Transaction Processing Facility

Main Supervisor Reference

Version 4 Release 1

SH31-0159-07

Transaction Processing Facility

Main Supervisor Reference

Version 4 Release 1

SH31-0159-07

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page xi.

Eighth Edition (December 2001)

This is a major revision of, and obsoletes, SH31-0159-06 and all associated technical newsletters.

This edition applies to Version 4 Release 1 Modification Level 0 of IBM Transaction Processing Facility, program number 5748-T14, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

IBM welcomes your comments. Address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Notices

References in this book to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service in this book is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 830A
Mail Drop P131
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Any pointers in this book to non-IBM Web sites are provided for convenience only and do not in any way serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this book or accessed through an IBM Web site that is mentioned in this book.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Enterprise Systems Connection Architecture

ES/9000

IBM

Sysplex Timer

System/370

VisualAge.

Other company, product, and service names may be trademarks or service marks of others.

About This Book

The *TPF Main Supervisor Reference* describes the functions performed by the main supervisor, a component of the TPF control program, in coordinating the use of resources and maintaining processing unit operations by performing initialization, service and control, and error processing. This book describes system start-up, online system operations, the High Performance Option, system errors, and checking system internals.

In this book, abbreviations are often used instead of spelled-out terms. Every term is spelled out at first mention followed by the all-caps abbreviation enclosed in parentheses; for example, Systems Network Architecture (SNA). Abbreviations are defined again at various intervals throughout the book. In addition, the majority of abbreviations and their definitions are listed in the master glossary in the *TPF Library Guide*.

Before You Begin

You should be familiar with the main supervisor and its interfaces to other TPF components before you begin this book. See *TPF Concepts and Structures*.

Who Should Read This Book

This manual is intended for system programmers responsible for maintaining and modifying the main supervisor; it provides the information needed to understand the main supervisor component.

Conventions Used in the TPF Library

The TPF library uses the following conventions:

Conventions	Examples of Usage
italic	Used for important words and phrases. For example: A *database* is a collection of data. Used to represent variable information. For example: Enter **ZFRST STATUS MODULE** *mod*, where *mod* is the module for which you want status.
bold	Used to represent text that you type. For example: Enter **ZNALS HELP** to obtain help information for the ZNALS command. Used to represent variable information in C language. For example: **level**

`monospaced`	Used for messages and information that displays on a screen. For example:
	`PROCESSING COMPLETED`
	Used for C language functions. For example:
	`maskc`
	Used for examples. For example:
	`maskc(MASKC_ENABLE, MASKC_IO);`

bold italic	Used for emphasis. For example: You ***must*** type this command exactly as shown.

Conventions	Examples of Usage
Bold underscore	Used to indicate the default in a list of options. For example: **Keyword=OPTION1** | **DEFAULT**
Vertical bar |	Used to separate options in a list. (Also referred to as the OR symbol.) For example: **Keyword=Option1** | **Option2** **Note:** Sometimes the vertical bar is used as a *pipe* (which allows you to pass the output of one process as input to another process). The library information will clearly explain whenever the vertical bar is used for this reason.
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=*option*

Scale	Used to indicate the column location of input. The scale begins at column position 1. The plus sign (+) represents increments of 5 and the numerals represent increments of 10 on the scale. The first plus sign (+) represents column position 5; numeral 1 shows column position 10; numeral 2 shows column position 20 and so on. The following example shows the required text and column position for the image clear card.
	`|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...`
	`LOADER  IMAGE CLEAR`
	Notes:
	1. The word LOADER must begin in column 1.
	2. The word IMAGE must begin in column 10.
	3. The word CLEAR must begin in column 16.

Related Information

A list of related information follows. For information on how to order or access any of this information, call your IBM representative.

IBM Transaction Processing Facility (TPF) 4.1 Books

- *TPF ACF/SNA Data Communications Reference*, SH31-0168
- *TPF Concepts and Structures*, GH31-0139
- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF System Macros*, SH31-0151
- *TPF General Macros*, SH31-0152
- *TPF Multi-Processor Interconnect Facility Reference*, SH31-0155
- *TPF Operations*, SH31-0162
- *TPF Program Development Support Reference*, SH31-0164
- *TPF System Generation*, SH31-0171
- *TPF System Installation Support Reference*, SH31-0149.

IBM Enterprise Systems/9000 (ES/9000) Books

- *ES/9000 Models 330, 340, 580, 620, and 720 Functional Characteristics and Configuration Guide*, GA22-7138
- *ES/9000 Models 520, 640, 660, 740, 820, 860 and 900 Functional Characteristics and Configuration Guide*, GA22-7139
- *ES/9000 Models 711, 821, 822, 831, 941, 942, 952, 962, 972, and 982 Functional Characteristics and Configuration Guide*, GA22-7144.

Miscellaneous IBM Books

- *Environmental Record Editing and Printing Program (EREP) User's Guide*, GC35-0151
- *Environmental Record Editing and Printing Program (EREP) Reference*, GC35-0152
- *ESA/390 Principles of Operation*, SA22-7201.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TPF information, use one of the methods that follow. Make sure you include the title and number of the book, the version of your product and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send your comments electronically, do either of the following:
  - Go to http://www.ibm.com/tpf/pubs/tpfpubs.htm.

    There you will find a link to a feedback page where you can enter and submit comments.
  - Send your comments by e-mail to tpfid@us.ibm.com
- If you prefer to send your comments by mail, address your comments to:

  IBM Corporation
  TPF Systems Information Development
  Mail Station P923
  2455 South Road
  Poughkeepsie, NY 12601-5400
  USA
- If you prefer to send your comments by FAX, use this number:
  - United States and Canada: 1 + 845 + 432 + 9788
  - Other countries: (international code) + 845 + 432 +9788

System Initialization

System initialization consists of 4 functions:

- Initial program load (IPL). The *initial program loader* loads the control program from auxiliary direct access storage devices (DASD) into main storage.
- Main storage initialization. The *initializer*, CCCTIN, allocates and initializes all of the main storage areas the control program needs to operate.
- System restart. The *system restart program* activates a series of ECB-controlled system programs, which place the system in a condition to process real-time input.
- State change. The *system state change* program activates and deactivates system resources and functions.

Initial Program Loading

There are 2 types of IPL, software and hardware.

- An operator activates a *hardware IPL* through the processor control console.
- A *software IPL* can be activated by:
  - The ZRIPL command, issued by an operator or by control program software, or
  - A catastrophic error.

On a hardware IPL the operator is given the opportunity to modify the current active TPF image.

The devices reset by a hardware IPL are not reset by a software IPL. IPL segments:

- Read the image control record (ICR) and obtain the address of the image pointer record (CTKX).
- Perform the initial call to the common I/O handler.
- Load the CTKX and keypoint records.
- Determine the number and status of I-streams on the processor.
- Verify the online DASD.
- Load the control program segments.
- Load the tables from the core image restart area to their main storage locations.

Multiple TPF Images

Multiple TPF images allow the TPF user to integrate changes in a TPF environment by:

- Performing loads in NORM state without destroying the existing program base.
- Falling back immediately to a previous program base with a single IPL without re-IPLing the loader general file.

It is possible to have as many as 8 images. One primary image is designated for use during a hard IPL. Each image has a unique core image restart area (CIMR). Each image also has an IPL area (IPLA and IPLB), E-type program base, and a keypoint staging area (to facilitate the pre-loading of keypoints in NORM state).

Note: There is one working keypoint area that contains the active keypoints. This area is used no matter what image you IPL on.

If, upon a hardware IPL, you want to modify the current active image and more than one image is enabled, all the enabled images will be displayed. You will then be prompted to select an image for the BSS. If the selected image's IPL area is different from the primary image's IPL area then a software IPL will occur, so that the new IPL area can be read in. The primary image is the image whose IPL area is used during a hardware IPL prior to prompting the operator for image selection.

The ZIMAG commands and ZTPLD command can be used to define and manipulate the TPF images. See *TPF Operations* for detailed information on these commands.

The ZIMAG Commands

Option	Description
DEFINE	Allows you to define (or redefine) as many as 8 images.
DISPLAY IMAGE	Displays the image name, status, associated IPL and program areas, CTKX version code (if physically loaded), and CIMR component.
ENABLE	Allows you to enable an image for an IPL.
PRIMARY	Defines an enabled image as the primary image. The primary image is used during a hard IPL and is valid only on the basic subsystem (BSS).
DISABLE	Disables an enabled image so that it cannot be IPLed.
CLEAR	Deletes a disabled image.
COPY	Allows you to copy core image restart area (CIMR) components from one image to another by reference (logically) or physically.
UNREF	Deletes the logical references of CIMR components from a disabled image.
MAKEPHYS	Allows you to make all of your CIMR component references physical copies.
DISPLAY PROG	Displays all of the program areas defined in the TPF 4.1 system and identifies which (if any) images they are associated with.
DISPLAY IPL	Displays all the IPL areas that were loaded and identifies which (if any) images they are associated with. This option also displays IPLA and IPLB information for each area.
DISPLAY PROCESSOR	Displays the image associated with each processor in the complex, as well as the status of the processor.

The ZTPLD Command

Message	Description
ZTPLD	Load data to a specified image

Considerations for HPO Users

Under MDBF, the operator specifies the IPL type as one of the following:
- Basic subsystem only
- Multiple subsystems
- Fast IPL.

You can define a new subsystem configuration when you select the multiple subsystems option. Any existing subsystem configuration is cleared.

The fast IPL option allows you to configure subsystems from the existing configuration. If there is an I/O error, the subsystem configuration is purged. Otherwise, the new subsystem configuration is copied from the existing one, and further operator intervention is not required.

Keypoint record M (CTKM) contains subsystem configuration data. The IPL program uses CTKM each time a new subsystem configuration is generated. Under MDBF, a software IPL can be activated when the system operator issues ZRIPL, or when a catastrophic error occurs. During the IPL, the operator may receive a message indicating there is not enough storage to continue. The operator can use the ZFKPA command to reduce the number of main storage blocks or to cancel the IPL.

MDBF System Definition

The MDBF system is loaded as one of the following:
- A basic subsystem with a single subsystem user
- A basic subsystem with multiple subsystem users
- Multiple subsystems with single or multiple subsystem users.

Generating multiple subsystems is a process that includes:
- Loading subsystem keypoint records
- Performing disk roll call (verifying all the channel paths to the online modules)
- Loading the subsystem tables.

The IPL program determines the configuration for a basic subsystem with one or more subsystem users. IPL also assigns an identifier or *subsystem user ID* (SSU ID) to each subsystem user.

In a system with multiple subsystems, the configuration can be determined by either the system operator or the IPL program. For an operator-determined configuration, the operator provides subsystem definitions during loading of the subsystem keypoints. The IPL program configures multiple subsystems when a software IPL is started or fast IPL has been specified.

Both methods require the use of CTKM. The system initialization program (SIP) generates CTKM, but this keypoint record is not completely initialized until the first IPL is completed. System startup, restart, and state change programs perform initialization during the first and subsequent IPLs of the system. The MVS utility program ICKDSF initializes the basic subsystem loader general file with standard IPL records and the IPL segment IPLA. ICKDSF also initializes the volume labels on the online basic subsystem file disk packs. The general file loader online segment (ACPL) writes the IPL1 and IPL2 records to these disk packs.

Processor Identification

Loosely coupled processors require unique identification for a number of reasons, such as identifying processor unique keypoints and communication between processors. The control program records the serial numbers from the processors that are IPLed and places them in keypoint record I (CTKI). CTKI is referred to as

the *loosely coupled identity table.* For each processor it loads, the IPL program establishes a connection between the processor and a central processing unit (CPU) ID. Either the system operator (under IPL program direction) or the IPL program itself (using the information in CTKI) can establish processor identifiers.

Each processor must be assigned an ID for the IPL to continue. Once an ID is assigned to a processor, the process of coupling the processor with the rest of the complex begins.

Lock Management

TPF loosely coupled processors share a common database that resides on shared DASD. Updates to the database are controlled by locking instructions executed in the shared DASD control units. DASD lock facilities-the limited lock facility (LLF) and the concurrency filter lock facility (CFLF)-reside in DASD control units and provide the hardware support for these locking instructions. All DASD used by a TPF loosely coupled complex must be connected to control units with LLF or CFLF.

LLF and CFLF provide locking based on the physical attachment of the processor to the control unit. The point of attachment between a DASD control unit and a channel from a processor is called a control unit interface. In an 8 processor loosely coupled complex each control unit would require 8 interfaces, one for each processor in the complex.

IDs are assigned to the control unit interfaces on each control unit starting with one and as many as the maximum number of interfaces available. A given processor in a loosely coupled complex using LLF must be attached to the same interface ID on all attached DASD control units. This restriction does not apply if CFLF is used.

The IPL program examines all of the control units that a processor has access to and identifies the existing LLF or CFLF relationships. If LLF or CFLF errors are found, the IPL program may limit the system to single processor mode **despite the fact that the loosely coupled facility is present**.

Only one processor in a loosely coupled complex loads the control program to the online modules. No other processor should be active when the one processor loads the control program.

Initializing Main Storage

The programs that initialize main storage, referred to collectively as the *initializer,* receive control directly from the IPL program.

The initializer is a stand-alone CSECT, CCCTIN, which does not need to access tape or disk data. All of the necessary information is maintained in the keypoint records and essential control program records. The IPL program passes the initializer a parameter list that contains the addresses of the records. The initializer must be executed before ECB-controlled programs can use control program services. (Refer to “Controlling E-Type Programs” on page 15 for a description of ECB-controlled programs.)

The initializer sets up all fixed and working main storage. The main storage allocation programs then control the dynamic use of working storage (see “Managing Storage” on page 21).

Low-Address Protection

Low-address protection is a hardware facility that protects the first 512 bytes of each CPU's page 0 — the part of storage most apt to be corrupted by programs. Neither applications nor the TPF control program can modify this storage while low-address protection is active.

Low-address protection is turned on by IPLB. Programs that modify the first 512 bytes of memory can use the LCPCC macro to turn low-address protection off (and on again). Commands that alter main storage (such as ZACOR and ZAPGM) turn off low-address protection internally.

System Restart

System restart places the system in a condition to process real-time input; that is, it places the system in 1052 state. System restart initializes system tables used for resource management and control program execution. The order in which system restart initializes these programs is extremely important. You should analyze any change to the restart series very carefully before implementing it.

The initializer activates the restart program when an IPL of the online system occurs. The general file loader online segment (ACPL) activates the restart program when an IPL of the general file occurs. The general file is an offline TPF-formatted disk pack. See *TPF System Installation Support Reference* for more information on the contents of the general file.)

State Change

The state change process is known as *cycle-up* when system resources and functions are being activated and *cycle-down* when system resources and functions are being deactivated. During cycle-up, additional resources and functions are made available in successively higher *system states*, until all resources are available. There are 5 system states cycled up in the following ascending order:

- 1052 state
- Utility state
- CRAS (computer room agent set) state
- Message switching state
- Norm state.

You can change system states with the ZCYCL command and display the system state with the ZDSYS command. See *TPF Operations* for detailed information on the functions available in each state.

Note: Application programmers should consider the services and functions that are operational during a particular system state when designing application programs.

1052 State

1052 state is the lowest system state. Most system services are inactive. Services active during 1052 state include:

- Commands from the system console or 3270 local devices logged into the system message processor. 3270 devices must have their addresses in the CRAS status table (CRAT).
- Keypoint update.

Utility State

Utility state can be entered only from 1052 state. Thus, the system must be returned to 1052 state from any other state (CRAS, message switching, or norm) before utility state can be entered. Utility state services include:

- Commands from CRAS terminals
- The real-time clock (which is adjusted to compensate for time spent in 1052 state)
- Time-initiated entries
- Keypoint update
- Disk lost interrupt
- Tape lost interrupt
- Interrupts from 3270 local devices (but messages are rejected if the terminal is not logged into the system message processor).

CRAS State

In *CRAS state*, all of the services active during utility state are active, as well as the following services:

- All messages are accepted, but only from CRAS terminals. All other terminals receive a response of: `SYSTEM RESTRICTED. RETRY IN 5 MIN.`
- The GFS (get file storage) facility is available. You can issue GETSC, GETLC, and RELFC macros. See *TPF General Macros*.
- All active high-speed and BSC communication lines are polled. Synchronous link control (SLC) is active.

Message Switching State

Message switching state is similar to CRAS state, with the following exceptions:

- All non-SDLC (Synchronous Data Link Control) lines are active.
- Only message switching entries are processed. All other entries are either rejected or queued for later processing.

Norm State

Norm or normal state allows the activation of all system and application services and functions. All entries are allowed in norm state.

- The time available supervisor (TAS) is active.
- Catastrophic error recovery is attempted.
- Polling of the Network Control Program (NCP) can be started.

Restart and State Change Considerations for HPO Users

- The restart program (CTKS) invokes programs CNAE and CNAJ to send the status of a processor to every other active processor in the complex.
- CTKS checks each processor during restart to determine if the processor was previously inactive. If the processor was inactive, CTKS initiates a software IPL to synchronize this processor with the others in the complex.
- Program CNPY is called to ensure that only one processor is active if the IPL program designated the processor as the first in the IPL chain.
- Certain restart programs ensure that all loosely coupled processors access the same DASD via the same channel and control unit. See the ZRSTT command in *TPF Operations*. These programs also maintain the status of each processor and make the status available to the other processors in the complex.

- In an MDBF system, the basic subsystem must be cycled up before any other subsystem; no subsystem can be in a higher system state during cycle-up.

System Service and Control Functions

This chapter explains the purpose and use of the system and control functions provided by the control program main supervisor. System service and control functions are configuration dependent. These vary in each TPF installation. For example, a system that supports financial applications may require control or maintenance functions not necessarily required in an airlines reservation system.

Supporting a Tightly Coupled System

A TPF *tightly coupled processing system* consists of main storage, one copy of the TPF system, a Central Processing Complex (CPC), and a channel subsystem. A CPC is a configuration that contains one or more processors or instruction streams (I-streams). The I-streams within the CPC share main storage. TPF uses the term I-stream interchangeably with processor.

TPF tightly coupled support is designed to provide the image of one single, very fast *uniprocessor*. There is only one copy of the TPF system executing on a CPC regardless of the number of I-streams that CPC contains.

A detailed description of the TPF system tightly coupled support can be found in *TPF Concepts and Structures*.

Supporting Common I/O Operations

Common I/O (CIO) supports System/370 XA I/O architecture for processors executing in ESA (ESA/370 and ESA/390) mode. This support includes:

- A device addressing scheme called *symbolic device addressing.*

  In ESA mode, all I/O operations are directed to subchannel numbers. The subchannel numbers are generated in ascending order, starting with 0. A symbolic device address (SDA) is the user's designation for the subchannel associated with a particular device.
- Processing to initialize CIO control blocks.

  CIO code, included as part of IPLB, allocates and initializes the following control blocks as part of system initialization:
  - CIO communications area
  - I/O trace table
  - Lost interrupt table
  - Fast path translate table
  - Device blocks
  - Interrupt save blocks.
- CIO functions to process System/370 XA I/O:

  These functions include processing the system macros that request I/O services, initiating I/O operations, and intercepting the I/O interrupts and delivering them to the TPF device interrupt handlers.
- System macros to invoke the CIO functions.

  The system macros used to request I/O services include the following:

  MSDAC: Mount SDA

  DSDAC: Dismount SDA

  ISDAC: Display SDA status

RIOSC: Reset SDA

SIOSC: Initiate I/O to SDA

HIOSC: Halt active I/O

IOIRC: CIO interrupt routine return

A system macro, CIOSC, allows real-time (E-type) programs to mount, dismount or request status for an SDA.

- A preemptive I/O (PIO) mode enabling special I/O operations while normal system I/O activity is suspended.

  This mode provides similar functions to the normal CIO activity. I/O operations are initiated, interrupts are intercepted, and an interface to PIO device interrupt handlers is supported.

  The following PIO macros are supported:

  SPNDC: Suspend normal system I/O activity

  RESMC: Resume normal system I/O activity

  PIOFC: Initiate a PIO operation to an SDA

  PIORC: PIO interrupt routine return

  TIOSC: Test I/O services

Additional information regarding the macros can be found in *TPF System Macros* and *TPF General Macros*.

Using Keypoints to Maintain System Operations

Control program keypoints are data records used to maintain system operations. These records reflect the current status of the system and are essential to the startup/restart process. A copy of the keypoint records is maintained in the fixed file area on each disk pack in the online system.

File-resident keypoints are retrieved and updated via system ECB-controlled programs. There is one program for retrieving records; one for filing. As these keypoint records are resident in file storage, control program macros are used to activate the update sequence (see “Initializing ECBs for Entries from Control Transfer Macros” on page 15).

TPF users can maintain main-storage resident data records in a portion of main storage called the *global area*. Global records are data records that support user written applications. A back-up copy of the global records is maintained on file. For this reason, global records are often referred to as application keypoint records. Main-storage resident control program and application keypoints are updated through a program called the keypoint update mechanism. Requests to update main-storage resident keypoints (both control program and application) receive a higher priority than do requests to update file-resident keypoints. This process is referred to as *demand* keypointing.

Control program keypoints are 4096-bytes (4KB) and are retrieved in 4KB blocks. Any ECB-controlled program can retrieve or file a keypoint record. When the file copy of a main storage keypoint is updated, that is, *keypointed*, the name of the program that initiated the request is placed within the filed record.

The record types used to manage keypoints are the keypoint staging area (#KSA*x*), the working keypoint area (#KEYPT), and the keypoint backup area (#KBA). The keypoint staging areas are image-unique records; therefore, there is a one-to-one correspondence between the #KSA areas and the #CIMR areas. The auxiliary loader loads to the keypoint staging areas rather than directly to the working keypoint area. The keypoints can then be moved to the working area by issuing the ZIMAG KEYPT MOVE command. There are also other ZIMAG KEYPT command options to manipulate and display the keypoint staging area. The working keypoint area is shared by all images and contains the keypoints that are currently used by the online system. Keypoints can be loaded directly to the working keypoint with the general file loader. The keypoint backup area is used to maintain backup copies of the active keypoints that have been overlaid by a ZIMAG KEYPT MOVE command. Keypoints can be selectively fallen back from the keypoint backup area with the ZIMAG KEYPT RESTORE command. See *TPF Operations* for more information about the ZIMAG command. See *TPF System Installation Support Reference* for more information about multiple TPF images.

Note: Each subsystem in a multiple database function (MDBF) system can maintain unique keypoints or may share keypoints with the basic subsystem. Shared keypoints are filed using information that identifies the filing subsystem.

Time Initiated Keypoint Copy

Copies of the control program keypoints are written, on a rotational basis, to the fixed file area on the first 256 device type A disk packs in the system. The keypoints are copied from a location on one pack that is not used in the copy rotation. This pack is called the prime module. In a duped system, the dupe of the prime module will not be in the copy rotation. In a fully duped system, keypoints are written to one disk pack and not its corresponding dupe pack. This is because the order in which the keypoints are written is determined by the disk pack's symbolic module number. This function is used to propagate the ICR, CTKX and IPL areas across the online modules. Using the time initiated keypoint copy function means that when someone is IPLing a pack with the most recent keypoints, the most recent copies of the ICR, CTKX, and IPL areas will also be there. Note that because the ICR, CTKX, and IPL areas are not altered often, they do not need to be propagated with every time initiated update. Instead, these areas are copied to all modules in a cyclic order when any of them is modified. Because the propagation of the IPL areas is infrequent and is not required when the ICR or CTKX is modified, it is controlled separately.

Note: In an MDBF environment, the active processor with the lowest ordinal number copies the keypoints for all processors in the complex.

An alternate keypoint copy mechanism called *fallback extent keypointing* can also be generated. In fallback extent keypointing, keypoints are copied from their prime location to an extent area on the same disk pack. This area is an extension of the prime area (not necessarily contiguous to it). Each fallback copy area (fallback extent) is a separate record definition in the FACE table (FCTB) with record types #KFBX*n* (where $n = 0 – 254$). Fallback extents must be defined using contiguous record types starting with record type #KFBX0. If *n* fallback extents are defined, the record types that are used to define them in the FACE table must be #KFBX0 – #KFBX*m*, where *m* equals $n - 1$. The keypoints are copied to these areas on the same timed rotational basis as described previously. This option is intended for use only in an MDBF system where the basic subsystem (BSS) is generated on a small number of disk packs. Fallback extent keypointing is not necessary when the

number of packs that are available is enough for normal rotational keypointing. Keypoint fallback extents that are defined on subsytems other than the BSS are not used.

The time increment for keypoint copying is set in keypoint record A (CTKA) during system initialization. The keypoint copy program reactivates itself with a CRETC (create a time-initiated entry) macro using the increment value in keypoint A. See *TPF General Macros* for more information about the CRETC macro, or the *TPF C/C++ Language Support User's Guide* for more information about the cretc function.

Control Program Keypoints

Table 1. Control Program Keypoints

Keypoint	Macro Name	Function	Processor	SS	Initialized By	Residency	Demand Keypointable
Record A (CTKA)	CK1KE	Contains information required for system loading and for the initializer program.	Unique	Unique	SIP	File	No
Record B (CTKB)	CK9KC	Miscellaneous initialization and restart values, for example, clock status, VFA status, and DASD error thresholds.	Unique	Unique	SIP	Main storage	Yes
Record C (CTKC)	CK8KE	Status of Computer Room Agent Set (CRAS) attached terminals, initial Routing Control Application Table (RCAT) and Terminal Address Table (WGTA) control information.	Shared	Shared	SIP	Main storage	No
Record D (CTKD)	CK7KE	Status used by the synchronous link programs.	Unique	Shared	SIP	Main storage	Yes
Record E (CTKE)	CK6KE	Describes the non-SNA communications network.	Unique	Shared	SIP	File	No
Record I (CTKI)	IC0CK	Describes the status of all processors in a loosely coupled complex of the HPO feature.	Shared	Shared	SIP	File	No
Record M (CTKM)	MK0CK	Describes the status of each subsystem and each subsystem user.	Shared	Shared	SIP	Main storage	No
Record V (CTKV)	IDSCKV	Contains volume serial number ranges for the online modules, the copy module, and the loader general file.	Shared	Unique	SIP	File	No
Record 0 (CTK0)	CK0KE	Contains legal disk hardware addresses.	Shared	Shared	SIP	File	No
Record 1 (CTK1)	CK2KC	Contains the Tape Status Table (TSTB).	Unique	Shared	N/A	Main storage	Yes
Record 2 (CTK2)	CK2SN	Contains all the information in the system about the SNA configuration and the TCP/IP device parameters.	Unique	Shared	Source, contains no SIP provided inputs	Main storage	No
Record 3 (CTK3)	None	This keypoint is available for customer use.	Unique	Shared	Customer	File	No
Record 4 (CTK4)	VK4CK	This keypoint is available for customer use.	Shared	Unique	Customer	File	No
Record 5 (CTK5)	None	This keypoint is reserved for IBM use.	Shared	Shared	N/A	File	No

Table 1. Control Program Keypoints (continued)

Keypoint	Macro Name	Function	Processor	SS	Initialized By	Residency	Demand Keypointable
Record 6 (CTK6)	CJ6KP	Contains the DASD module status indicators.	Shared	Unique	SIP	File and main storage (see note 1)	No
Record 9 (CTK9)	CY1KR	Contains the status of the DASD pools.	Shared	Unique	Source, contains no SIP provided inputs	Main storage	No

Note:

1. The entire keypoint is file-resident; the first section of the keypoint is also main-storage-resident.
2. *Processor shared* means that there is one copy of the keypoint for all processors in a loosely coupled environment.
3. *Processor unique* means that there is one copy of the keypoint per processor in a loosely coupled environment.
4. *Subsystem (SS) shared* means that there is one copy of the keypoint residing in the BSS in an MDBF environment.
5. *Subsystem (SS) unique* means that there is one copy of the keypoint per subsystem in an MDBF environment.

Controlling E-Type Programs

The entry control block (ECB) is the primary interface between real-time programs and the control program. Programs that require an ECB to operate are called ECB-controlled or E-type programs. An ECB is created for each new entry in the system and assigned an activation number, which is obtained from a system counter in OPZERO. This activation number determines which version of the E-type program the ECB will use. All references to system resources and register usage are maintained in the ECB. The ECB remains associated with the entry for which it was created until the entry is completely processed. After the entry is processed, the ECB is returned to the pool of available ECB storage blocks (see "Managing Storage" on page 21). ECB processing includes:

- Initializing ECBs for input message entries
- Initializing ECBs for entries from control transfer or create macros
- Transferring control among ECB-controlled programs
- Suspending processing of ECBs
- Returning ECBs and associated resources after entries are processed.

Initializing ECBs via OPZERO

The ECB initialization routine, OPZERO, initializes an ECB for each new entry in the system. OPZERO is usually activated by the CXFRC macro or by the control program polling the various communication lines for incoming messages.

OPZERO does the following:

1. Obtains a storage block from the pool of available ECB blocks.
2. Initializes fields in the block to indicate the status of the ECB, the resources attached to the ECB, and the activation number assigned to the ECB.
3. Initializes the ECB to service premature error conditions.

CSECT	Segment	ECB Initialization for
CCCCP1	CLPE	1052/3215 console entries
CCCCP2	CLSQ	High-speed line entries
CCCCP3	CLQC	Synchronous link control (SLC) entries
CCCCP4	CLOC	Entries from 3270 local devices
CCCCP5	CRMR	Binary synchronous communication (BSC) entries
CCCSNA	CS10	Systems Network Architecture (SNA) entries

Initializing ECBs for Entries from Control Transfer Macros

Control program segments use the control transfer, CXFRC, to create an ECB and transfer control to another program that is then free to use any control program macros. By using CXFRC, you can create an ECB immediately or defer creation. You can also create an ECB with an attached block.

The CXFRC macro obtains the current activation number from OPZERO and assigns it to the ECB that is being created.

Initializing ECBs for Entries from Create Macros

When an active ECB needs to create a second ECB to perform independently, it can issue one of the following *create* macros:

CREMC Creates an independent ECB for immediate processing and adds it to the ready list.

CREDC Creates an independent ECB for deferred processing and adds it to the deferred list.

CREXC Creates an independent ECB for low-priority deferred processing and adds it to the deferred list.

CREEC Creates an independent ECB for immediate or deferred processing and adds it to the ready list or the deferred list.

CRETC Creates an independent ECB for activation after a specified interval of time.

SWISC TYPE=CREATE
Creates an independent ECB on a specified I-stream.

These create macros, except for the CRETC macro, assign the second ECB the same activation number as the parent ECB. The CRETC macro obtains the current activation number from OPZERO and assigns it to the second ECB.

The create macros are serviced by the control transfer routine (CXFRC).

Transferring Control among E-Type Programs

When E-type programs need to request the services of other E-type programs, they use Enter/Back macros. E-type programs, which can reside in file storage, virtual file access (VFA) buffers, or main storage in either the 24- or 31-bit core resident program areas or in a common block are brought into main storage upon request for execution. Enter/Back macros perform the following tasks:

- Maintain control of program records and associated ECBs
- Pass the contents of registers R0–R7 to the program
- Save the program base, addressing mode, and next sequential instruction (NSI) of the active program
- Activate the requested program at the correct entry point in 24-bit or 31-bit addressing mode (addressing mode is based on the allocation characteristics of the program).

Enter macros are used to transfer control as follows:

ENTNC Releases the active program from the ECB and transfers control to the requested program.

ENTRC Transfers control to the requested program. The original program is not released from the ECB; it is attached to the ECB and is said to be nested. Return is expected to the NSI in the original program.

ENTDC Releases all nested programs from the ECB and transfers control to the requested program.

BACKC Returns control to the nested program (attached to the ECB) that last issued an ENTRC.

SWISC TYPE=ENTER
Performs the ENTDC function while moving the existing ECB to a specified I-stream.

Suspending Processing of Entries

There are two distinct sets of macros that you can use to suspend processing of an entry. The first set of macros are the defer, delay, and event macros which you can

use to immediately force an active entry to be suspended. These macros provide only limited control over a suspended entry.

The second set of macros are the load-balancing and time-slicing macros. These macros allow more control over when an entry gets suspended and when the entry resumes running. The load-balancing macros allow an entry to be suspended and unsuspended based on the availability of system resources. The time-slicing macro allows an entry to be suspended and unsuspended based on user-specified time intervals.

Defer, Delay, and Event Macros

Macros are provided that permit the entry currently in control in an I-stream engine to request processing to be suspended. A suspended entry is placed on the input or deferred CPU loop list (dispatch control list) or in the event table. The following macros are used to suspend processing:

- DEFRC — defers processing of the current entry. The entry is placed at the bottom of the deferred list. The deferred list is simply the lowest-priority list serviced by the CPU loop. After the entry is placed on the deferred list, the system will service the higher-priority lists.
- DLAYC — delays processing of the current entry. The entry is placed at the bottom of the input list. The DLAYC macro suspends processing for a shorter period of time than the DEFRC macro.
- EVNWC — delays processing of the current entry until a named event completes its function. The entry is placed in the event table until the event completes its function.

DEFRC and DLAYC macro requests are different from a WAITC request. WAITC is related to the completion of an I/O operation; if no I/O is pending, no delay occurs. DLAYC and DEFRC macro requests place the entry at the bottom of a list; their use is usually related to indicators to be set by other programs associated with the program that requests the suspension.

The EVNTC macro is used to name an event which is an item in the event table. The EVNWC macro is used to specify, through a field in the event table that is known to all parties, that a bit-setting event should occur before the entry regains control (after the EVNWC is specified). This removes the need for an ECB-controlled program to continue to place itself on the deferred list each time the event fails; the checking is driven by timers in the control program and the entry is placed on the ready list when the event occurs.

Load-Balancing and Time-Slicing Macros

Two macros are provided to help to automatically control system activity. One macro can be used to suspend low-priority work during peak or busy periods. The other macro provides a way to force CPU-intensive applications to repeatedly give up control after running for a specified period of time. The macros are:

- LODIC macro — checks if enough system resources are available to begin processing low-priority work. Depending on the parameters with which LODIC is called, the ECB that issued the LODIC macro can become suspended when system resources fall below user-selected shutdown levels. Once suspended, the ECB will not get back control until enough system resources are available again.
- TMSLC macro — enables or disables an ECB to be time sliced. When an ECB is enabled for time slicing, the ECB will lose control at user-defined time intervals.

When you run one of these macros, the macro service routine sets up the appropriate ECB fields and return conditions. In most cases, control returns to the

entry through the common macro exit routine. In other cases, the entry does not immediately get back control. The entry is placed on the suspend list instead, and the system task dispatcher (CPU loop) receives control.

While suspended, ECBs that issued the LODIC or TMSLC macros reside on a suspend list. The suspend list is checked once every pass through all items on the input list.

Returning ECBs after Entries Are Processed

The EXIT routine does the processing necessary to remove an entry from the system. All of the exiting entry's resources are returned to the control program. These resources can include storage blocks, programs, data records, unit record devices and tapes. The EXIT routine checks fields in the ECB for the status of these resources, returns the resources, and ultimately releases the ECB. You can activate the EXIT routine explicitly with the EXITC macro, or system error programs can invoke it. (See "Error Recovery" on page 39).

Managing Address Spaces

TPF supports two types of address spaces, the system virtual memory and the ECB virtual memory. The system virtual memory (SVM) contains all storage that can be used by a particular I-stream and dispenses storage, as needed, to ECBs. Each I-stream has its own SVM. The ECB virtual memory (EVM) contains all storage that can be referenced by an ECB. An ECB can only access storage in its own EVM.

ECBs run in the EVM and control program services run in either the EVM or the SVM. I/O servers execute in the EVM for macro service routines and then switch to the SVM for starting I/O and I/O interrupt routines. I/O servers execute in the SVM for I/O post-interrupt routines that then switch to the EVM for returning to the user. The layouts of both the EVM and SVM are shown in Appendix B, "Virtual Storage Layout" on page 51.

When you write control program code, you must know whether you are in the SVM or EVM. The control program uses the following macros for managing these address spaces:

$GEVAC To convert SVM addresses to EVM addresses

$GSVAC To convert EVM addresses to SVM addresses

$SWSPC To switch between the 2 address spaces

E-type programs use the following macros for managing address spaces:

MOVEC To move data between the EVM and the SVM in any combination (from EVM to EVM, EVM to SVM, SVM to SVM, and SVM to EVM).

GSVAC To convert EVM addresses to SVM addresses

For more information about each of these macros, see *TPF General Macros*.

Handling System Interrupts

An *interrupt* is a hardware-enforced transfer of control. There are five types of interrupts: external, input/output (I/O), machine check, program check, and supervisor call (SVC). Table 2 on page 19 summarizes the different types of interrupts, which areas of the control program process them, and what causes them.

Table 2. System Interrupts

Type of Interrupt	Processed By	Caused By
External	External interrupt handler (CTME)	• Signal from the interrupt key on the system console • TOD clock comparator • CPU timer • Sysplex Timer (STR) • TOD clock *synch checks* • Service signals • Malfunction alerts • External calls.
I/O	Common I/O handler (CCIO)	Status from an I/O operation
Machine check	Machine check handler (CMKH)	Equipment malfunction (hardware error)
Program check	System error processor (CPSE)	Programming error
SVC	SVC macro decoder (CEDM)	Supervisor call instruction

SVC and Fast-Link Macros

SVC instructions are requests for control program services. When an SVC is executed, the macro decoder gets control and PSWs are swapped. The SVC new PSW allows the macro service routine to store into any area of memory, and issue supervisor state instructions.

For some requests for control program services the storage protection key does not need to be changed, privileged instructions are not needed, and protected storage is not updated. For these requests, TPF *fast-link* macros provide a faster processing path than SVC instructions. (This is because fast-link macros use a Branch and Save Register instruction, rather than swapping PSWs.)

Pointers to SVC routines are maintained in tables. There are 255 SVC entries in the primary SVC tables (for main and application I-streams). Of these 255 entries, 31 are reserved for customers or “users,” and 2 are reserved to point to a second-level structure. This *secondary* or *indexed* table supports an additional 32,767 SVCs. To specify secondary SVCs, add a halfword index number to the SVC instruction.

You can define and maintain data in the primary and secondary SVC tables and the fast-link macro tables using the following macros:

CRESVC Defines an SVC or fast-link macro to the system and adds the macro definition to the primary, secondary, or fast-link macro table.

CREGPC Defines a macro group so that you can set debugger macro breakpoints to trap a group of SVCs instead of requesting the SVC names individually.

USRSVC Contains user-defined CREGPC and CRESVC macro calls.

CFISVC Returns the address of the macro table entry for a specified SVC.

SVC Macro Decoder

All SVC instructions that are requests for some control program service activate the SVC macro decoder (CEDM).

1. CEDM saves the active program's registers in the entry's ECB.
2. If keypoint record A (CTKA) specifies that macro trace is active, CEDM records the use of supervisor call (SVC) macros in the ECB.
3. CEDM routes the request to the correct macro service routine.
4. Once the macro request is completed, CEDM restores the active program's registers and returns to the next sequential instruction.

(See "Checking System Internals" on page 47 for information on ECB-check and the internal trace table.)

Fast-Link Macro Decoder

The fast-link macro decoder is activated through a branch and save register (BASR) instruction. Therefore, the fast-link macro decoder operates in problem state. Control is passed directly to the requested macro service routine, which completes the request and activates the common fast-link exit routine. This routine is responsible for restoring registers and returning control to the program's next sequential instruction.

Adding New SVC or Fast-Link Macros to the System

To add new SVC or fast-link macros:

1. Code the SVC service routine and the macro that invokes the service routine. The SVC service routine should be in a copy member that is called by CCUEXT.
2. Code a CRESVC macro for each new SVC or fast-link macro, and optionally code the CREGPC macro for each new macro group and put these calls into USRSVC. (USRSVC will use these entries to generate the macro decoder table and your secondary SVC tables.)

You can call CFISVC from both E-type and CP programs, as well as from ICDF and STPP, to get the address of an entry in any SVC or fast-link table. Use data macro ISV0SV to reference fields in the table entries.

Displaying SVC Codes

You can display any SVC interrupt code in hexadecimal using the ZDSVC command. The display SVC code program returns the SVC code to the CRAS terminal that issued the ZDSVC command.

Dispatching Work

The system task dispatcher, which is also known as the CPU loop, dispatches all activity for a particular I-stream. Each I-stream in a CPC has its own set of CPU loop lists. The system task dispatcher (CPU LOOP) CSECT CCCLHR contains the CPU loop support code. The equate macro, CLHEQ, defines working storage constants necessary for CPU loop list management.

The CPU loop services four dispatch control lists and a number of unique program switches. Processing work items consists of serially scanning the loop and (1) removing a task from one of the lists or (2) finding one of the switches set to indicate that some event that requires processing (such as a clock update request or a keypoint update) has occurred.

Priority scheduling of jobs is not present in the TPF system. In the TPF system, interrupt routines do not call the dispatcher to check for dispatching priorities when

interrupt processing is completed. Interrupt routines always return to the interrupted program, placing the system's priority on work in progress rather than on new work.

The dispatcher is activated when a task either completes or enters WAIT state. The CPU loop lists are processed in the following order:
1. Cross list
2. Ready list
3. Input list
4. Deferred list.

In addition to the four main CPU loop lists, there are two secondary lists. These two lists, which are checked once every pass through all items on the input list, are:
- Virtual file access count (VCT) list
- Suspend list.

The structure of these lists and the processing theory incorporated in their design is discussed in detail in *TPF Concepts and Structures*.

The dispatcher also controls system activity levels. During CPU loop execution, the dispatcher checks the available number of different types of storage blocks against threshold numbers for each block type. TPF sets these threshold limits at system generation time in terms of frames, common blocks, ECBs, I/O control blocks (IOCBs) and system work blocks (SWBs). You can change these values with the ZACLV command (see *TPF Operations* for more details). When the dispatcher checks, if the number of blocks is not in the threshold value, no work is dispatched until additional resources are available.

Managing Storage

Main storage allocation for ECB-controlled programs is a dynamic process of getting and releasing virtual storage. All requests are processed through the storage management CSECT, CCSTOR.

There are two types of storage blocks: logical and physical. *Logical blocks* are carved from 4K *frames*. *Common frames* reside below 16MB and can be shared by application programs.

Application programs normally use logical block types, which include:

Block Type	Use
128-byte block	
381-byte block	Data records Message blocks
1055-byte block	Data records Message blocks
4095-byte block	Data records Keypoint records Program segments Message blocks

The control program normally uses *physical block* types that include:

Block Type	Use
Frames	Working storage for ECBs (carved into 128-, 381-, 1055- and 4095-byte blocks).

Common frames	Working storage that can be shared between ECBs assigned for 128-, 381-, 1055- and 4095-byte blocks).
ECB	Entry control block.
SWB	System work block. SWBs are used as work blocks for the control program, for example, to contain multi-system request blocks used by the Multi-Processor Interconnect Facility (MPIF) feature. See the *TPF Multi-Processor Interconnect Facility Reference*.
IOCB	Input/output (control) block. Only DASD servicing routines use IOCBs.

In each ECB virtual memory is a one-megabyte area of private storage, below 16MB, known as the *ECB private area.* Application programs can issue macros (such as GETCC, RELCC, FINWC, FILEC and EXITC) to get and return storage blocks in this area. This virtual storage is carved from 4K frames of main storage.

ECBs can also get contiguous (or heap) storage in the *heap private area* area above 16MB. Application programs can issue the CALOC, MALOC, REALLOC and FREEC macros (or the `calloc`, `malloc`, `realloc` and `free` C functions) to get and release storage in this area.

ECBs share a pool of working storage below 16MB called the common area. Application programs use the GETCC macro (with the COMMON=YES parameter) to get common blocks in this area.

The control program uses the following macros to get and release storage:

- $GCOMC and $RCOMC to get and release common blocks
- $GIOBC and $RSWBC to get and release IOCBs
- $GSWBC and $RSWBC to get and release system work blocks
- $GETBC and $RELBC to get and release other types of storage blocks
- $GMNBC and $RMNBC to get and release contiguous storage in the heap private area.file

The control program uses the $CONBC and $DISBC macros to connect and disconnect blocks to and from an ECB virtual memory.

The working storage blocks for ECBs are allocated by CTIN and assigned by the OPZERO program and returned to the ECB pool by the EXIT routine (see “Initializing ECBs via OPZERO” on page 15 and “Returning ECBs after Entries Are Processed” on page 18).

IOCBs and SWBs are for the exclusive use of the control program and reside in protected storage. All other blocks reside in unprotected storage.

In an online system, you can display the number of available main storage blocks with the ZSTAT command.

Block Checking Mode

Block checking mode is a debugging tool that flags certain coding errors, such as writing beyond the end of a block, passing blocks chained to other blocks, and using storage that has already been released. When block checking mode is on:

- ECBs run in single block mode. *Single block mode* dispenses a single block in each frame. The block is located in the last block slot available in the frame. Therefore, if a program overwrites the block, there is an increased chance of receiving a page fault for going beyond the end of the frame. This is not foolproof, however. If the subsequent frame is valid in the ECB's address space, a page fault will not occur. Furthermore, L0 (127-byte) blocks are located in the same size block slot as used for L1 (384-byte) blocks, and overwriting these blocks will not result in a page fault until the end of the frame is reached. Single block mode is automatically suspended for an individual ECB if the ECB's available storage falls below 10 pages.
- *Release block processing* disconnects a frame from an ECB if the block being released is the only block in the frame. Subsequent references to the block will result in a page fault because the address is no longer valid in the ECB's address space.
- *ECB exit processing* interrogates each frame it disconnects from the ECB to look for lost blocks. If a block is found to be flagged **in use**, then a CTL-749 system error will occur to indicate that a missing block has been found.

You can turn block checking mode on and off with the ZSTRC command, without re-IPLing.

Note: Block checking mode should be used with caution in a production system, since it degrades CPU performance and depletes working storage.

Managing and Synchronizing Clocks

The control program clock software uses three hardware clock components:
- The time of day (TOD) clock
- The TOD clock comparator
- The CPU timer.

Timekeeping in this system is performed using the TOD clock to drive a set of software clocks. Whenever the system is above 1052 state, TOD clock comparator interrupts increment a control program seconds clock, a systems clock, and a set of subsystem clocks. Software clocks are not incremented when the system is not above 1052 state. The CPU timer is used to verify that ECB-controlled programs release control of the CPU, in a half-second.

Note: Base systems (those that do not include HPO/MDBF) are treated as MDBF systems with a basic subsystem only. For systems that include MDBF, all subsystem clock values, including the basic subsystem, are derived from the system clock.

The system clocks maintained in this system are:
- System seconds clock
- System perpetual minutes clock
- System local standard time (LST) clock
- System last midnight value clock.

The subsystem clocks are:
- Subsystem perpetual minutes clock
- Subsystem local standard time clock
- Subsystem last midnight value clock
- Subsystem Greenwich mean clock
- Subsystem Greenwich mean time midnight value clock.

Commands

A number of commands exist to either display or alter clock and date information. These include: ZDDAT, ZDTIM, and ZATIM. Consult the *TPF Operations* for detailed information on these messages.

Displaying the Time and Date

Clock display facilities can display:

- The system clock
- A subsystem clock
- The system clock and the subsystem clock for each subsystem. The subsystem must be cycled above 1052 state.
- The Time of Day value given by a Sysplex Timer (STR).

  Note: The IBM 9037 Sysplex Timer is part of the IBM Enterprise Systems Connection Architecture.

Date displays are identical to those designed for clocks. Thus, display capabilities include:

- The system date
- A subsystem date
- The system date and the subsystem date for each subsystem. Again, the subsystem must be cycled above 1052 state.
- The date given by an STR.

Altering Clock Values

Periodically, the need arises to alter various system and/or subsystem clock values. In TPF, the capability exists to alter either: a subsystem time clock, or the TOD clock. Altering the TOD clock will initiate an automatic change to the system clock as well.

Considerations for TOD Clock Alteration

- The system must be in 1052 state.
- When operating under a supported version of VM, issuing the ZATIM command does not alter the VM TOD clock.
- Subsystem clocks are altered when the subsystem is cycled above 1052 state.

Considerations for Subsystem Clock Alteration

1. A subsystem may be in any system state (NORM,1052,etc.) when a subsystem clock is altered.
2. Altering a subsystem clock has no effect on the TOD clock, the system clock, or any other subsystem clock.
3. Issuing the ZATIM SET option to alter a subsystem clock in 1052 state, does not change the subsystem date from the date when the subsystem was last above 1052 state.
4. Issuing the ZATIM ADD option to alter the subsystem clock in 1052 state, the system calculates what the time would be if the subsystem was above 1052 state (including any date change) and then updates the subsystem clock as requested.

Restart Facilities

During system restart, the clock program verifies that the TOD clock is operating. If it is not operating, the operator is prompted for a ZATIM command to set the TOD clock. When the TOD clock is operating or after it is started, the system generates the following message to inform the operator:

```
CLKS0001I   HH.MM.SS  SYSTEM CLOCK IS NOW SET
```

Cycle Facilities

Entering the ZCYCL command to cycle a subsystem above 1052 state results in a clock program test of the midnight boundary. If the subsystem crosses a midnight boundary since the last time it was above 1052 state:

1. A message is sent to inform the operator
2. The operator is prompted with a second command, ZATME, before cycling the system. The operator can choose one of 3 options:
   a. Cycle-up the subsystem with the new date - ZATME GOOD
   b. Alter the subsystem time to maintain the old date and then cycle-up the subsystem - ZATIM SET followed by ZATME GOOD
   c. Cancel the cycle request - ZATME CNCL

The subsystem calendar is initialized each time a subsystem is cycled above 1052 state. The subsystem perpetual minutes clock is used to determine the correct date.

Time-Initiated Functions

Using the CRETC macro, real-time programs activate other programs at a later interval of time. The system calculates the clock value when the program is to be activated, and transfers control to the program at that time. (Refer to the section entitled “Initializing ECBs for Entries from Create Macros” on page 15.)

CRETC Considerations

1. For subsystems in 1052 state, CRETC requests are ignored if the subsystem clocks are not operating, unless the STATE=1052 option is specified.
2. CRETC macro requests with the seconds option are activated when the system seconds clock reaches the calculated time. Thus, an alter time request has no effect on the activation of the program.
3. CRETC macro requests with the minutes option are activated when the subsystem minutes clock reaches the calculated time. Thus, if the subsystem clocks are altered, such a request may be activated at a time interval other than had been specified in the request.
4. When a subsystem is cycled-down, all pending CRETC macro requests for the subsystem are purged unless the STATE=1052 option was specified.
5. ECBs created by the CRETC macro are dispatched on the I-stream on which the CRETC was issued.
6. ECBs created by the CRETC macro with the STATE=1052 option specified are honored at the time requested regardless of the system state.

Timekeeping Considerations for Loosely Coupled Processing

- Native processors that are loosely coupled require either the TOD clock synchronization RPQ or an STR to perform TOD clock synchronization for multiple processors. ***Clocks must operate with the same time value in all processors***.
- Control program synchronization of the system clock is performed:

  - During system restart
  - Whenever the operator alters the TOD clock.

  If the satellite processors in a loosely coupled system are in 1052 state when the time is changed on the master, the TOD clocks on the satellite processors will automatically be re-synchronized.
- Altering one processor's subsystem clocks, results in the same change(s) to the subsystem clocks in all other processors.

Synchronization Check Error

A synchronization check error (referred to as *synch check* in *Messages (System Error and Offline)* and *Messages (Online)*) is a condition detected by the hardware that indicates 2 clocks are not in synchronization when they are supposed to be. Clocks are either reference points sending timing signals (*master* clocks) or are remote points receiving timing signals (*remote* or *slave* clocks). If an STR is the synchronization source, then all the clocks in the complex are considered remote. Otherwise, a CPC is the synchronization source. Each remote clock is either in synchronization with the master synchronization source or it is in error. This error is known as a *synch check*. There is no functional connection between remote clocks. However, there is a functional connection between the master synchronization source and the remote clocks.

Hardware features determine the relationships between clocks. Control register 0 contains a bit that determines whether the clock on that processor will be sensitive to the timing pulses sent to it. If this bit is not set, any stepping pulses sent to it will be ignored. If the bit is set, a stepping pulse every microsecond is expected. If the TOD RPQ is used, the master clock ignores timing pulses, while remote clocks expect the timing pulses. If an STR is used, all clocks are remote and expect timing pulses. Control register 0 also contains a bit that determines whether *synch checks* will be recognized. If this bit is set, *synch checks* will be recognized, otherwise not. Recognizing a *synch check* means that at the moment the remote clock reaches the one second mark it expects a synchronization pulse to be sent by the master clock and, if this pulse doesn't occur at that moment, an external interrupt will take place on the remote processor. A *synch check* is a catastrophic condition under most configurations.

This logical picture of clocks in synchronization is the same whether the system consists of standard 370 hardware, STR hardware, or a TOD clock RPQ enhancement. The standard hardware maintains the synchronization in a CPC. The STR hardware or the TOD RPQ hardware maintains the synchronization between CPCs. The standard hardware, the STR hardware, and the TOD RPQ can be involved in loosely coupled complexes of tightly coupled CPCs.

When a *sync check* occurs, the external interrupt handler is invoked to determine which error has occurred. When the sync check is isolated, TOD clock synchronization routines are called to attempt to resolve the error. If the processor in error is a remote processor in a loosely coupled complex with the TOD RPQ or an STR, hardware revalidation is attempted. If the hardware is operational, the processor is not taken down. All other scenarios are catastrophic. After a catastrophic *synch check* the system reloads and the time must be confirmed with an external source. If a *synch check* is not catastrophic, the CPC will be *locally synchronous*, in the loosely coupled complex. To restore the CPC to the synchronization of the complex, the time should be reset on the master CPC, or the CPC must be reIPLed.

Altering and Displaying Data and Programs

TPF allows you to alter and display:

- Main storage (by address or label)
- Data or program records (by file address)
- Fixed file records (by record type and ordinal number)
- Pool file records (by pool record type and ordinal number)
- File-resident programs (by program name) except for ISO-C programs
- Main storage-resident programs (by program name) except for ISO-C programs
- Link map data in C load modules (display only)
- Entries in the program allocation table (by program)
- System generation values
- Resource control values
- System allocation values in keypoint record A (CTKA).

You can use various commands to alter or display programs or data. When altering data, you can include validation data in the message itself to ensure that you are altering the appropriate data. (If the validation data does not match, no alteration occurs.) Data is displayed as both hexadecimal codes and printed characters, and altered data is shown both before and after alteration. The amount of data that can be altered or displayed varies by command and generally starts on a fullword boundary. Following are summaries of the commands for altering and displaying data and programs. For more information see *TPF Operations*.

Altering and Displaying Main Storage

You can use 2 sets of messages to alter and display main (*core*) storage. For ZACOR and ZDCOR, you specify a main storage address. For ZADCA and ZDDCA you specify a dump label. In a tightly coupled system you can specify an I-stream.

ZACOR Alter main storage by specifying a system virtual address and display storage before and after alteration.

ZDCOR Display storage by specifying a system virtual address. The display starts on a fullword boundary unless you specify disassembled data which starts at the specified address.

ZADCA Alter main storage by specifying a dump label and display storage before and after alteration.

ZDDCA Display the address of a specified dump label or main storage by specifying a dump label.

Displaying Link Map Data in C Load Modules

You can display the link map contained in a C load module by using the ZDMAP command. The link map consists of a list of object files included in the C load module, a list of C function names in the object files, and the addresses of the object files and C functions.

Link map displays include both main storage addresses and the offsets of C functions into their respective object files. Any time an object file name or function name is displayed, its address is also displayed.

See *TPF Operations* for more information about the ZDMAP command.

Altering and Displaying File Records

You can alter and display any file records using ZAFIL and ZDFIL or fixed file records using ZAREC and ZDREC.

ZAFIL Alter a file record by specifying the file address.

ZDFIL Display or print a file record with a specified file address. You can also display a chain of as many as 33 addresses from the specified record or print a chain of as many as 33 records.

ZAREC Alter a fixed-file record or pool record.

ZDREC Display a specified number of bytes from a fixed-file record with a specified record type beginning at a relative starting address or display the file address of the record.

Other commands for altering or displaying files include:

ZIFIL Initialize a fixed file data record by specifying a FACE equate value, a record ID, record code check byte, and starting and ending ordinal numbers for the initialization.

ZDADD Display the file address for a specified record type and ordinal number.

Altering and Displaying Programs, by Program Name

You can alter and display programs (except for ISO-C programs) using ZAPGM and ZDPGM.

ZAPGM Alter the file or main storage copy of a file-resident program.

ZDPGM Display the file or main storage copy of a file-resident program.

Altering and Displaying Entries in the Program Allocation Table

You can alter and display entries in the program allocation table (PAT) using ZAPAT and ZDPAT.

ZAPAT Change a program entry in the file or main storage copy of the program allocation table. Changes to the file copy take effect when you re-IPL. If you specify a transfer vector entry, the parent program is changed.

For ISO-C programs ZAPAT can authorize programs to use reentrant static blocks. ZAPAT cannot change the module type. ZAPAT is restricted to CLASS=SHARED for ISO-C programs, dynamic load modules (DLMs) and libraries.

ZDPAT ZDPAT displays a program entry in the program allocation table that includes the program linkage type, class, file address, addressing mode, authorization, and program size, and the base address of its PAT entry in main memory (see the DSECT Program Allocation Table [IDSPAT] for more information). If you specify a transfer vector entry, the entry for the parent program is displayed.

Displaying Program Linkage Types

You can display program names from the program allocation table (PAT) with their program linkage type by using the ZDPLT command.

ZDPLT Display program names from the program allocation table by specifying the program linkage type and, optionally, the program name.

Maintaining a Memory Patch Deck

You can maintain and execute patch decks, which are groups of commands for changing main storage (ZACOR, ZADCA, and ZAPGM).

ZPTCH Maintain and execute patch decks. You can use ZPTCH to build as many as 20 patch decks, each containing as many as 50 messages.

Altering and Displaying System Generation Values

ZSYSG Change and display system generation bits initially defined by SIP. You cannot use this message to change system generation bits that require the reassembling of programs, allocator changes, or the rerunning of SIP if they are changed. Changes take effect on the next IPL of the system.

ZCTKA Change and display storage allocation values. If you are using the multiple database facility (MDBF), you can issue ZCTKA from any subsystem (although some subsystem shared values can only be changed from the BSS). Changes take effect on the next IPL of the system.

Altering and Displaying Resource Control Values

ZSYSL Change and display shutdown levels defined for various block types in the LODIC priority class table.

ZTMSL Add, remove, change, or display time-slice attributes in the time slice name table.

Initiating System Alerts

System alerts consist of 2 functions:

- Long-life ECB detection and removal
- Time-initiated alerts.

Long-Life ECB Detection and Removal

Entries that loop without relinquishing control are removed from the system by the application loop timeout mechanism. However, entries that relinquish control in an indefinite loop are detected by the long-life ECB detection program.

This detection does not occur automatically for all ECBs. Each ECB is created with its CE1LML (maximum permitted lifetime) field set to X'FF', which prevents long-life detection. To enable long-life detection for any given entry, the application program must issue the LONGC macro.

When a long-life entry (in other words, a looping entry) is detected, message ECBL01 is sent to the operator. The operator can request that a looping ECB be scheduled to EXIT by use of the ZECBL command with the E parameter (ZECBL E). The ECB address specified on the ZECBL E command must be a system virtual address. If a looping ECB that is scheduled to exit by ZECBL E command is dispatched again it is exited with system error dump number 0000D3.

If a looping entry scheduled to exit by the ZECBL E command does not exit in 1 minute, it is determined to be *hung;* it has lost control and will not be dispatched again (possibly waiting for an event that will never occur). Hung ECBs cannot be removed from the system, but message ECBL01 is sent to the operator whenever a new hung entry is detected.

The ZECBL command can be used to display all looping and hung entries, or to display an individual entry. The information provided by ZECBL D includes the ECB address, program name and address, PSW address, wait count, record hold count and maximum, and current ECB lifetime.

Time-Initiated Alerts Function

The time-initiated alerts function provides system operators with the capability of automatically starting system functions and maintaining operator communications. The operator adds a message to the time-initiated message table by specifying when the message is to be started and, optionally, the destination of the message. For example, if a user performs disk capture at a particular time every day, a message to remind the operator at the DASD functional support console to start the capture is put into the time-initiated message table. If a system function such as displaying pool counts periodically is required, the command to display pool counts is put into the table.

The contents of the table are controlled by the operator; there are options in the command support to add a message, display a single message or all the messages, delete a message from the table, and reinitialize the table.

The operator can add time-initiated messages or stage-initiated messages to the table. A *time-initiated message* is one that is initiated at a specified time. *Stage-initiated messages* are processed during periods of system operation when time cannot be used to control the processing. System stages have been defined to identify these periods:

- Beginning of restart
- End of restart
- End of cycle-up
- End of cycle-down
- When NORM stage is reached.

Messages are processed when a stage is entered or when a stage is left.

The time-initiated message table is a fixed file record (FACS record type #TIMRI); there are no overflow records. The record size is variable and is determined by the user based on estimates of how many messages will be in the table at any one time, and the lengths of the messages. The table record has a 28-byte header, and each message in the table record has an 18-byte header. The table record is defined in the DSECT macro TI0MT.

In a loosely coupled complex, there is 1 record per processor; the record ordinal number is the same as the processor ordinal number. In a multiple database function system, there is 1 record associated with each subsystem. It is not necessary for records to be allocated for all subsystems in the system. The records must be allocated at system generation time by coding the appropriate RAMFIL macros. If the records are not allocated, it is assumed that the time-initiated alerts function is not active, and processing is not performed for it.

Command Support

The command is ZSTIM. The add, display, cancel, and reinitialize options are described in detail in *TPF Operations*.

Initializing the Program Allocation Table

The system allocator creates 2 tables from SALO input files: the offline *system allocator table* (known as the SAL table) and the online *program allocation table* (PAT). There is an entry in each table for every allocated program. The TPF linkage editor uses the libraries to resolve external references (VCONS) in object modules. Because this process occurs in an offline environment, the SAL table is strictly an offline structure. The PAT is used online by Enter/Back routines. It contains the addresses of all E-type programs as well as other allocation information. The PAT resides in the core image restart area and is brought into main storage by IPLB. Some fields in each PAT entry are initialized by CTIN. There is a PAT in main storage and the core image restart (CIMR) area for each subsystem.

If you need to display allocation characteristics of E-type programs, use the ZDPAT command. If you need to change allocation characteristics of E-type programs, use the ZAPAT command. See *TPF Operations* for more information about the ZDPAT and ZAPAT commands.

Using Control Program User Exits for User Functions

To allow TPF system users to perform processing that is unique to the user's operation, a set of predefined *user exit points* have been provided in the TPF control program. These user exit points (when activated) will cause control to be passed to user-supplied routines. See *TPF System Installation Support Reference* for additional information.

High Performance Option

The *High Performance Option* (HPO) is a licensed feature available to TPF users. HPO is composed of 2 subfunctions:

- The multiple database function (MDBF), and
- The loosely coupled (LC) facility.

The *multiple database function* (MDBF) enhances portability, protection, and sharing of system resources. It consists of 2 distinct, yet related, operating modes; subsystems (SS) and subsystem users (SSU). Users can configure as many as 64 subsystems or 127 subsystem users. The combination of subsystems and subsystem users cannot exceed 128. Each subsystem owns independent disk packs (DASD) while sharing control program services with other subsystems. For ease of control, one subsystem contains all the system related software. This subsystem is called the basic subsystem (BSS). Base TPF systems (without HPO) consist of a single BSS with no subsystem users.

Both subsystems and subsystem users contain E-type (online) programs, V-type (offline) programs, and a database. A database can consist of fixed files, pool files, user global area(s), application control data, and system control data.

There are several different variations of MDBF systems, ranging from one basic subsystem with one or more subsystem users, to multiple subsystems with multiple subsystem users.

The loosely coupled (LC) facility provides the potential for 8 processors to access a shared database while presenting the outward appearance of a single processor. Database sharing is accomplished by one of 2 hardware RPQs: the limited lock facility (LLF) or the concurrency filter lock facility (CFLF). CFLF is a companion feature to the 3990 Storage Control Multi-Path Record Cache RPQ. A Sysplex Timer (STR) or a second RPQ (necessary for interprocessor clock synchronization) is also required in a loosely coupled system.

Note: The IBM 9037 Sysplex Timer is part of the IBM Enterprise Systems Connection Architecture.

Controlling Loosely Coupled Processors

Loosely coupled processors require control functions that are not needed in a single processor environment. The programs that perform these functions are referred to collectively as *processor status management services* (PSMS). These programs:

- Maintain the same system state for all processors in the complex. This is necessary during the processing of certain system utilities.
- Enable and disable system state change.
- Display the status for:
  - A processor
  - All subsystems within a processor
  - A single subsystem within all processors.
- Deactivate processors.
- Activate and deactivate subsystems.

PSMS programs are resident in the basic subsystem in a multiple database function (MDBF) system. PSMS is activated with the ZPSMS command.

Cross Subsystem Access Services

In an MDBF system, system programs need support facilities to access the program or database of subsystem users in the complex. Under MDBF, each subsystem maintains its own database, which is identified by a *subsystem database ID* (DBI). All subsystem users within a particular subsystem have the same DBI. During IPL each *subsystem* (SS) and *subsystem user* (SSU) is assigned an ID.

Cross subsystem access service routines use the program base ID (PBI), database ID (DBI), and subsystem user ID (SSU ID) to service MDBF macro requests. Normal changes to the DBI and PBI are from the basic subsystem (BSS) to a subsystem (SS), or vice versa. System programs should use only MDBF macros to change and manipulate these fields. Application programs should not use MDBF macros. MDBF macros include:

CROSC CROSC permits one subsystem to access another subsystem's data or program base. To access programs or global data, invoke the CROSC macro in either the ECB virtual memory or the system virtual memory. To get pools from another subsystem, use CEBIC to change the DBI and GETFC to get the pool.

UATBC Use the UATBC macro to locate information pertaining to a particular subsystem. UATBC calculates and returns the address of a subsystem user from a list of addresses in the subsystem user table (SSUT). UATBC then uses the SSUT to access specific subsystem user data.

CEBIC To access the database of any other subsystem, use the CEBIC macro to change the DBI and SSU ID in an active program's ECB. The DBI and SSU ID are preserved across CEBIC calls. Return is made to the original subsystem and/or subsystem user.

LEBIC Several tables are generated in an MDBF system. These tables are accessed by using either the subsystem or SSU ID. To locate the proper ID from a list of identifiers requires an index into the list. The LEBIC macro standardizes the retrieval of these index values. Use the LEBIC macro to obtain the index value for a specified ID, validate the ID, and convert a specified subsystem or SSU ID to a subsystem/SSU ordinal number.

Both CP segments and E-type programs can use CEBIC and LEBIC, however, different input restrictions apply. See *TPF General Macros* and *TPF System Macros* for these restrictions and more information on all the MDBF macros.

Owning Resources in a Loosely Coupled System

Hardware and software resources in a loosely coupled system are switchable, shared, or unique by processor:

- Switchable resources can be dedicated to any one of the processors. Tape drives are an example.
- Shared resources can be shared concurrently among all processors. Certain storage devices are an example. Shared resources can also refer to resources that are used by more than one subsystem or subsystem user.
- Processor unique resources cannot be shared or switched. Processor unique keypoints are an example.

Loosely coupled processors can dynamically acquire and release ownership of switchable hardware and software resources. The processor resource ownership facility (PROF) controls this dynamic ownership and maintains the status of tape drives and system utilities. *System utilities* are programs that perform repetitive everyday tasks, such as TPF capture.

Data Records Unique to Loosely Coupled Processing

Two tables support loosely coupled processing: the processor resource ownership table and the processor status table.

The processor resource ownership table (PROT) maintains the ownership status of tape drives and system utilities. You can use the ZPROT command to assign, release, and display ownership of resources. Although the PROT maintains tape drive ownership, the tape control program actually assigns and releases ownership. See *TPF Operations* for information on the ZTVAR command.

The system test compiler creates the PROT offline. You can load the PROT to the online system with the ZSLDR command when the system is in 1052 state.

The PROT consists of fixed file records of record type #PRORI. There is one record for tapes that contains space for a maximum of 257 entries. One or more records can be generated for system utilities; each record can contain as many as 102 entries.

The status of each processor is maintained in the processor status table (PIDT). TPF generates one PIDT for each processor in a loosely coupled complex. The PIDT is a main-storage resident record.

Interprocessor Communications (IPC)

Interprocessor communications (IPC) allows you to move data between loosely coupled processors using the Multi-Processor Interconnect Facility (MPIF) licensed feature and channel-to-channel (CTC) communication links. The MPIF path active exit establishes a MPIF connection across each CTC communication link between processors. The system whose name is first in the alphabetic generally initiates connection processing.

IPC maintains an internal control block structure. The IPC global table (IGT) contains control information and one entry for each processor in a loosely coupled complex. Each entry contains data collection counters and a chain of IPC connection definition blocks (ICDBs) mapped by data macro DCTICD. An ICDB is established for each IPC connection between processors. For example, if 3 CTC links exist between processors A and B, the IGT entry for B in processor A contains a chain of 3 ICDBs. The ICDB contains the connection token, the identify token (IDTOK) for the resident system, and a pointer to the relevant IGT entry. CCCTIN allocates the IPC control block area based on the maximum number of links between loosely coupled processors in keypoint record E (CTKE) and places the address of the IGT in the CINFC table.

Restart

IPC restart:

1. Initializes the IGT and ICDBs and initiates processing to establish communication with other processors.

2. Uses the CINFC macro to locate the IPC control blocks and CTKE to determine the maximum number of connections allowed between processors.
3. Initializes the IGT, ICDBs, and ICDB block list. The ICDB block list contains addresses of available ICDBs. The number of ICDBs allocated is the product of the number of processors in the complex and the maximum number of connections between processors.
4. Uses the MPIFC macro to identify itself to MPIF.
5. Places the IDTOK in the IGT.
6. Uses MPIFC QUERY to query all MPIF users named IPC. The QUERY reply area contains the hardware CPUID for each system.
7. Scans CTKI for a match of hardware CPUIDs. A match is found if the system is located in the loosely coupled complex.
8. Initializes the appropriate entry in the IGT.
9. Issues MPIFC CONNECT across each active path if the resident system name is alphabetically ahead of the other system name.

Sending Data

The SIPCC macro invokes the staging and transmit function to pass data to other processors within a loosely coupled complex. This function then:

1. Verifies that the requested destination processor is active.
2. Selects a connection to send the data over for each active processor. Connections are selected on a rotational basis to balance loads.
3. Maintains a pointer to the next connection to send over in each IGT entry.
4. Builds the send parameter list, the IPC 24-byte control area, and data area 1 in a system work block.
5. Copies data area 2 to a core block for each destination processor.
6. Assigns an IPC sequence number to the IPC item. IPC uses this number to keep IPC items in order in the receiving system.
7. Executes MPIFC SEND for each destination processor. If no connections exist to the remote system, the IPC item is queued. If the IPC sequence number is about to roll over to zero, the IPC item is queued until restart notification is received from the remote system.

The SIPCC macro passes IPC items to destination users in sequence. The MPIF data received exit queues items until they can be presented in sequence. The sending system halts transmission when its send sequence number reaches the largest positive value (X'FFFFFFFF'). When the receiving system has received all pending messages, a message (with sequence number zero) is returned. Transmission then begins with send sequence number 1.

Receiving Data

MPIF activates the receive function when data is received from IPC. Receive is also activated following a successful send if Device End notification was requested in the send parameter list.

If the data received exit has been activated for Device End notification, the receive function places the IPC item on the ready list to be passed to the IPC user by the IPC post interrupt routine. Otherwise, receive checks the sequence number and if the received item is next in sequence places the item on the ready list for IPC post interrupt processing. If the received item is not next in sequence, receive queues it for resequencing. Resequencing occurs when an in-sequence item is received. If

the receive sequence number is about to roll over to zero, receive sends a restart message to the remote system. This message (with an IPC sequence number of zero) informs the remote system to send IPC items again. Receive then releases the block containing the data received exit parameter list.

Timing Out

This function returns IPC items to the sender when data is not transmitted within the required time period. The timing function scans the IGT for remote systems that don't have connections to the local system. When such a system is found, timing decrements the time-out count for the items on the output queue. When the count reaches zero, timing sends a message to the operator. If the SIPCC return option was specified, each queued item is returned to the sender. Otherwise, the core blocks are released.

Displaying and Altering IPC Information

The ZSIPC command allows the operator to display and alter the IPC status, time intervals, number of IPC paths, class of IPC paths, and transmit or receive counts to any or all processors.

ZSIPC DISPLAY displays status and processor counts. The STATUS display shows the restart connection time-out value, the amount of time IPC will wait, the maximum number of IPC paths, and the IPC path class. The COUNTS parameter displays a processor's total activity and a breakdown of receiving processor activity.

ZSIPC ALTER alters the IPC timing values, the maximum number of paths between processors, the path class IPC uses, and resets the IPC counters. You can use the INTERVAL parameter to modify IPC time values. Use the TIME subparameter to set the length of the IPC interval and the TOUT subparameter to set the number of IPC intervals to wait before timing out.

Performance Considerations

Increasing the number of connections defined for IPC to use between processors improves IPC performance. Also, you can dynamically tune IPC for optimum system performance by altering the IPC timing values.

Cross References

For more information on IPC see:

- *TPF Multi-Processor Interconnect Facility Reference*
- *TPF General Macros*
- *TPF System Macros*
- *TPF Operations*

Error Recovery

Errors can occur at any time, in any area of processing. They may be software errors, hardware errors, or both. *TPF error processing* is concerned primarily with the speed of recovery; errors must not allow processing to be interrupted for a prolonged period of time.

Processing System Errors

System errors are errors which occur in software. Software error recovery in the TPF system is known as system error processing. There are 2 types of system errors:

- Error processing is purposely initiated by a TPF program when an unusual condition is detected, such as an invalid macro parameter or an error on an I/O device. These errors are known as *soft* errors because they are detected by software.
- Error processing is initiated in response to a program check resulting from a software logic error. These errors are known as *hard* errors because they are detected by hardware.

In both cases the philosophy of system error processing is the same:

- Send a message to the prime CRAS informing the operations staff that an error occurred. If the error occurred in an application program activated in response to an input message, send the message `CHECK DATA CALL SUPERVISOR` to the originating terminal.
- Generate a full or partial memory dump of main storage.
- Determine the appropriate recovery action.

The primary diagnostic aid in the TPF system is a main storage dump, formatted to facilitate identifying relevant storage locations. You can control which storage areas are included in different dumps, in order to limit their size. This is done by associating keywords with areas of storage, and mapping a given system error number to a list of these keywords.

The SNAPC and SERRC macros initiate system error processing procedures for soft errors. The following macros are used to control dump content:

IDATG Associates keywords with areas of storage.

IDATB Contains IDATG calls, and builds the *selective memory dump table (SMDT)*

IDOTB Codes dump overrides, and builds the *static override bitmap tables.*

The relationship of these macros and tables is discussed in “Controlling the Content of System Storage Dumps” on page 41 and shown in Figure 1 on page 42. See the *TPF System Macros* for details on how to code the IDATG macro.

You can use the ZASER and ZDSER commands to alter and display system error processing options and to activate the dump data user exit (whether and where the dump will be written, where to route messages, and so on). The ZIDOT command enables you to create additional keyword-to-storage-area mappings, and to override IDOTB calls. ZIDOT commands create entries in the *dynamic override bitmap table (DOBT).* These commands are described in detail in *TPF Operations*.

Types of System Errors

System errors can occur either in application programs, or in the TPF control program.

Application Program Errors

Application program errors are errors in logic detected during processing. These errors relate to an application, that is, an ECB. The application detects the error and chooses the necessary corrective action. When an application detects an error, it can either:

- Ask the control program to exit the ECB via the EXIT routine, or
- Resume control and process the error as it deems appropriate.

Either the SNAPC or SERRC macro tells the control program which option the application requests.

Control Program Errors

Control program errors may or may not be associated with an ECB. These errors occur for these reasons:

- Violation of the rules of the TPF system. For example, a GETCC macro is issued to retrieve a storage block on a data level that is already holding a block.
- Violation of hardware rules. These can be subdivided as follows:
  - A software logic error causes the program new PSW to be loaded, for example, a hard error occurs.
  - A hardware operation fails because of erroneous software input. For example, an I/O operation is initiated to retrieve a record, and the device address specified is wrong. The hardware then reports the error back to the software for it to process.

The system operator can also request a control program dump with the ZDUMP command, or by depressing the system restart key (O2 on the OPRCTL frame). This is referred to as a *manual dump*.

Initiating System Error Processing

Both the SNAPC and SERRC macros initiate system error processing; they differ only in the amount of diagnostic information they produce, and the amount of online resources they require.

The SNAPC macro

The SNAPC macro is used whenever the cause of the error condition is well understood, and less than 32K of data is required to fully document the problem. The SNAPC macro only dumps areas of main storage that the programmer has specified using LISTC macros (refer to the *TPF General Macros* for details on LISTC).

You can use the ZASER command to specify whether to route a snapshot dump to a tape or printer device, and how much of the snapshot dump information to display on the CRAS terminal. Note that routing a snapshot dump to CRAS generates extra ECBs, because of the use of WTOPC.

SNAPC uses far fewer processor resources than SERRC. Only the I-stream on which the error occurred is used to generate the error data; the other I-streams in the CPC are not affected.

You can use SNAPC for database corruption errors, hardware errors, or any other errors where you understand the cause of the problem and need less than 32K of data to fully document it.

The SERRC Macro

The SERRC macro always produces a standard memory dump of main storage. There are 2 types of SERRC dumps, one for application program errors and one for control program errors:

- An OPR dump, in which the error was detected by an *operational* program (application program);
- A CTL dump, in which the error was detected by the control program.

The first section of a dump is identical for both types of SERRC dumps. It contains information describing the system configuration, the status of each I-stream in the configuration, and the system trace tables for each I-stream. If there is an ECB associated with the error, a dump of the ECB virtual memory will follow, including all of the data objects that are logically attached to the ECB. An OPR dump ends at this point.

In a CTL dump, the dump of the ECB virtual memory (if any) is followed by a dump of system storage. This dump begins at location X'1000' and continues to the end of the system virtual memory. This dump is limited to a reasonable size through the use of pre-defined dump keywords, which identify large areas of storage that normally are omitted from dumps. One or more of these areas can be included in the dump for a particular system error if a *dump override* has been defined for that system error. See Controlling the Content of System Storage Dumps.

The format and content of the TPF system dumps is discussed in greater detail in *TPF Program Development Support Reference*.

Hard Errors

Four system error numbers are reserved exclusively for the purpose of identifying program checks that result from software logic errors. They are:

- CTL-I000001. Identifies a program exception which occurred in the SVM. This is a catastrophic error.
- CTL-I000002. Identifies a segment- or page-translation exception in the SVM. This is also a catastrophic error.
- CTL-I000003. Identifies a program exception which occurred in an EVM. The failing ECB is exited.
- CTL-I000004. Identifies a segment- or page-translation exception in the EVM. The failing ECB is exited.

Note that segment- or page-translation exceptions which occur in supervisor state are not necessarily catastrophic errors. If the error occurred in an EVM, the error will be converted to a CTL-I000004 and the ECB will simply be exited.

Controlling the Content of System Storage Dumps

TPF dumps system storage under the following conditions:

- There is no ECB associated with an error.
- There is an ECB associated with an error, the SERRC macro is coded in an ECB-controlled program, and SYSDUMP=YES has been coded on the SERRC macro call.

- There is an ECB associated with an error, the SERRC macro is coded in an ECB-controlled program, and ZASER LONG is in effect.
- There is an ECB associated with the error, and the SERRC macro is coded in the control program.

You can control the contents of your SERRC dumps by selecting which large areas or tables you want in certain dumps. There are 2 steps to this process:

1. Defining areas of storage that are normally excluded from dumps. This is done by defining *keywords* and associating these keywords with areas of storage.
2. Coding *dump overrides* is done by mapping system error numbers to keywords.

The relationship of the macros and tables described in the following sections is shown in Figure 1.

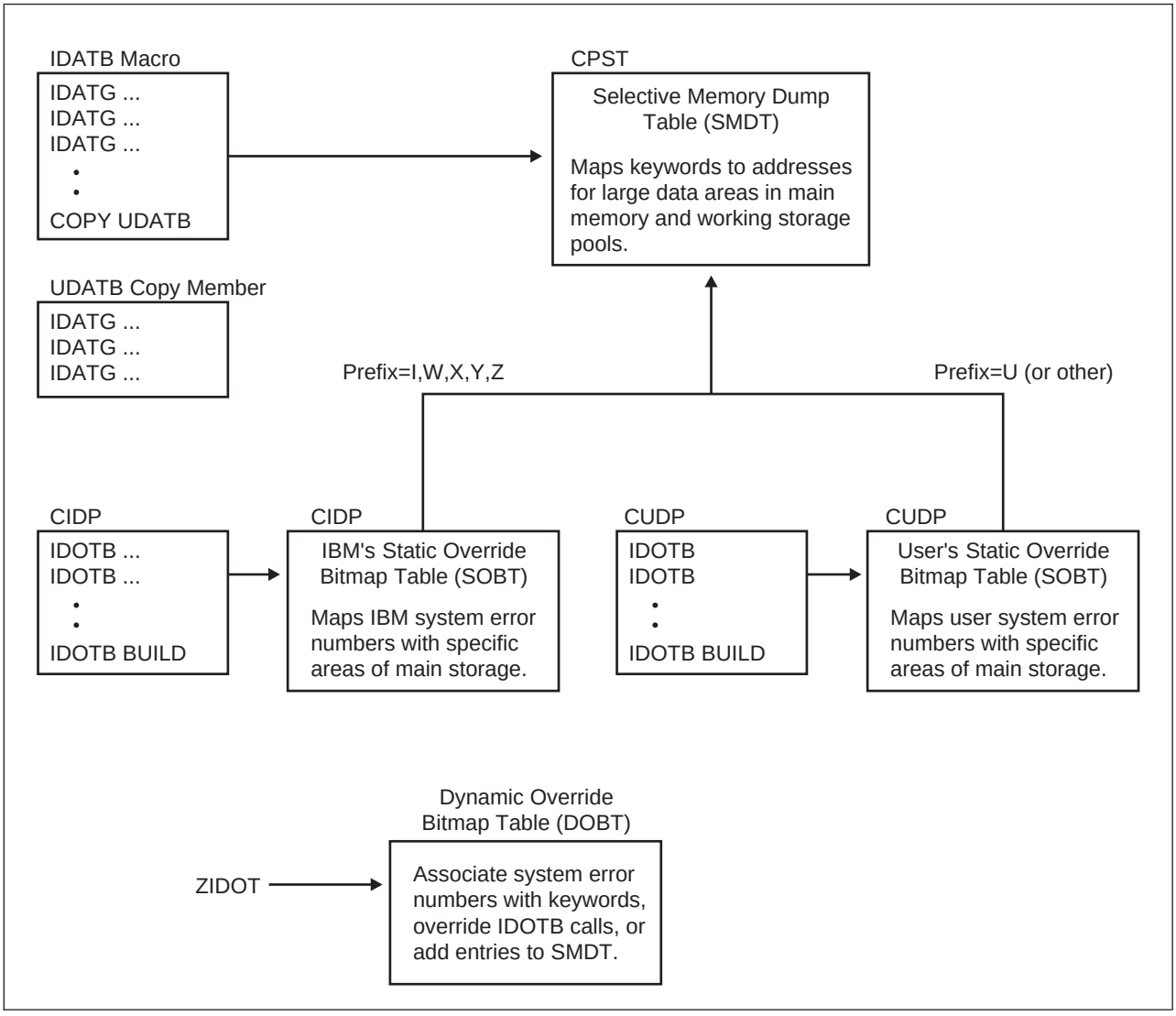

Figure 1. Components of Dump Content Control

Defining Keywords

IBM ships a list of *predefined keywords* for large IBM storage areas in the IDATB macro. These keywords are defined using IDATG macro calls. When IDATB is assembled, these calls generate entries in the selective memory dump table

(SMDT). Entries in the SMDT represent data areas that are normally *excluded* from dumps. All of IBM's predefined areas start with **I** for example, ICLH represents the CLH block management tables.

The IDATB macro contains a call to copy member UDATB, which contains *user* IDATG calls, for customer use. You can code your own IDATG calls to associate keywords with regions of storage that you want to include in (or exclude from) dumps using the IDATG calls in IDATB as an example. Include your IDATG calls in UDATB.

Coding Dump Overrides

The areas defined in the SMDT will not be included in dumps unless a *dump override* is created for a particular system error requesting that a storage area be included. The IDOTB macro maps specific system error numbers to keywords, creating dump overrides. IBM's dump overrides are coded on IDOTB macros in segment CIDP. When CIDP is assembled, these calls generate entries in IBM's static override bitmap table (SOBT).

You can code your own dump overrides using IDOTB calls in copy member CUDP of user exit CSECT CCUEXT using IBM's IDOTB calls in CIDP as an example. Your IDOTB calls will generate entries in the user SOBT.

Once the system is up and running, you can use the ZIDOT commandto associate system error numbers with keywords or to override what's coded on IDOTB calls.

Coding Prefixes

You can code a *prefix* on IDOTB calls to distinguish between the system error numbers you created and those provided by IBM. IBM reserves prefixes I, W, X, Y, and Z for its own use. You can use other prefixes to distinguish between, or to group, system errors. If you do not code a prefix, it defaults to U. The prefix determines which dump override tables the TPF system will refer to when a system error occurs. (There is one exception to this: the TARGET(TPF) functions `exit` and `perror` cannot accommodate a prefix character, so a single set of system error numbers is used by programs provided by IBM and developed on your site.

If you want to use the version of SMDT that IBM ships with minimal tailoring, but you want to include areas that are excluded by default (such as the global areas), you can do one of the following:

- Remove the IDATG call from the IDATB deck that excludes the area you want included, or
- Code an IDOTB macro to override the entry (excluding the area) in the SMDT for certain system errors, and include it in CUDP.

All IBM SERRC calls are prefixed with the letter I. The SERRC service code will supply a default prefix of U for all other code. When you use ZIDOT to include or exclude specific areas in your dump, you must include the prefix. If you are allowing the prefix to default to U, code this prefix on the ZIDOT command or the IDOTB macro.

Determining Appropriate Recovery Action

After generating appropriate diagnostic data, the system error routine examines the error options specified by the program that detected the error. These options indicate which recovery technique is needed. Recovery alternatives are:

- Return to the program that detected the error
- Remove the entry associated with the error via the EXIT routine
- Transfer control to the catastrophic error recovery routine.

When a catastrophic (unrecoverable) system error occurs, the present condition of the system is examined to determine the extent of the recovery procedures. The basic catastrophic recovery options are:

- Return to the CPU loop
- Cycle-down, software IPL (restart), cycle-up to the previous state
- Software IPL (restart) and cycle-up (no cycle-down is attempted)
- Cycle-down, software IPL (restart), cycle to 1052 state.

If additional program errors occur during catastrophic processing, the control program may decide that further processing is impossible. In this situation no attempt will be made to re-IPL the system; instead, a disabled wait PSW will be loaded. The 6-digit code in the address field of the PSW provides an indication of the nature of the error. These error codes are documented in *TPF Operations*.

Channel Check Handling

A *channel check* is an error associated with the channel subsystem, or the devices attached to it, that is reported in the interrupt response block (IRB) by the Test Subchannel (TSCH) instruction. There are 3 channel check conditions:

- *Channel control check* (CCC) is caused by any machine malfunction affecting channel-subsystem controls.
- *Channel data check* (CDC) indicates that an uncorrected storage error was detected related to data contained in main storage that is currently used in the execution of an I/O operation.
- *Interface control check* (IFCC) indicates that an incorrect signal occurred on the channel path.

See *ESA/390 Principles of Operation* for more information about channel check conditions.

TPF handles channel checks by logging them on the RTA or RTL tape and tracking how often they occur. The channel check handler (CNCEX1) issues message CNCE0001E and cycles down the TPF system when the fault-rate threshold is exceeded. IFCCs are not tracked because they originate outside of the channel subsystem; tracking them would allow a defective device to cause TPF to automatically cycle down.

The channel check threshold is defined in subroutine EFL000 of segment CMKH of CSECT CCMCKH.

Machine Check Handling

Machine checks are rare, and when they do occur they typically report that a device has been added to or deleted from the I/O configuration. However, when they occur, you must be prepared to handle the full range of machine check conditions the *ESA/390 Principles of Operation*.

The TPF machine check handler consists of:

- CPPMKH — the machine check first level interrupt handler (FLIH)
- CMKHRDO — the machine check second level interrupt handler (SLIH)
- CNCEX9 — the CRW handler
- CPPLMI — the event information handler
- CZSA — the Sysplex Timer machine check handler
- CPPMAL — the check stop handler.

Checking System Internals

Checking system internals is an essential function of a properly operating TPF system. The following section describes some of the methods used by the TPF system to perform this checking.

I/O Trace

The common I/O routine creates the I/O trace table IDSTTR, and traces all I/O activities continuously. The I/O trace table contains the following I/O interrupt information, except when requested via macros SPNDC and RESMC:

- System device address
- Subchannel number
- SVC trace pointer
- I/O old PSW
- SCSW from IRB.

For SPNDC and RESMC macros, the I/O trace table contains:

- Indicator for SPNDC/RESMC macros
- Type indicator for SPNDC/RESMC macros
- SVC trace pointer
- Caller's return address.

An I/O trace table example can be found in *TPF Program Development Support Reference*.

Displaying Online Dump Tag Addresses

Dump tags are labels attached to significant storage locations when system error dumps are created. This program displays online the storage address for a given dump tag using the tag itself as input. Dump tags are retrieved in blocks from file. The ZDDCA command activates this program. See *TPF Operations* for the format of this message. See *TPF Program Development Support Reference* for a listing of all system dump tags.

System Maintenance

TPF provides both offline and online segments to collect, edit, and print hardware error and trace data. The online segments:

- Record environmental data
- Close log entries
- Log error entries
- Log 37x5 related data records.

The restart program activates AMX2 during a cycle-down of the system. AMX2 scans the SON file status tables to locate the online disk packs. AMX2 reads the environmental data from the buffer of these packs. All of this information is passed to segment CYSA. CYSA initializes these closing log entries and passes control to CYSM. CYSM logs the data to a real-time log tape. CYSM also logs device error records for DASD.

The 37x5 communications controller is a programmable control unit. It performs the line handling and processing functions in a communications network. The network control program (NCP) regulates operation of the 37x5.

The offline segments:
- Record, edit, and print environmental data
- Process device error statistics.

The Environmental Recording, Editing and Printing (EREP) Postprocessor

Operating under MVS control, the EREP interface postprocessor uses the real-time log tape to create 2 distinct output files; work tapes, and history tapes. All TPF device errors are written to work tapes. All other device errors are written to history tapes. History tapes are compatible with the format of the EREP accumulation input data set (ACCIN). See *Environmental Record Editing and Printing Program (EREP) User's Guide* and *Environmental Record Editing and Printing Program (EREP) Reference* for more information about this data format.

To activate EREP follow the procedure listed below. The job control language (JCL) statements referred to in the text are listed in "EREP Job Control Language" on page 49.

1. Startup the system with the bypass label processing (BLP) function.
2. Prepare and input the JCL.
3. Mount the TPF real-time tape (//INTAPE in the JCL) on the drive specified by MVS.
4. Mount the output history tape (//HISTORY in the JCL) on the tape drive specified by MVS.
5. Mount the work tape (//WORKOUT in the JCL) on the drive specified by MVS.

Processing Device Error Statistics

This program itemizes the hardware error messages sent to the CRAS console. It provides varying levels of error message summaries by device type, device address, and the time period under operator control. The error message listing and summaries are used to locate system hardware problems. The EREP generated work tapes are the input to this program. The EREP output (the work tape) is sorted beforehand using the MVS sort/merge utility program. This program is often referred to as the device error statistics postprocessor. "Device Error Statistics Postprocessor JCL" on page 49, lists the JCL statements used to start the program.

Follow this procedure to activate the postprocessor:
1. Prepare and input the JCL
2. Mount the work tape on the drive specified by MVS.

Recording 37x5 Native Subchannel Errors

A subchannel is part of the channel data path; a piece of hardware. This program records the pertinent data associated with 37x5 native subchannel errors. This data aids in locating the source of such errors. This program also increments the error counter in the 3705 keypoint record — CEPR. Error data is written to the real-time log tape, reformatted by program AMX1, and ultimately processed by EREP.

Display 37x5 Native Subchannel Error Counts

This program processes ZNERR commands. ZNERR is issued to display and/or reset (to zero) the 37x5 native subchannel error counters. There is one counter per control unit. In addition, this program checks the error totals maintained in the 3705 keypoint record. SNA support must be generated in the system to activate this program.

Appendix A. Job Control Language

This appendix contains examples of load decks to run the environmental recording, editing and printing (EREP) postprocessor and the device error statistics postprocessor. The format of the load decks vary depending on the type of storage medium used.

EREP Job Control Language

```
//JOBLIB   DD      DSN=xxxx,DISP=SHR
//AMS      EXEC    PGM=AMX1
//INTAPE   DD      VOL=SER=xxxx,LABEL=(2,BLP),UNIT=3480,
                   DISP=(OLD,KEEP),DCB=BLKSIZE=32760,DSN=xxxx
//HISTORY  DD      LABEL=(,NL),UNIT=3480,VOL=SER=xxxx,
                   DISP=(NEW,KEEP),DSN=xxxx,DCB=(BLKSIZE=yyyy,LRECL=zzzz)
//WORKOUT  DD      LABEL=(,NL),UNIT=3480,VOL=SER=xxxx,
                   DSN=xxxx,DISP=(NEW,KEEP)
//SYSUDUMP DD      SYSOUT=A
```

Where:

xxxx
: Any combination of characters or numbers in accordance with standard MVS conventions. Tape serial numbers and the data set on which the program AMX1 (EREP) must also be supplied.

yyyy
: 1832 bytes and *zzzz* = 1828 when creating a history file for EREP0 processing.

yyyy
: 1948 bytes and *zzzz* = 1944 when creating a history file for EREP1 processing.

Device Error Statistics Postprocessor JCL

```
//HLST1           EXEC PGM=SORT,PARM='MSG=AP'
//SORTIN    DD     LABEL=(,NL),UNIT=TAPE,VOL=SER=xxx,DISP=OLD,DSN=xxxx,
                  DCB=(RECFM=FB,LRECL=100,BLKSIZE=1000)
//SORTOUT   DD     DSN=&&HLSTINP,UNIT=DISK,SPACE=(SYL,(50,20),
                  DISP=(NEW,PASS),DCB=(RECFM=FB,LRECL=100,BLKSIZE=3600)
//SORTLIB   DD     DSN=SYS1.SORTLIB,DISP=SHR
//SYSOUT    DD     SYSOUT=A
//SORTWK01  DD     UNIT=DISK,SPACE=(3600,200,,CONTIG)
//SORTWK02  DD     UNIT=DISK,SPACE=(3600,200,,CONTIG)
//SORTWK03  DD     UNIT=DISK,SPACE=(3600,200,,CONTIG)
//SYSIN     DD     *
    SORT FIELDS=(1,6,BI,A),SIZE=E2000
/*
//HLST2     EXEC   PGM=HLST
//INPUT     DD     DSN=&&HLSTINP,DISP=(OLD,DELETE)
//SYSPRINT  DD     SYSOUT=A,DCB=BLKSIZE=121
//STEPLIB   DD     DSN=xxxx,DISP=SHR
```

Where:

xxxx
: Any combination of characters or numbers in accordance with standard MVS conventions. Tape serial numbers and the data set on which the program HLST resides must also be supplied.

Appendix B. Virtual Storage Layout

Figure 2 shows the layout of both ECB and system virtual memory.

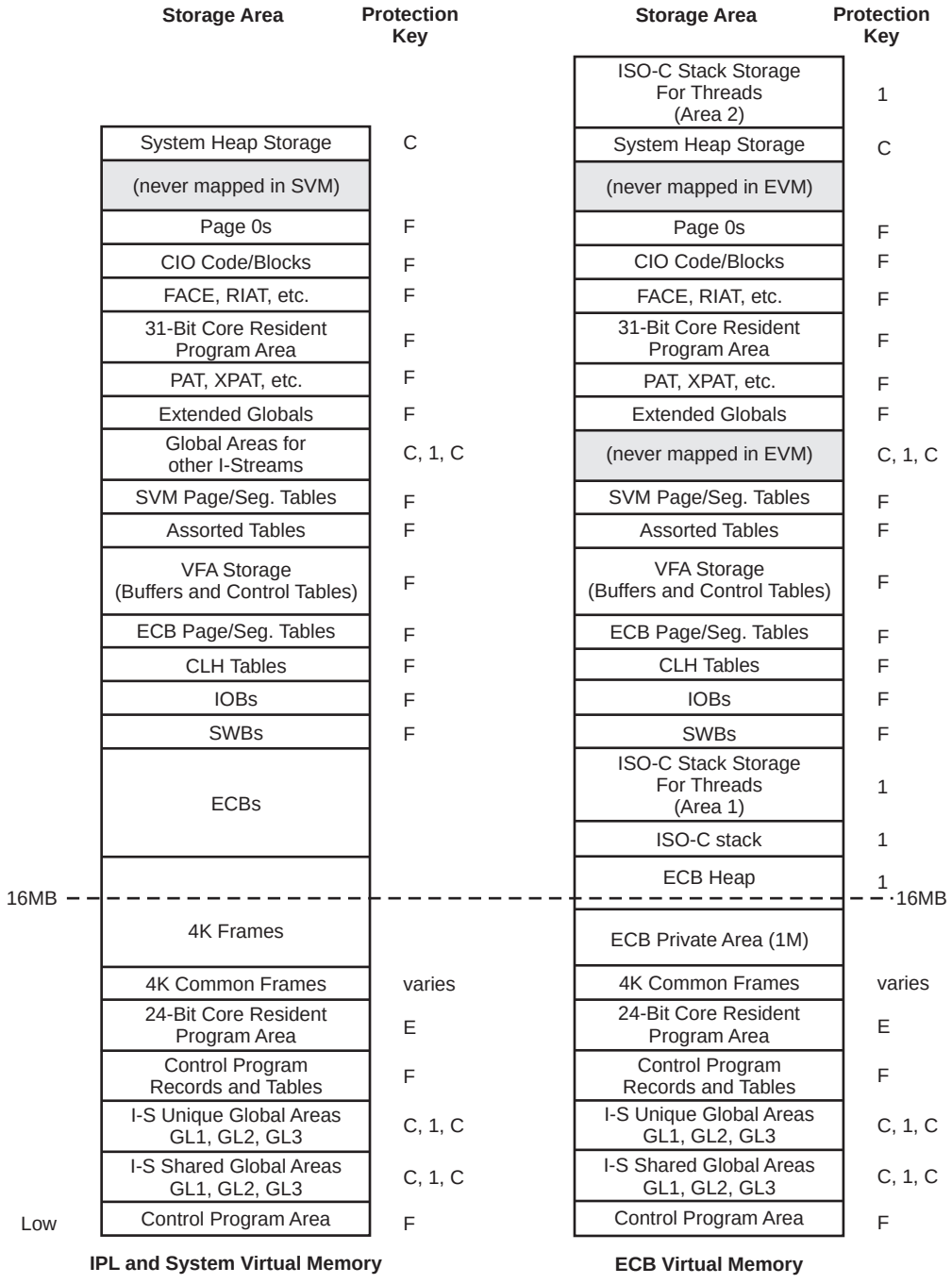

Figure 2. Virtual Storage Layout

Note: TPF's virtual address spaces makes it impossible to use a full 2 gigabytes of real memory.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

S

T

U

V

W

Z

File Number: S370/30XX-36
Program Number: 5748-T14

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH31-0159-07