stop bits.
2	bParity	1	0, 1, 2 – odd, even, none.
3	bCharBits	1	5, 6, 7 or 8 data bits.
4	fFlowGen	1	If true, then send XOFF/XON.
5	fFlowAct	1	If true, then accept XOFF/XON sequences from the terminal.
6	fFlowany	1	If the FlowAct field is true and this field is true, then any character can serve as an XON.
7	fLeadinEnable	1	If true, then every <01> character in the input is followed by a byte that gives a delay time in 20–msec units.
8	fNoHangOnClose	1	If true, then the line will not be placed on–hook when a CloseTerminal request is received.
9	bOwner	1	Slot number of owner of port. Read only.

Table 5–2 **Baud Rates**

bBaud	Rate	bBaud	Rate
1	50	8	600
2	75	9	1200
3	110	10	1800
4	134.5	11	2400
5	150	12	4800
6	200	13	9600
7	300	14	19200

ReadTerminal

Description

This request is used by the client process to read data from one of the asynchronous ports on a CP or TP.

The user specifies a buffer size and a timeout value. ReadTerminal returns when one of the following three conditions occurs:

- The number of requested characters is received.
- The timeout has elapsed. The timeout begins after the first character is received.
- An error is detected.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3015
12	bDestCPU	1	
13	bSourceCPU	1	
14	bPort	1	
15	bWindow	1	
16	bFunctionCode	1	
17	bDummy	1	
18	psCountRet	2	2
22	ssCountRet	2	
24	pDataRet	4	
28	sDataMax	2	

bDestCpu is the slot number of the CP or TP connected to an RS232–C serial terminal connected directly to a port.

bSourceCpu is the slot number of the client.

bPort is the physical port number of an RS232–C serial terminal or the logical port number of a PT 1500.

bWindow is the number of the referenced window, which must be zero for RS232–C serial terminals.

bTime is the number of 20–msec time units to wait for the request to be satisfied after the first character is received. This allows a large amount of data to be received, particularly if the data rate is high. If there is little data, the timeout causes a partially filled buffer to be returned.

bDummy is a filler byte that ensures word alignment of the rest of the request block.

psCountRet and **ssCountRet** describe the area to receive the count of the number of bytes actually read. The returned count will be in linear format (byte swapped from normal 80186 usage.)

pDataRet and **sDataMax** describe the area to receive the output data.

WhereTerminalBuffer

Description

This request is used by the client to locate the output buffer. The returned address of the output buffer is in linear format.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3014
12	bDestCPU	1	
13	bSourceCPU	1	
14	bPort	1	
15	bWindow	1	
16	ppOutputBufferRet	4	
20	spOutputBufferRet	2	4

bDestCpu is the slot number of the CP or TP to which the terminal is connected.

bSourceCpu is the slot number of the client.

bPort is the physical port number of an RS232–C serial terminal.

bWindow is the number of the referenced window, which must be zero for RS232–C serial terminals.

ppOutputBufferRet and **spOutputBufferRet** are the output buffers for the terminal specified by bDestCpu. This returned pointer is in linear format. (Refer to section 2 for a description of linear format and terminal output buffers.)

CloseTerminal

Description

This request is sent by a client to indicate that it is finished with a port. It causes any pending ReadTerminal request to be returned with an error.

If the line is connected to a data set and NoHangUpOnClose has not been selected, it is placed back on-hook in anticipation of a new call. If NoHangUpOnClose has been selected, DTR is not deasserted.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3013
12	bDestCPU	1	
13	bSourceCPU	1	
14	bPort	1	
15	bWindow	1	

bDestCpu is the slot number of the CP or TP to which the terminal is connected.

bSourceCpu is the slot number of the client.

bPort is the physical port number of an RS232-C serial terminal.

bWindow is the number of the referenced window, which must be zero for RS232-C serial terminals.

RemoteBoot

Description

This request, issued by the master FP only, causes another processor to be bootstrapped with a specified operating system. The specified target processor must have passed boot ROM checks and be ready to boot. There must be sufficient memory available (according to the remote processor CDT) to hold the designated operating system.

The target processor's CDT is updated with current configuration data. A doorbell interrupt initiates the execution of the operating system. If the target processor does not set the CDT flag fOsInitialized with 30 seconds after the doorbell interrupt, then status code 153 is returned. This status code tells you that the target processor did not respond to the remote boot.

Procedural Interface

This is a level 3 request and, as such, has no procedural interface.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	4
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	hex 3002
12	fh	2	
14	bSlot	1	
15	fReserved	1	0

fh is the file handle of the operating system to be loaded into the designated processor. This operating system must have a valid BTOS.RUN file header. The loaded system must contain a valid linear CDT pointer in location hex 1F8.

bSlot is the slot number of the target processor.

fReserved must be set to zero.

RequestRemote

Description

The RequestRemote primitive is issued by a process on one processor to request a system service from a process on another processor. The request bypasses the ICC server agent on the client process's processor. It is sent directly to the ICC server agent on the remote processor, which infers from the request block the appropriate service exchange and sends the request there.

The ICC routes the response, without any server intervention, from the remote process to the response exchange on the client process, which is also specified in the request block.

A client process may use the RequestRemote primitive directly when it is necessary to bypass the normal request routing mechanisms of the operating system.

It is the responsibility of the client process to ensure that the specified remote processor actually exists.

Procedural Interface

RequestRemote (bRemoteCpuId, pRq): ercType

bRemoteCpuId is the slot number of the remote processor.

pRq is the local memory address of the request block.

Request Block

RequestRemote is a kernel primitive.

ReadRemote

Description

The ReadRemote operation receives standard read requests mapped by the ICC if DMA is required.

Procedural Interface

ReadRemote has no procedural interface. You must make a request block and issue the request using the Request, RequestDirect, or RequestRemote operations. For details on these operations, refer to the *BTOS II System Procedural Interface Manual.*

Request Block

Offset	Field	Size	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	2
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	12325
12	fh	2	
14	lfa	4	
18	pBufferRet	4	
22	sBufferMax	2	
24	psDataRet	4	
28	ssDataRet	2	2

SetCommISR

Description

The SetCommISR service establishes the Communications Interrupt Services Routines (CISRs) for the specified communications channel. Separate CISRs are established to process transmit, external/status, receive, and special receive conditions.

Note: SetCommISR is now available for backwards compatibility only. InitCommLine has been implemented as a replacement for SetCommISR.

Procedural Interface

SetCommISR (iLine, pDS, pTxIsr, pExtIsr, pRxIsr, pSpRxIsr): ercType

iLine is SIO Channel A if it is 0 and SIO Channel B if it is 1.

pDS is the memory address of any byte in the memory segment to be used as the data segment of the CISRs. The segment base address part of pDS is to be used as the data segment base (that is, loaded into the DS register) when any of the four CISRs is activated.

pTxIsr is the memory address (CS:IP) of the CISR that is to process Transmit-Data-Buffer-Empty interrupts.

pExtIsr is the memory address (CS:IP) of the CISR that is to process External/Status interrupts.

pRxI is the memory address (CS:IP) of the CISR that is to process Receive-Character-Available interrupts.

pSpRxIsr is the memory address (CS:IP) of the CISR that is to process Receive-Special-Condition interrupts.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	1	22
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	101
12	iLine	2	
14	pDS	4	
18	pTxIsr	4	
22	pExtIsr	4	
26	pRxIsr	4	
30	pSpRxIsr	4	

ResetCommISR

Description

The ResetCommISR service purges the Communications Interrupt Services Routines (CISRs) previously established for the specified communications channel. Future interrupts from the specified channel are ignored.

Note: ResetCommISR is now available for backwards compatibility only. ResetCommLine has been implemented as a replacement for ResetCommISR.

Procedural Interface

ResetCommISR (iLine): ercType

iLine is SIO Channel A if iLine is 0 and SIO, Channel B if iLine is 1.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	102
12	iLine	2	

InitCommLine

Description

InitCommLine allocates a communications channel (serial port) to the user and specifies how interrupts from the channel are to be serviced.

InitCommLine performs only partial initialization of the 8274–type controller. InitCommLine resets the designated communications channel before responding. This allows time for chip initialization before interrupts can occur, and it means InitCommLine can and should be called with interrupts enabled. Your program is responsible for all other 8274–type initialization.

You must specify Status Affects Vector by setting bit 2 (04h) in Write Register 1 during channel initialization. If you fail to set this bit, input/output can be disrupted in progress on another channel, as well as preventing interrupts from being vectored appropriately to your channel.

For the B24 workstation, InitCommLine resets the designated Comm channel before responding to the user. On the B24 workstation ports A and B, the user must specify STATUS AFFECTS VECTOR by setting bit 2 (04h) in Write Register 1 during channel initialization. Failure to set this bit may disrupt I/O in progress on another channel, and prevent interrupts from being vectored appropriately to your channel.

The channel is unavailable to other users until ResetCommLine is issued. Task termination does this automatically if you fail to issue ResetCommLine.

If you are using a Comm line interface in a modemless synchronous configuration, you must divide the baud rate by a factor of 16 when using InitCommLine.

Procedural Interface

InitCommLine (pbSpec, cbSpec, pbClcb, cbClcb, pbRet, cbRet): ercType

pbSpec is the memory address of the device specification string.

cbSpec is the length of the string (word).

pbClcb is the memory address of the Communications Line Configuration Block (CLCB).

cbClcb is the size in bytes of the CLCB.

pbRet is the memory address of an area into which the system writes return values.

cbRet is the size in bytes of the return area (normally 19).

Request Block

Offset	Field	Size	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	2
3	nRespPbCb	1	1
4	userNum	4	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	1006h
12	reserved	6	
18	pbSpect	4	
20	cbSpec	2	
24	pbClcb	4	
28	cbClcbq	2	
30	pbRet	4	
34	cbRet	2	

ResetCommLine

Description

ResetCommLine makes the specified communications channel available for use again. No further interrupts are routed to the user who executed the InitCommLine. If an interrupt is received after ResetCommLine is executed but before another InitCommLine, the MPSC or 2681 (on the B24 workstation) channel is reset. (MPSC is a multi-protocol serial controller.)

Procedural Interface

ResetCommLine (commLineHandle):ercType

Request Block

Offset	Field	Size	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	1007h
12	CommLineHandle	2	
14	reserved	4	

OpenTape

Description

OpenTape gives a particular user exclusive access to the magnetic tape drive. It returns a tape handle to be used on subsequent tape requests. OpenTape does not position the tape.

Procedural Interface

OpenTape (pbTapeName, cbTapeName, fStreaming, fHighDensity, fclear, mode, pThRet): ercType

pbTapeName and **cbTapeName** specify the memory area of tape to open. Tape names take the form [tapen] for half-inch tape drives where n is a tape number and [qic] for a quarter-inch cartridge (QIC) tape drive. Any characters following the tape number are ignored by this request.

fStreaming is a flag indicating whether data transfer is to be performed with the transport at high speed. The flag is significant only for tape drives with high-speed streaming mode; it is ignored for other drives.

fHighDensity is a flag indicating if high density is selected for tape operation. The flag is significant only for remote-selectable dual-density drives. At this time only 1600 bpi is supported.

fClear is a flag indicating whether, after a data error, a TapeStatus request must be issued to clear the error condition before sending another request.

mode is a word indicating read (mr) or write (mw) as the mode. If a tape without a write-ring is opened for write, an error is returned.

pThRet is the memory address of the word to which the tape handle is returned.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	1
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	190
12	fstreaming	1	
13	fhighdensity	1	
14	fclear	1	
15	reserved	1	
16	mode	2	
18	pbTapeName	4	
22	cbTapeName	2	
24	pthRet	4	
28	cbThRet	2	2

TapeStatus

Description

TapeStatus allows users to determine the status of a tape drive. The TapeStatus request also clears outstanding error conditions.

The tape status returned is one or a sum of the following bit values which are listed with their meanings. For example, if a tape were write-protected, ready, and on-line, the status would be hex 4A (hex 2 + hex 8 + hex 40).

Status Returned (hex)	Meaning
2	The formatter is currently write-protected.
4	The formatter is busy and is not ready for commands.
8	The tape drive is currently ready for new commands.
10	The end of the tape has been reached.
20	The tape is at the load point.
40	The tape drive is on-line.

Procedural Interface

TapeStatus (th, pStatusRet): ercTape

th is the tape handle returned by OpenTape.

pStatusRet is the memory address of the word in memory to which tape status is returned.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	196
12	th	2	
18	reserved	4	
22	pStatusRet	4	
26	pStatusRet	2	2

TapeOperation

Description

The TapeOperation service allows the user to issue to the tape drive such non–data–transfer commands as rewind tape, erase, and skip records.

In the TapeOperation procedural interface, the command parameter has one argument, the subop parameter, which some commands use and others do not. However, even if the subop parameter is not used, it must be present in the TapeOperation procedure call; TapeOperation will not work without it. 0 (zero) is a safe value to put into the subop parameter for commands that do not use this parameter.

In the command parameter list, for those commands which use the subop parameter, n represents the number of times the command is performed. If n is positive, the command is performed with the tape moving forward. If n is negative, the command is performed with the tape moving backward. For commands that use the supop parameter, n cannot be zero.

Procedural Interface

TapeOperation (th, fFast, command, subOp): ercType

th is the tape handle returned by OpenTape.

fFast is a flag indicating operations are performed in high–speed mode. This flag overrides the parameter given in the OpenTape command, allowing the user to selectively skip single records at low speeds, skip long files at high speeds, and erase short lengths of tape.

command is the command code (not the command name) for the operation to be performed. (Refer to the Command Parameter list).

subop is the argument to be supplied to certain commands. (Refer to the Command Parameter list.)

Command Parameter

Command Name	Code,Subop	Description
Rewind	(1,0)	Rewinds the tape to the load point. This procedure call does not return until the rewind is complete (tape is at beginning of tape [BOT]). During the rewind, the tape server can process other users' requests for operations on other tape drives.
Unload	(2,0)	Rewinds and unloads the tape. This procedure call does not return until the rewind is completed and the unload begins.
SkipRecord	(3,n)	Skips n records
SearchMark	(4,n)	Searches for n consecutive tape marks.
WriteTapeMark	(5,n)	Writes n consecutive tape marks. The n parameter must be positive.
Erase	(6,n)	Erases n*3.5 (n times 3.5) inches of tape. n must be a positive number.
EraseTape	(7,0)	Erases the tape until end of tape (EOT) is reached.

Command Name	Code, Subop	Description
RewindAsync	(8,0)	Rewinds the tape to the load point. This procedure returns immediately after the command is issued. Hence, you can overlap rewind with the application task.
UnloadAsync	(9,0)	Rewinds and unloads the tape. This procedure call returns immediately after the command is issued.

Below are two examples of the TapeOperation procedure. The first includes a command with the subop parameter and the second includes a command without it.

TapeOperation (th, FALSE, 3, –1)

backspaces the tape one record.

TapeOperation (th, TRUE, 1, 0)

rewinds the tape to the load point.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	193
12	th	2	
14	fFast	1	
15	command	1	
16	subop	2	

ReadTapeRecords

Description

ReadTapeRecords reads the next n fixed–length records from the tape into a user buffer. The buffer must be aligned to a word boundary, and its size must not exceed the buffer size specified during installation of the tape server.

Any number of records can be read, as long as they collectively do not exceed the user buffer size. The nRecords parameter must be greater than zero. A read in the reverse direction is not allowed.

If an error condition occurs, nRecordsRet reflects the number of records successfully read. CbRet is updated with the number of bytes read.

If an error occurs and the fClear flag was set when the tape was opened, then issuing a TapeStatus request allows the user to try to read the tape again. If another ReadTapeRecords is issued before a TapeStatus request, an ercTapeErrorOutstanding status code results, and the user is not allowed to read the tape until TapeStatus is issued.

This command cannot detect an overrun on read, that is, reading a block that is larger than the memory buffer. An overrun on read, therefore, does not cause an ercTapeTruncated error status. It is not a fatal error.

Procedural Interface

ReadTapeRecords (th, nRecords, pb, cb, pNRecordsRet, PCbRet): ercType

th is the tape handle returned by OpenTape

nRecords is the number of records to read.

pb is the memory address in the user area to store the tape records read.

cb is the buffer size in bytes of the user area.

pCRecordsRet is the memory address to store the number of records that were successfully read.

pCbRet is the memory address to store the number of bytes transferred to the user buffer.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	0
3	nRespPbCb	1	3
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	191
12	th	2	
14	nRecords	2	
16	reserved	2	
18	pb	4	
22	cb	2	
24	pCRecordsRet	4	
28	cbcRecordsRet	1	2
30	pCbRet	4	
34	cbcbRet	1	2

WriteTapeRecords

Description

WriteTapeRecords writes one or several fixed length records. The nRecords parameter must be greater than zero.

Writing in the reverse direction is not allowed.

The number of records written and the count of bytes written are returned, even if an error stops the writing.

Buffer alignment and size restrictions are the same as for ReadTapeRecords.

Procedural Interface

WriteTapeRecords (th, nRecords, pb, cb, pCRecordsRet, pCbRet): ercType

th is the tape handle returned by OpenTape.

nRecords is the number of records to write.

pb is the memory address of the bytes to write on tape.

cb is the number of bytes to write. The size of each record is computed as cb/n records.

pCRecordsRet is the memory address at which the number of records successfully written is stored.

pCbRet is the memory address at which the number of bytes written to tape is stored.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	6
2	nReqPbCb	1	1
3	nRespPbCb	1	2
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	192
12	th	2	
14	nRecords	2	
16	reserved	2	
18	pb	4	
22	cb	2	2
24	pCRecordsRet	4	
28	cbCRecordsRet	1	2
30	pCbRet	4	
34	cbcbRet	1	2

PurgeTapeUser

Description

PurgeTapeUser aborts all the user's outstanding tape requests and releases all tape drives held exclusively by the user.

PurgeTapeUser, an asynchronous operation, does not wait for the drive to reset its current operation. A drive performing a rewind operation cannot be stopped by PurgeTapeUser.

Procedural Interface

PurgeTapeUser (work): ercType

work is a temporary 2-byte area for the service.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	195
12	work	2	

CloseTape

Description

CloseTape removes a user's exclusive access to the tape drive, making the drive available to other users. The tape position is not affected.

Procedural Interface

CloseTape (th): ercType

th is the tape handle returned by OpenTape.

Request Block

Offset	Field	Size	Contents
0	sCntInfo	2	2
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	194
12	th	2	

Glossary

$ Directories. The $ directories are special directories required for the BTOS software to operate correctly. When a request with the directory <$> is given as a part of a file specification to BTOS, the directory name is expanded to the form <$nnn>, where nnn is the user number of the application system.

Allocation Bit Map. The Allocation Bit Map controls the assignment of disk sectors. It has 1 bit for every sector on the disk and the bit is set if the sector is available. The Allocation Bit Map is disk-resident.

AllocExch. AllocExch is a BTOS system procedural interface call that allocates exchanges. On the XE520, it allows direct interboard message passing on XE520 processor boards.

Application Process. An application process executes code in the application system. It is not a system service process. Also see System Service Process.

Application System. An application system is the collection of all tasks currently in an application partition. The tasks in an application system access a common set of files and implement a single application. The tasks execute asynchronously. Also see Task and Application Partition.

Application System Control Block (ASCB). The Application System Control Block (ASCB) communicates parameters, the termination code, and other information between an exiting application system and a succeeding application system in the same partition. Also see the Variable-Length Parameter Block.

ASCB. See Application System Control Block.

Bad Sector File. The Bad Sector File contains an entry for each unusable sector of a disk. The size of the Bad Sector File is 1 sector.

Bootstrap. To bootstrap (or boot) the system is to start it by reloading BTOS from disk. On other systems, this may be referred to as Initial Program Load (IPL).

BSWA. See Byte Stream Work Area.

BX. Base general register.

Byte Stream. A byte stream, a concept of the Sequential Access Method, is a readable (input) or writable (output) sequence of 8–bit bytes. An input byte stream can be read until either the reader chooses to stop reading or it receives status code 1 **(End of File)**. An output byte stream can be written until the writer chooses to stop writing. Also see Byte Stream Work Area, Communications Byte Stream, Sequential Access Method, and Spooler Byte Stream.

Byte Stream Work Area. The Byte Stream Work Area is a 130–byte memory work area for the exclusive use of Sequential Access Method procedures. Any number of byte streams can be open concurrently, using separate Byte Stream Work Areas.

cb. A cb is the count of bytes in a string of bytes.

Client Process. A client process is a process that makes a request of a system service. Any process, even a BTOS process, can be a client process since any process can request system services.

Cluster Processor. A Cluster Proccessor is a board in the XE520 system that runs communications software and supports workstations, a parallel printer, and up to three RS232–C serial devices.

Cluster Workstation. A cluster workstation is a workstation in a cluster configuration connected to a master workstation. B24, B26, B27, and B28 workstations can serve as cluster workstations.

Communication Byte Stream. A communications byte stream is a byte stream that uses a communications channel.

Communications Interrupt Service Routine. A Communications Interrupt Service Routine (CISR) is similar to a mediated interrupt handler except that a CISR serves only one of the two communications channel of the SIO communications controller.

Compact System. A compact system is a version of BTOS that provides all operating system functions except the concurrent execution of multiple applications systems. A compact system has a primary application partition and can execute application systems one at a time. The operating system is specified as compact during system build.

Configuration File. A configuration file specifies the characteristics of either the parallel printer, the serial printer, or any other device attached to a communications channel. Examples of characteristics are number of characters per line, baud rate, and line control mode (XON/XOFF, CTS). A configuration file is created by the Create Configuration File command and is used by printer, spooler, and communications byte stream.

Contingency. A contingency can refer to a variety of hardware and software conditions that have undesirable effects. These conditions can be hardware faults such as a memory parity error, inconsistencies detected by BTOS such as a bad checksum of a Volume Home Block, or conditions detected by the application system. BTOS always terminates execution when it detects an inconsistency.

CP. See Cluster Processor.

CPU. The CPU (central processing unit) is the microprocessor.

Crash Dump Area. The Crash Dump Area (the file [Sys]<Sys>CrashDump.Sys) contains a binary memory dump in the event of a system failure.

CWS. See Cluster Workstation.

DAM. See Direct Access Method.

Data Segment. A data segment contains data. It can also contain code, although this is not recommended. If a data segment is shared among several processes, concurrency control is the responsibility of those processes. A data segment that is automatically loaded into memory along with its task image is called a status data segment. A dynamic data segment is allocated by a request from the executing process to the memory management facility.

Date/Time Format. The date/time format provides a compact representation of the date and time of day that precludes invalid dates and allows simple subtraction to compute the interval between two dates. The date/time format is represented in 32 bits to an accuracy of one second.

DAWA. See Direct Access Work Area.

DCB. See Device Control Block.

Default Response Exchange. Each process is given a unique default response exchange when it is created. This special exchange is automatically used as the response exchange whenever a client process uses the procedural interface. For this reason, the direct use of the default response exchange is not recommended. The use of the default response exchange is limited to requests of a synchronous nature. That is, the client process, after specifying the exchange in a Request, must wait for a response before specifying it again (indirectly or directly) in another Request. Also see Exchange and Response Exchange.

Device. A device is a physical hardware entity. Printers, tapes, floppy disks, and Winchester disks are examples of devices.

Device Control Block. There is a Device Control Block (DCB) for each physical device. The DCB, generated at system build, contains information about the device. The kind of information for a disk, for example, includes how many tracks are on a disk and the number of sectors per track. The DCB points to a chain of I/O Blocks. The DCB is memory–resident.

Direct Access Method. The Direct Access Method (DAM) provides random access to disk file records identified by record number. The record number is specified when the DAM file is created. DAM supports COBOL Relative I–O, but can also be called directly from any of the Unisys languages. Also see Direct Access Work Area.

Direct Access Work Area. A Direct Area Work Area (DAWA) is a 64–byte memory work area for the exclusive use of the Direct Access Method (DAM) procedures. Any number of DAM files can be open simultaneously using separate DAWAs. Also see Direct Access Method.

Directory. A directory is a collection of related documents, programs, or other data store on a volume. A directory is protected by a directory password.

Directory Password. A directory password protects a directory on a volume.

Directory Specification. A directory specifications consists of a node (node name), volname (volume name), and a dirname (directory name).

Dirname (Directory name). A dirname is the third element of a directory specification or a full file specification.

Disk Processor. A Disk Processor is an XE520 processor board that is formed by connecting an SC board to an SP board. The Disk Processor supports I/O to half–inch magnetic tape drives and SMD disks.

DP. See Disk Processor.

DS. Data Segment

Erc. An Erc is a error code.

Error message. An error message is a message that appears on the screen when an error occurs in the file management system or a subsystem.

Exchange. An exchange is the path over which messages are communicated from process to process (or from interrupt handler to process). An exchange consists of two first–in, first–out queues, one of processes waiting for messages, the other of messages for which no process has yet waited. An exchange is referred to by a unique 16–bit integer. Also see Default Response Exchange and Response Exchange.

Executive. The Executive is the BTOS user interface program; it provides many convenient utilities for file management.

Exit Run File. An exit run file is a user–specified file that is loaded and activated when an application system exits. Each application partition has its own exit run file.

ExpandSpec. ExpandSpec is an XEBTOS procedural interface call that returns an expanded file specification so that it includes information from the caller's path.

FALSE. FALSE is represented in a flag variable as 0.

FCB. See File Control Block.

fh. File handle.

FIFO. First in, first out.

File. A file is a document, program, or other set of related data stored as a unit in a directory on a single volume.

File Access Methods. Several file access methods augment the capabilities of the file management system. The file access methods are object module procedures that are located in the standard BTOS library and are linked to application systems as required. They provide buffering and use the asynchronous input/output capabilities of the file management system to automatically overlap input/output and computation. Also see Direct Access Method, Record Sequential Access Method, and Sequential Access Method.

File Area Block. There is a File Area Block (FAB) for each Disk Extent in an open file. The FAB specifies where the sectors are and how many there are in the Disk Extent. The FAB is pointed to by a File Control Block or another FAB. The FAB is memory–resident.

FCB. See File Control Block.

File Control Block. There is a File Control Block (FCB) for each open file. The FCB contains information about the file such as the device on which it is located, the user count (that is, how many file handles currently refer to this file), and the file mode (read or modify). The FCB is pointed to by a User Control Block and contains a pointer to a chain of File Area Blocks. The FCB is memory–resident.

File Header Block. There is a File Header Block (FHB) for each file. The FHB of each file contains information about that file such as its name, password, protection level, the date/time it was created, the disk address and size of each of its Disk Extents. The FHB is disk–resident and is 1 sector in size.

Filename (File name). A filename is the fourth element of a full file specification.

File Password. A file password protects a file in a directory on a volume.

File Processor. A File Processor is an XE520 processor board that supports I/O operations to disk devices.

File Protection Level. A file protection level specifies the access allowed to a file when the accessing process does not present a valid volume or directory password.

FP. See File Processor.

Frame. A frame is a separate, rectangular area of the screen. A frame can have any desired width and height (up to the dimensions of the screen.

Full File Specification. A full file specification consists of a node (node name), volname (volume name), dirname (directory name), and filename (file name).

GetUserNumber. GetUserNumber is a BTOS procedural interface call that returns the user number. XEBTOS uses a modified version of the call.

Indexed Sequential Access Method. The Indexed Sequential Access Method (ISAM) provides efficient and flexible, random access to fixed–length records identified by multiple keys stored in disk files.

Internal Interrupt. An internal interrupt (often called a trap) is caused by and is synchronous with the execution of processor instructions. The causes of internal interrupts are an erroneous divide instruction, the Trap Flag, the INTO (interrupt on overflow) instruction, and the INT (interrupt) instruction.

Interrupt. An interrupt (external or internal) is an event that interrupts the sequential execution of processor instructions. When an interrupt occurs, the current hardware context (the state of the hardware registers) is saved. This context save is performed partly by BTOS.

Interrupt Handler. An interrupt handler is a locus of computation that is given control when an interrupt occurs. Since an interrupt handler is not a process, it is permitted to invoke only few specific operations. BTOS interrupt handlers are provided for each interrupt type. Each interrupt handler services two kinds of interrupt handlers: mediated and raw. .

IOB. See I/O Block.

I/O Block. The I/O Block (IOB) is used by BTOS as temporary storage during **Read**, **Write**, and other input/output operations. The IOB contains information obtained from the request block. The number of IOBs specified at system build must be adequate for the maximum number of input/output operations that will be in progress simultaneously. The IOB is memory–resident.

IPC. Interprocess Communication.

ISAM. See Indexed Sequential Access Method.

Kernel. The kernel is the most primitive and the most powerful component of BTOS. It executes with higher priority than any process but it is not itself a process. The kernel schedules process execution; it also provides interprocess communication primitives.

Linker. The Linker links one or more object files into a task image stored in a run file.

Log File. The Log File (the file [Sys]<Sys>Log.Sys) is an error-logging file. An entry is placed in the Log File for each recoverable and nonrecoverable device error.

Logical Memory Address. (Usually abbreviated as memory address). A logical memory address is a 32-bit entity that consists of a 16-bit offset. The physical memory address of a byte is computed by multiplying the segment base address by 16 and adding the offset. A byte of memory does not have a unique logical memory address. Rather, any of 4096 combinations of segment base address and offset refer to the same byte of memory.

Master File Directory. There is an entry for each directory on the volume in the Master File Directory (MFD), including the Sys Directory. The position of an entry within the MFD is determined by randomization (hashing) techniques. The entry contains the directory's name, password, location, and size. The Master File Directory is disk-resident.

Master Workstation. A master workstation is the hub of a cluster configuration. It provides a file system, queue management facility, and other services to all the cluster workstations. In addition, if supports its own interactive and batch application systems. Also see Cluster Workstation.

Master Workstation Agent Service Process. The Master Workstation Agent Service Process reconverts an interstation message to an interprocess request and queues it at the exchange of the master workstation system service process that performs the desired function. Also see Cluster Workstation Agent Service Process.

Mediated Interrupt Handler. A mediated interrupt handler (MIH) is easier to write than a raw interrupt handler; permits automatic nesting by priority since processor interrupts are enabled during its execution; and can communicate its results to processes through certain kernel primitives. Also see Interrupt Handler.

Memory Address. See Logical Memory Address.

Message. A message is the entity transmitted between processes by the interprocess communication facility. It conveys information and provides synchronization between processes. Although only a single 4–byte data item is literally communicated between processes, this data item is usually the memory address of a larger data structure. The larger data structure is called the message, while the 4–byte data item is conventionally called the address message. The message can be in any part of memory that is under the control of the sending process. By convention, control of the memory that contains the message is passed along with the message.

MFD. See Master File Directory.

MIH. See Mediated Interrupt Handler.

Mode peek. Mode peek is one of several mode parameters used in BTOS procedural interface calls. Mode peek allows you to read a file without preventing a different user from modifying the file.

Multiprogramming. Multiprogramming is supported at three levels by BTOS. First, any number of application systems can coexist, each in its own partition. Second, any number of tasks can be loaded into the memory of the partition and be independently executed. Third, any number of processes can independently execute the code of each task. Also see Application System, Process, and Task.

Object Module Procedure. An object module procedure is a procedure supplied as part of an object module file. It is linked with the user–written object modules of an application system and is not supplied as part of the System Image. Most application systems require only a subset of these procedures. When the application system is linked, the desired procedures are linked together in the run file of the application. The sequential Access Method is an example of object module procedures. Also see System Common Procedure.

Offset. The offset is the distance, in bytes, of the target location from the beginning of the hardware segment. Also see Logical Memory Address and Physical Memory Address.

Operation. An operation is a BTOS primitive, service, or procedure.

Overlapped. In the context of file access methods, overlapped means that although the application system makes a call to an access method read or write operation and that operation returns, input/output can continue simultaneously and automatically with the computations of the application system.

Partition Descriptor. A Partition Descriptor is located in each application partition and contains the partition name, the boundaries of the partition and of its long– and short–lived memory areas, and internal links to partition descriptors in other partitions.

Partition Handle. A Partition Handle is a 16–bit integer that uniquely identifies a secondary application partition. It is returned by the CreatePartition operation and is used to refer to the partition in subsequent operations such as LoadPrimaryTask, GetPartitionStatus, and RemovePartition.

pb. A pb is the memory address of a string of bytes.

pb/cb. A pb/cb is a 6–byte entity consisting of the 4–byte memory address of a byte string followed by the 2–byte count of the bytes in that byte string.

PCB. See Process Control Block.

Physical Memory Address. Each byte of memory has a unique 20–bit physical memory address. Software uses logical memory addresses, not physical memory addresses. The physical memory address of a byte is computed by multiplying the segment base address by 16 and adding the offset. Also see Logical Memory Address, Offset, and Segment Base Address.

Primary Application Partition. The primary application partition is for interactive programs that use the keyboard and video display to interact with the user. Such partitions can be loaded with interactive programs chosen by the user, such as the Editor, Word Processor, or terminal emulators.

Primitive. A primitive is an operation performed by the kernel. Also see Kernel.

Procedural Interface. A procedural interface is a convenient way to access system services and is compatible with FORTRAN, Pascal, and assembly languages.

Procedure. A procedure is a subroutine.

Process. A process is the basic entity that competes for access to the processor and which BTOS schedules for execution. Associated with a process is the address (CS:IP) of the next instruction to execute on behalf of this process; a copy of the data to be loaded into the processor registers before control is returned to this process, a default response exchange, and a stack. A process is assigned a priority when it is created so that BTOS can schedule its execution properly.

Process Context. The context of a process is the collection of all information about a process. The context has both hardware and software components. The hardware context of a process consists of values to be loaded when the process is scheduled for execution. This includes the registers that control the location of the process's stack. The software exchange and the priority at which it is to be scheduled for execution. The combined hardware and software context of a process is maintained in a system data structure called a Process Control Block. Also see Context Switch and Process Control Block.

Process Control. The context of a process is the collection of all information about a process. The context has both hardware and software components. The hardware context of a process consists of values to be loaded when the process is scheduled for execution. This includes the registers that control the location of the process's stack. The software context of a process consists of its default response exchange and the priority at which it is scheduled for execution. The combined hardware and software context of a process is maintained in a system data structure called a Process Control Block. Also see Context Switch and Process Control Block.

Process Control Block. The combined hardware and software context of a process is maintained in a system data structure called a Process Control Block. A Process Control Block is the physical representation of a process. Also see Process Context.

Processor. A processor consists of the central processing unit (CPU), memory, and associated circuitry. Also see CPU.

RAM. Random Access Memory.

Randomizing Techniques. A file entry in a directory (or a directory entry in a Master File Directory) is located by means of the character string that identifies the file (or the directory). The character string is converted to a psuedorandom number, which is then converted to the address of the sector where the entry is expected to be located. If the entry is not in the expected sector, then adjacent sectors are searched.

rDevice. rDevice (0F7h) is the value of the routing code for system services that are BNET II compatible and that are installed more than once on different XE520 processor boards.

ReadRemote. ReadRemote is an XEBTOS procedural interface call that reads one or more characters from a terminal.

Ready State. The ready state is one of three states in which a process can exist. A process is in the ready state when it is ready to run, but a higher priority process is currently running. Any number of processes can be in the ready state simultaneously. Also see Running State and Waiting State.

Record Fragment. A record fragment is a contiguous area of memory within a record. A record fragment is specified using an offset from the beginning of the record and byte count.

Record Number. A record number specifies the record position relative to the first record in a file. The record number of the first record in a file access method file is 1.

Record Sequential Access Method. The Record Sequential Access Method (RSAM) provides blocked, spanned, and overlapped input and output. An RSAM file is a sequence of fixed– or variable–length records. Files are opened for read, write, or append operations Also see Blocked, Record Sequential Work Area, and Spanned.

Record Sequential Work Area. A Record Sequential Work Area (RSWA) is a 150–byte memory work area for the exclusive use of the Record Sequential Access method procedures. Any number of RSAM files can be open simultaneously using separate RSWAs. Also see Record Sequential Access Method.

Request. A request asks a system service process to perform an operation.

Request Block. A request block is a block of memory provided by the client process that contains highly structured information. The memory address of the request block is provided by the client process during a Request primitive and by the system service process during a Respond primitive. A request block is the element that the application system (or BTOS) sends to BTOS to request that a particular operation be performed.

Request Code. A request code is a 16–bit value that uniquely identifies a system service. For example, the request code for the Write operation is 36. The request code is used both to route a request to the appropriate system service process and to specify to that process which of the several services it provides is currently requested.

Request Control Block. A Request Control Block (RCB) is an internal data structure. There is one for each concurrent request. The number of RCBs is a system build parameter.

RequestDirect. RequestDirect is a BTOS kernel primitive that allows direct interboard message passing on the XE520.

Request Level. Sixteen request levels are supported in a request table. Twelve requests levels are reserved for internal use and four are available to the user. A request code structure is composed of 16 4KB request levels, each containing up to 4KB request definitions. Half of these levels have a procedural interface and validity checking structures. Various levels are reserved for internal use.

Request Table. A request table is a table that defines the file specification and s/b routing for requests. The request table supports 16 request levels. Twelve request levels are reserved for internal use and four are available to the user. The table contains information on file specifications, routing rules, destination, exchange, local service code, and procedural information.

Response Exchange. A response exchange is the exchange at which the requesting client process waits for the response of a system service. Also see Default Response Exchange and Exchange.

ROM. Read–Only Memory.

RSAM. See Record Sequential Access Method.

RSWA. Record Sequential Work Area.

Run File. A run file is created by the linker and contains a task image.

Running State. The running state is one of three states in which a process can exist. A process is in the running state when the processor is actually executing its instructions. Only one process can be in the running state at a time. Also see Ready State and Waiting State.

SAM. See Sequential Access Method.

SAMgen. See SAM Generation.

SAM Generation. Sam generation permits the specification of the device–dependent object modules to be linked to an application system.

Segment. A segment is a contiguous (usually large) area of memory that consists of an integral number of paragraphs. Segments are usually classified into one of three types: code, static data, or dynamic data. Each kind of segment can be either shared or nonshared. Also see Code Segment and Data Segment.

Segment Base Address. A segment base address is the high–order 16 bits of the 20–bit physical memory address. (The low–order 4 bits are implicitly 0.) The 8086 processor segment registers CS, DS, SS, and ES contain segment base addresses. Also see Logical Memory Address and Physical Memory Address.

Sequential Access Method. The Sequential Access Method provides device–independent access to devices (such as the video display, printer, files, and keyboard) by emulating a conceptual, sequential character–oriented device known as a byte stream.

Server Process. A server process (such as the spooler, remote job entry, and batch manager) is a system service that has established itself as an active server for a particular queue.

Service Exchange. A service exchange is an exchange that is assigned to a system service process at system build. The system service process waits for requests for its services at its service exchange. Also see Service Exchange Table.

Service Exchange Table. The Service Exchange Table is constructed at system build. It resides in the System Image and translates request codes to service exchanges. Also see Service Exchange.

Service Process. See System Service Process.

SetTrapHandler. SetTrapHandler is a BTOS procedural interface call that sets trap handlers. In XEBTOS, it does not create trap handlers that are a part of a partition's context; instead, it sets trap handlers that are global to the processor board on which it executes.

Size. The size of a data item or structure always refers to the number of bytes contained.

SP. See Storage Processor.

Spanned. A record file in which a record can begin and end in different physical sectors is spanned. Also see Record Sequential Access Method.

Spooler. The spooler is a dynamically installed system service that transfers text from disk files to the printer interfaces of the workstation on which the spooler is installed. It can simultaneously control the operation of several printers. A disk–based priority–ordered queue controlled by the queue manager contains the file specifications of the files to be printed and the parameters (such as the number of copies and whether to delete the file after printing) controlling the printing. This allows the spooler to resume printing automatically when reinstalled following a BTOS reload. Also see Spooler Byte Stream.

Spooler Byte Stream. A spooler byte stream automatically creates a uniquely named disk file for temporary text storage. It then transfers the text to the disk file and expands the disk file as necessary. When the spooler byte stream is closed, a request is queued through the queue manager to the spooler to print the disk file and delete it after it is printed. This is spooled printing.

STAM. Standard Access Method.

Status Code. A status code reports the success or failure of the requested operation. The system service process stores a status code in a request block, for the client process to examine.

Storage Controller. A Storage Controller is an XE520 processor board that is used with an SP to form a Disk Processor board.

Storage Processor. A Storage Processor is an XE520 processor board that controls half-inch magnetic tape drives.

Submit Facility. The submit facility permits a sequence of characters from a file to be submitted for characters typed at the keyboard. The use of submit files allows the convenient repetition of command sequences. Also see Submit File.

Submit File. A submit file is a file used in the submit facility. It contains the same sequence of characters that would be typed to the desired programs. When a submit file is activated by a request from an application process or a command to the Executive, a character from the file is returned to the application process whenever it requests a character from the keyboard. A recording file and a submit file cannot be used simultaneously. Also see Recording File and Submit Facility.

Sys.Cmds. Sys.Cmds is the Executive's command file. It contains information about each command known to the Executive. [Sys]<Sys>Sys.Cmds is used if there is no Sys.Cmds file in the Application System Control Block. The New Command command is used to enter additional commands into Sys.Cmds.

System Administrator. See System Manager.

System Build. System build is the collective name for the sequence of actions necessary to construct a customized BTOS System Image. System build allows the specification of installation-specific parameters and the inclusion of user-written system services.

System Common Address Table. The System Common Address Table contains the 4-byte logical memory address of each of a number of BTOS system data structures. It starts at physical memory address 240h.

System Common Procedure. A system common procedure performs a common system function, such as returning the current date and time. The code of the system common procedure is included in the System Image and is executed in the same context and at the same priority as the invoking process. The Video Access Method, for example, is a system Common procedure. Also see Object Module Procedure.

System Configuration Block. The System Configuration Block allows the application system to determine detailed information about the System Image (workstation configuration and system build parameters).

System Data Structures. System data structures are data areas contained within BTOS that are necessary for its operation. These structures are often configuration–dependent. A File Control Block and a File Area Block are examples of system data structures.

Sys(tem) Directory. The Sys(tem) Directory of each volume contains entries for system files, including the Bad Sector File, the File Header Blocks, the Master File Directory, the System Image, the Crash Dump Area, the Log File, and the Executive. The Sys Directory is created by the IVolume command rather than by the CreatDir operation. Also see Sys(tem) Volume.

System Event. A system event affects the executability of a process. Examples of system events are an interrupt from a device controller, Multibus device, timer, or Real–Time Clock, or a message sent to another process. The system event causes a message to be sent to an exchange at which a higher priority process is waiting; this, in turn, causes BTOS to reallocate the processor.

System Image. The System Image (the file [Sys]<Sys>SysImage.Sys) contains a run–file copy of BTOS.

System Manager. The system manager is the person responsible for planning, generating, extending, and controlling the use of BTOS to improve the overall productivity of the installation.

System Memory. System memory is a contiguous area of memory beginning at address 0 that is permanently reserved for use by BTOS.

System Partition. A system partition contains BTOS or dynamically installed system services. Also see Application Partition.

System Service. A system service is an operation performed by a system service process.

System Service Process. A system service process is a BTOS process that services and responds to requests from client processes. Both Unisys and user–written system service processes can be dynamically installed or linked to the System Image at system build. A system service process is scheduled for execution in the same manner that an application process is scheduled. Also see Application Process and Client Process.

Sys(tem) Volume. BTOS is bootstrapped from the Sys(tem) Volume. The Sys(tem) Directory of the Sys(tem) Volume contains entries for system files that are not necessary in the Sys Directories of other volumes. These additional entries must be placed in [Sys]<Sys> when the volume is initialized. SysImage.Sys, CrashDump.Sys, and Log.Sys are created (but not initialized) by the **IVolume** command. The other file entries are created using the CreateDir operation or the Create Directory command. These system files are the System Images, the Crash Dump Areas, the Log File, the Debugger, the Executive, the Executive's command file, and the standard character font. Also see CrashDump Area, Executive, Log File, Sys(tem) Directory, and System Image.

Task. A task consists of executable code, data, and one or more processes. Also see Application System, and Run File.

TP. See Terminal Processor.

Terminal Processor. A Terminal Processor is an XE520 processor board that supports a parallel printer and up to 10 RS232–C serial devices.

TRUE. TRUE is represented in a flag variable as hex OFF.

UCB. See User Control Block.

User Control Block. There is a User Control Block (UCB) for each user number. The UCB contains the default volume, default directory, default password, and default file prefix set by the last Setpath and SetPrefix operations. The UCB is memory–resident.

User File Block. The User File Block contains a pointer to the File Control Block for each open file.

User Number. A user number is a 16–bit integer that uniquely identifies an application system. Each application partition has a different user number. Processes in the same application partition share the same user number. A process obtains its user number with the GetUserNumber operation. In the primary application partition of a standalone or master workstation, the user number is always 0.

VAM. See Video Access Method.

VCB. See Volume Control Block.

VDM. See Video Display Management.

VHB. See Volume Home Block.

Video Access Method. Video Access Method (VAM) is one of three video management software levels that controls the monitor's screen.

Video Control Block. The Video Control Block (VCB) contains all information known about the video display, including the location, height, and width of each frame, and the coordinates at which the next character is to be stored in the frame by the Sequential Access Method. The VCB is located in BTOS memory at an address recorded in the System Address Table.

Virtual Code Management. Virtual code management is a facility that uses virtual memory to permit the execution of an application system that exceeds the physical memory of its partition.

Video Display Management. Video Display Management (VDM) is one of three video management software levels that controls the monitor's screen.

Volname. (Volume name) A volname is the second element of a full file specification.

Volume. A volume is the medium of a disk drive that was formatted and initialized with a volume name, a password, and volume control structures including the Volume Home Block, the File Header Blocks, and the Master File Directory. A floppy disk and the medium sealed inside a Winchester disk are examples of volumes.

Volume Control Structures. Volume control structures allow the file management system to manage (allocate, deallocate, locate, avoid duplication of) the space on the volume not already allocated to the volume control structures themselves. A volume contains a number of volume control structures: the Volume Home Block, the File Header Blocks, the Master File Directory, and the Allocation Bit Map, among others.

Volume Home Block. Each volume has a Volume Home Block (VHB). The VHB is the root structure (that is, the starting point for the tree structure) of information on a disk volume. The VHB contains information about the volume such as its name and the date it was created. The VHB also contains pointers to the Log File, the System Image, the Crash Dump Area, the Allocation Bit Map,the Master File Directory, and the File Header Blocks. The VHB is disk–resident and 1 sector in size.

Volume Password. A volume password protects the volume.

Waiting State. The waiting state is one of three states in which a process can exist. A process is in the waiting state when it is waiting at an exchange for a message. A process enters the waiting state when it must synchronize with other processes. A process can enter the waiting state only by voluntarily issuing a Wait kernel primitive that specifies an exchange at which no messages are currently queued. The process remains in the waiting state until another process (or interrupt handler) issues a Send (or PSend, Request, or Respond) kernel primitive that specifies the same exchange that was specified by the Wait primitive. Any number of processes can be in the waiting state at a time. See Ready State and Running State.

Index

E

F

G

H

I

K

L

M

Help Us To Help You

Publication Title _____

Form Number _____ Date _____

Unisys Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments _____

_____

_____

Name _____

Title _____ Company _____

Address (Street, City, State, Zip) _____

Telephone Number _____

Help Us To Help You

Publication Title _____

Form Number _____ Date _____

Unisys Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments _____

_____

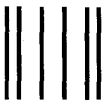
_____

Name _____

Title _____ Company _____

Address (Street, City, State, Zip) _____

Telephone Number _____



No Postage
necessary
if mailed in the
United States

BUSINESS REPLY MAIL

First Class Permit No. 817 Detroit, MI 48232

Postage Will Be Paid By Addressee

Unisys Corporation
ATTN: Corporate Product Information
P.O. Box 418
Detroit, MI 48232-9975 USA

No Postage
necessary
if mailed in the
United States

BUSINESS REPLY MAIL

First Class Permit No. 817 Detroit, MI 48232

Postage Will Be Paid By Addressee

Unisys Corporation
ATTN: Corporate Product Information
P.O. Box 418
Detroit, MI 48232-9975 USA