Transaction Processing Facility

# Multi-Processor Interconnect Facility Reference

*Version 4 Release 1*

SH31-0155-01

Transaction Processing Facility

# Multi-Processor Interconnect Facility Reference

*Version 4 Release 1*

SH31-0155-01

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

**Second Edition (June 2000)**

This is a major revision of, and obsoletes, SH31-0155-00 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
Enterprise System/9000
ESCON
ES/3090
ES/9000
IBM
RETAIN
S/370
400

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

# About This Book

This book describes the Multi-Processor Interconnect Facility (MPIF), provided to meet the need for increased processing power. In addition, this book provides information concerning MPIF that you must understand when you are designing system or utility programs that interface with MPIF.

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 have a basic understanding of MPIF before reading this book. For an overview of MPIF, see *TPF Concepts and Structures*.

## Who Should Read This Book

This book is intended for system programmers doing system level coding for system services and support or utility programs that require MPIF services.

## How This Book is Organized

This book is organized by chapters and also contains several appendixes. The appendixes provide examples of the commands required to initialize a MPIF loosely coupled complex, and discuss 3088 address pairing, and the ESCON Architecture.

## 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:<br>A *database* is a collection of data.<br>Used to represent variable information. For example:<br>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:<br>Enter **ZNALS HELP** to obtain help information for the ZNALS command.<br>Used to represent variable information in C language. For example:<br>**level** |

| Conventions | Examples of Usage |
|---|---|
| monospaced | Used for messages and information that displays on a screen. For example:<br>`PROCESSING COMPLETED`<br>Used for C language functions. For example:<br>`maskc`<br>Used for examples. For example:<br>`maskc(MASKC_ENABLE, MASKC_IO);` |
| ***bold italic*** | Used for emphasis. For example:<br>You ***must*** type this command exactly as shown. |
| **Bold underscore** | Used to indicate the default in a list of options. For example:<br>**Keyword=OPTION1** \| **DEFAULT** |
| Vertical bar \| | Used to separate options in a list. (Also referred to as the OR symbol.) For example:<br>**Keyword=Option1** \| **Option2**<br>**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:<br>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.<br>`\|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...`<br>`LOADER   IMAGE CLEAR`<br>**Notes:**<br>1. The word LOADER must begin in column 1.<br>2. The word IMAGE must begin in column 10.<br>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 Concepts and Structures*, GH31-0139
- *TPF Library Guide*, GH31-0146
- *TPF Operations*, SH31-0162
- *TPF Transmission Control Protocol/Internet Protocol*, SH31-0120

## IBM Enterprise Systems/9000 (ES/9000) Books

- *ESA/390 ESCON Channel-to-Channel Adapter*, SA22-7203
- *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
- *ES/9000, ES/3090 Input/Output Configuration Program User's Guide and ESCON Channel-to-Channel Reference*, GC38-0097.

## 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

# Introduction to the MPIF

The High Performance Option feature allows a TPF loosely coupled complex, with up to eight copies of TPF, to communicate through a common database.

To meet the growing customer need for increased processing power, the Multi-Processor Interconnect Facility (MPIF) was introduced. MPIF supports multiple TPF complexes that communicate with each other. A TPF complex can consist of a single loosely coupled complex or a central processing complex (CPC) running TPF. Using MPIF, these complexes are be able to distribute processing across the TPF systems.

MPIF, pronounced *MIP-if* provides an interface that allows TPF systems to perform cooperative processing with other TPF systems. Using MPIF, one TPF system can request a system service that is received, processed, and returned by another TPF system. The requesting system need not be aware that the receiving system can be in a different processor, or that the receiving processor can have a different architecture. MPIF uses channel-to-channel (CTC) support to communicate in a single TPF loosely coupled complex, or between multiple TPF complexes.

Figure 1 shows a typical MPIF complex using the IBM* 3088 Multisystem Channel Communication Unit (MCCU).

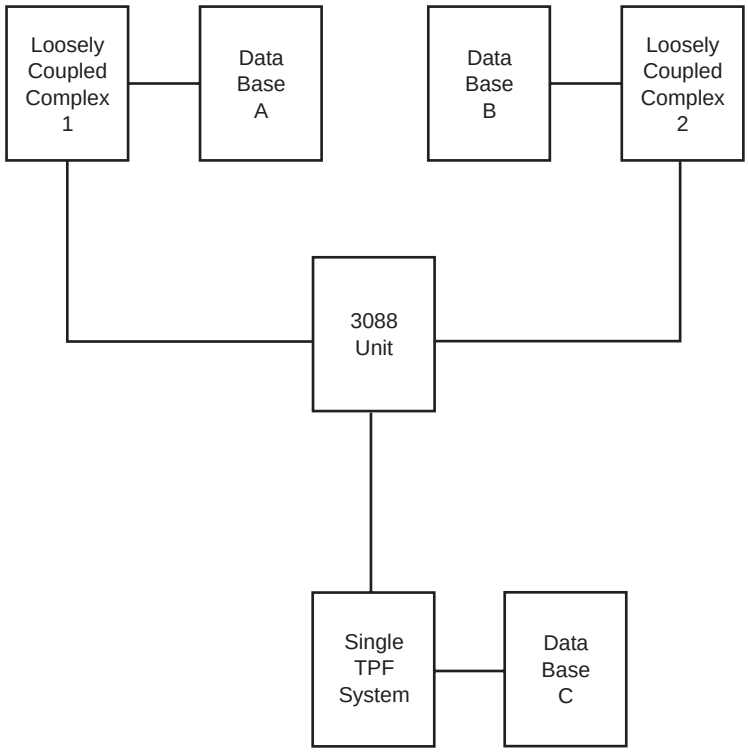


*Figure 1. A MPIF Complex Using a 3088*

Figure 2 on page 2 shows a typical MPIF complex, using the IBM Enterprise System Architecture* (ESCON*) Channel-to-Channel (CTC).

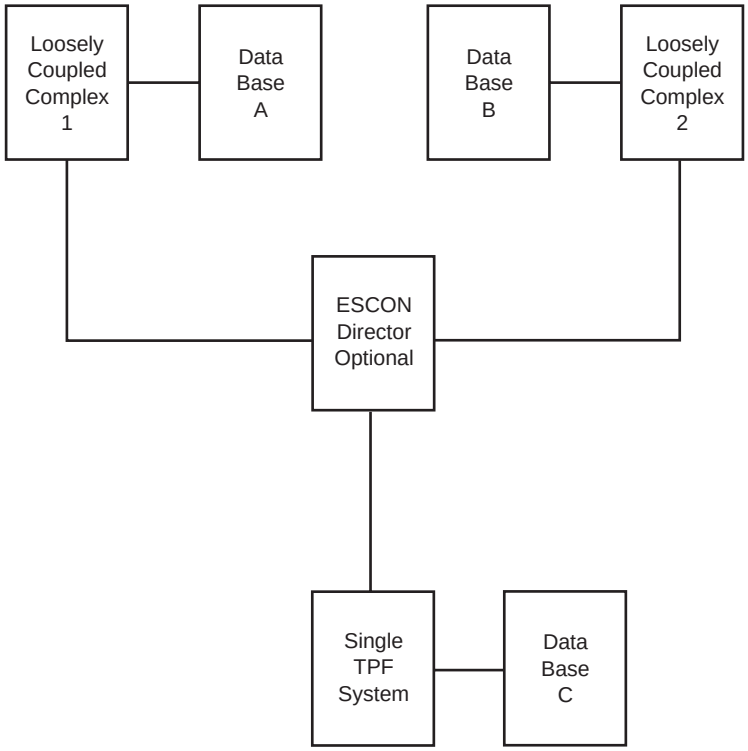


*Figure 2. A MPIF Complex Using an ESCON CTC*

If you are using ESCON CTC, see *ESA/390 ESCON Channel-to-Channel Adapter* for more information.

See the appropriate *Enterprise System/9000 Functional Characteristics and Configuration Guide* for more information about your particular ES/9000 processor model.

# Understanding and Using MPIF

This chapter describes:
- Basic MPIF terms
- A MPIF processing overview
- The services provided by MPIF
- User requirements in interfacing when using MPIF
- Processing MPIF errors
- An example of using MPIF services.

## Basic MPIF Terms

This section defines some basic MPIF terms. See *TPF Library Guide* for a complete list of definitions.

**Connection** A session established between two users.

**CNC** An ESCON channel attached to an ESCON capable device

**CTC** A CTC link, whether a 3088 or an ESCON CTC

**CPC** Central processing complex.

**Path** The route of communication between two processors. This route, or circuit, consists of a pair of physical units on each side of the circuit. These physical units are part of the multichannel communications device being used. The pair of units provides simultaneous read and write operations across the circuit. On each side of the path, one unit is designated for reading data while the other unit is designated for writing data.

**MPIF complex**
Multiple TPF complexes that communicate using a CTC communication link.

**MPIF user** System and support programs, that are run periodically, as part of TPF processing. Also referred to as a *user*.

**TPF complex** A TPF loosely coupled complex or a CPC running TPF.

**TPF loosely coupled complex**
Two or more copies of TPF contained in two or more processors, sharing a single data base. Also referred to as a *loosely coupled complex.*

**TPF system** A single copy of TPF. Also referred to as a *system.*

**User** See MPIF user.

To help clarify these definitions, consider the following:
- A user exists in a TPF system.
- Multiple TPF systems exist in a TPF loosely coupled complex.
- Multiple TPF complexes exist in a MPIF complex.
- Multiple TPF complexes communicate using a CTC communication link.
- A path must be defined over which two users can communicate.
- A connection must be established to allow two users to communicate.

## MPIF Processing Overview

MPIF uses channel-to-channel (CTC) communication links to establish connections between connected TPF complexes. Once these cross-system connections are made, logical connections can be established between MPIF users. Once the logical connection is established, transfer of data and commands across a path between the two MPIF users can be requested.

MPIF provides the typical TPF system control functions, such as error detection and buffer management. It is suitable for TPF system service programs that will be enhanced to schedule their services in one system (the requesting system) and perform their services in another locally-connected system (the server system).

MPIF allows TPF system services to be designed as resident or non-resident functions in that they can perform the service either in the requester's processor or in a locally attached processor. Any MPIF user connected to another user is be able to interact with no awareness of the location of the other user. MPIF resolves the residency question. Any control program service enhanced to be non-resident determines whether the service should be performed in the requester's processor or in a locally, channel attached processor. For the latter condition, the function uses MPIF to transfer the request to the appropriate processor. The TPF system service provides the same view to the requester regardless of whether the processing is performed in this processor or in a locally attached processor. A response to the requester has the same meaning, but the time to execute might be different.

## Services Provided by MPIF

MPIF allows users in two TPF systems to communicate. It does this through its path, connection, and data transfer services. MPIF also supports user exits to process different events.

- The *path services* allow operators to define paths to be used in the connection.
- The *connection services* allow users to establish a session for transferring data.
- The *data transfer services* allow users to send data over the defined connections.
- *User exit support* is provided to allow MPIF users to process asynchronous events relating to connections established or being established.

  See "User Exits" on page 20 for a complete description of the MPIF user exits.

## Path Services Support

MPIF defines logical paths and physical units. Logical paths are identified by a name, and are defined by operator commands. Physical units are defined with operator commands. When a logical path is started, two physical units are assigned to it. This allows logical paths to be assigned different functional characteristics.

### Path Class

Path classes are supported to allow users to select paths based on algorithms for bandwidth or functional criteria. Paths can be defined as belonging to a class, and users can request a connection by class when connecting to other users.

### Multiple Path Support

MPIF supports multiple paths between systems for bandwidth or user functional requirements. Multiple paths can be established when multiple CTC links exist, which allows for maximum bandwidth capability.

Users desiring multiple paths can issue a connect request to another user over each available path to that user. It is the user's responsibility to manage the multiple paths for performance and for sequencing of data, if relevant.

Users needing to separate their data requests for functional reasons can do so using their own algorithm, as is done for path classes.

### Protected Path

A protected path cannot have the last path of a path class to a system stopped by the **ZMPIF STOP PATH** command. In addition, a protected path causes messages to be placed on the receiving system's ready list instead of its input list, as in standard MPIF message processing. Changing the list used to receive messages from the input list to the ready list raises the priority associated with MPIF message processing.

### Load Balancing

MPIF load balancing supports multipathing by ensuring that each path, within a path class defined with load balancing, is assigned to a unique device (for example, a 3088 communication unit or an ESCON CTC where appropriate). The number of paths defined within such a class should be equivalent to, or a multiple of, the number of devices used by the class. The association of a path to a CTC link is made during cross-system startup.

### Alternate Path Support

For backup support, each system should have two paths established through CTC support. When a path fails, MPIF notifies affected users about the failure, and allows them to re-establish their connections using alternate paths, if they exist.

### Dedicated Path Support

MPIF users can separate their requests by data types or priority by using dedicated paths. A dedicated path supports one user-to-user connection. A path class can be defined that identifies dedicated paths on one or more CTC communication links. Users can then send their requests to the paths in the appropriate class.

MPIF also allows users to restrict CTC links to certain path classes. Users can then direct their requests to the links restricted to the appropriate class. In this way, a CTC link can be restricted to a specific function or purpose.

### Directory Update

MPIF maintains a global directory that contains an entry for each user in a MPIF complex. Each system in the MPIF complex maintains its own copy of the directory, which is updated by activity from other systems in the MPIF complex. As each MPIF user identifies itself, using the IDENTIFY parameter on the MPIFC macro, its name is broadcast to all attached systems as a directory update request.

## Connection Services

MPIF provides services for creating and terminating logical connections between MPIF users. The main services are called connection and disconnection. A connection must be established between two users before any communication between them can take place. The connection process involves coordinated actions of the two users and a series of requests, which are exchanged between the two users and the two associated MPIF systems. Both users must be activated before a connection can be established. A user becomes activated by identifying itself to MPIF.

When the users no longer want to communicate, the connection can be terminated. Either of the users can request a disconnection.

The following sections describe the MPIF connection protocols and services.

## Connection Protocols

MPIF supports two types of connection protocols:

- MPIF-to-MPIF connection
- User-to-user connection.

A *MPIF-to-MPIF connection* is used to exchange messages between the two MPIF systems, for the purpose of managing connections between users in the two systems. A MPIF system uses the path startup process to establish logical connections to other MPIF systems in other processors.

*A user-to-user connection* is used to exchange data between the two users. A connection is assigned to a single path. The connection is broken on path failure or failure, or on a disconnect request from either user.

## Using the Connection Services

Users initiate connection services by coding one of the following parameters on the MPIFC macro:

- **IDENTIFY**
- **CONNECT**
- **ACCEPT**
- **DISCONNECT**
- **FORGET**
- **QUERY**

The definition of these parameters follows:

**IDENTIFY**
: This function identifies the user to MPIF. The user is associated with MPIF resources that are required for other MPIF functions. Once a user is identified, MPIF considers that user as active.

**CONNECT**
: To establish a logical connection between two MPIF users, the two users must agree to the connection. One of the users indicates the desire to connect to another specifically named MPIF user by using the CONNECT parameter.

**ACCEPT**
: This function is used by a user to agree to a connection request from another user.

**DISCONNECT**
: This function is used by one of two connected users to remove a logical connection, or by a user to reject a CONNECT request. MPIF also uses this function for a given connection during failure of connection, user, or MPIF processing.

**FORGET**
: This function removes the named user from the MPIF complex. Additionally, FORGET does the same processing as DISCONNECT. FORGET is the opposite of IDENTIFY.

**QUERY**
: This function is used to determine if other specified users have identified themselves to MPIF. QUERY also provides the user the option to request notification when one or more specified users become identified to the resident MPIF system, or when one or more paths become available to a specific system.

## Data Transfer

When a user wants to send data to another user with which it has established a connection, the user invokes the MPIFC macro. In the macro, the user codes a request type of SEND, and creates a parameter list describing the data. When the request is transferred to the destination MPIF user, it executes a user exit routine to process the data.

## User Exits

MPIF supports the following user exits:

- Data received exit - Provides the user with data received from the other user across a specific connection.
- Connection request exit - Informs a MPIF user that another user has requested a connection.
- Connection completion exit - Informs a MPIF user that a connection request has been completed.
- Directory update exit - Informs a MPIF user of a newly identified user.
- Path active exit - Informs a MPIF user of a newly activated path, pertinent to the user's requirements.
- Error exit - Informs the MPIF user of asynchronous errors that occur, such as loss of a connection.

See "User Exits" on page 20 for details on these user exits.

# Interfacing with MPIF

This section describes the requirements for a user of MPIF, how a user can request MPIF services, and what data can be passed in requests.

## MPIF User Profile

MPIF provides services to a restricted set of users. These users should be system programs or system support programs, not application programs. MPIF users have the following profile:

- MPIF users operate within a complex of interconnected processors. The users are transportable within the complex.
- In a tightly coupled multiprocessing configuration, MPIF can operate in any I-stream engine within a single CPC. The I/O code supported by MPIF operates on the second I-stream, unless there is only one.

## Issuing MPIF Requests

MPIF requests can be issued using macros or operator commands, which are summarized below.

### Macros

The MPIFC macro is provided to allow MPIF users to request the following services:

- **IDENTIFY** - To establish the existence of a named MPIF user.
- **CONNECT** - To request a logical connection to another named MPIF user.
- **ACCEPT** - To accept a connection request from another MPIF user.
- **DISCONNECT** - To break an existing connection or refuse to establish a connection.
- **FORGET** - To eliminate all knowledge of a MPIF user.
- **QUERY** - To obtain miscellaneous parameters.
- **SEND** - To initiate transfer of data to another MPIF user.

This macro can be executed from an ECB driven program (E-type program) or a control program module (C-type program). C-type programs executing this macro must be aware of what address space they are running in.

### Operator Commands

MPIF provides commands for the operator to define and display status for resources such as paths and time-out intervals. All MPIF operator commands are prefixed with the action code ZMPIF and follow TPF conventions for commands. Some differences are:

- Commands and parameters can be abbreviated.
- A command can be used to change individual parameters; thus corrections do not require re-specification of all parameters when repeating a command.

Commands are provided to:

- Define path
- Start path
- Stop path
- Define device
- Start device
- Display device information
- Modify MPIF parameters
- Delete MPIF path/device
- Display MPIF activity/status
- Trace MPIF path/user.

See *TPF Operations* for detailed descriptions of these commands.

## Data Requirements for MPIF Requests

This section describes the data that can be passed in the various MPIF requests.

### MPIF Naming Conventions

A name can consist of one to 8 EBCDIC characters, left-justified in any 8-character field it might be contained in. A null always consists of eight blank characters. Exceptions are stated in the specific name description.

There are four types of names:

- System names
- User names
- Path names
- Generic names.

#### System Names

MPIF knows the logical name of each system in the MPIF complex by the SET command, a DEFINE PATH command, or Path startup. This information is available to MPIF users on request (see the QUERY parameter on the MPIFC macro).

#### User Names

All MPIF users issuing an IDENTIFY request type (see the MPIFC macro) are made known to each MPIF system. Each MPIF system maintains a directory of all the identified users in the MPIF complex (those users that are currently accessible). This distributed directory is updated when MPIF users identify themselves, execute a MPIFC forget request, or when the connection to a system is lost.

One MPIF user can request a connection to another user specifying user names in either of two ways:

- By specifying both the user name and the logical system name in which the user must be executing. This form is useful to select a specific copy of a distributed function.
- By specifying the user name and an asterisk (*) for the system name. If this form is used, the desired user must be located in a connected system, not the resident system. Because user names are guaranteed unique only within a single system, this user name must be unique within a MPIF complex.

### Path Names

All MPIF paths defined with the DEFINE PATH operator command are given a name for identification purposes. A path's name is provided when an action is to be performed on a path such as re-defining parameters for the path (DEFINE PATH), starting a path (START PATH), or stopping a path (STOP PATH).

### Generic Names

Generic names are used in MPIF to provide the capability to group paths by CTC addresses for START and STOP path commands. For example, all active paths through a specific 3088 MCCU or an ESCON CTC are grouped under the generic name. Generic names are in the format $n, where:

```
$ - is used to indicate a generic name
n - can be one to seven characters
```

## Tokens

The following 32-bit tokens are used in various MPIF requests. They are used for efficient interfacing between MPIF and MPIF users without, in most cases, defining what the value actually is.

*Table 1. MPIF Token Definitions*

| Token | Created by | | Description |
|---|---|---|---|
| | User | MPIF | |
| UITOK | x | | Given to MPIF via an IDENTIFY.<br><br>Kept by MPIF and given to user exits. This optional token is defined by the user. It might serve as a pointer to a primary control record or table. |
| IDTOK | | x | Given to user on return from an IDENTIFY request.<br><br>Kept by user and given to MPIF via MPIF connection functions. Associates the user with MPIF resources used for the user and is required for other MPIF functions. |
| UCTOK | x | | Given to MPIF via a CONNECT or ACCEPT request.<br><br>Kept by MPIF and given to user exits. This optional token can serve as a pointer to a secondary control record that represents a subfunction or connection. UCTOK might be converted to the assigned CONTOK which can vary when the connection is re-established due to failures or normal procedures. |

*Table 1. MPIF Token Definitions (continued)*

| Token | Created by | | Description |
|---|---|---|---|
| | User | MPIF | |
| CONTOK | | x | Given to user via a connect request or connect completion exit.<br><br>Kept by user and given to MPIF via the MPIFC macro. This token associates a request with a logical connection represented by a connection control block. |

# Initializing MPIF

## Defining the MPIF Environment

The definition of the MPIF environment is done using operator commands with the action code ZMPIF. The information provided with these commands become effective when the next IPL is done.

Follow these steps:

1. Define the complex parameters with the **ZMPIF SET COMPLEX** commands.
2. Define the system parameters with the **ZMPIF SET SYSTEM** command.
3. Define the path classes with the **ZMPIF SET CLASS** command.
4. Define the MPIF device(s) with the **ZMPIF DEFINE DEVICE** command.
5. Define the MPIF path(s) with the **ZMPIF DEFINE PATH** command.

Display this newly defined MPIF information using the following commands with no other operands:

- **ZMPIF SET COMPLEX**
- **ZMPIF SET SYSTEM**
- **ZMPIF SET CLASS**
- **ZMPIF DEFINE DEVICE**
- **ZMPIF DEFINE PATH**

## MPIF Initialization

Initialization of MPIF is done during system restart processing to make MPIF operational as early as possible during that process. Once MPIF is operational it schedules path startup processing which occurs asynchronous to the remainder of the system restart process.

MPIF users might at times need to wait for connections to critical resources such as users in other systems not yet identified to this system. Such users might want to use the event facility of TPF (for example, the EVNTC, EVNWC, and POSTC macros) and the MPIF user exits (for example, the path activation exit or the directory update exit) to implement a wait process for critical resources.

## Other Considerations

MPIF users should quiesce send activity before executing a **DISCONNECT** for a connection.

## MPIF Error Processing

MPIF failures, generally speaking, can occur in any one of several levels:

- MPIF failures
- Data transfer failures
- MPIF user failures
- Path failures
- Connection failures.

## MPIF Failures

Failures in MPIF C-type code are considered catastrophic and must be followed with a re-IPL of the system. Failures in MPIF E-type code is treated individually and appropriately for the condition and the function being performed. Generally, E-type failures does not cause a re-IPL of the system.

## Data Transfer Failures

During data transfers via MPIF from one user to another, many key MPIF resources are involved.

- MPIF buffers, staging areas, connections, and path resources are allocated.
- Both the **READ** and **WRITE** sides of the path must be manipulated.
- Failures might affect only a connection, or might affect the path and all other connections using the path.

To insure that these key resources are freed in case of failure, MPIF enforces disconnections for all users of the path. The processing includes the invoking of user error exits to process the queued items.

## MPIF User Failures

Each MPIF user has allocated to it MPIF resources primarily in the form of dedicated control blocks and dynamically allocated MPIF resources. These resources are shared by the users during MPIF processing of their requests (for example, during send or receive).

For example,

- A connection definition block is allocated and maintained at each end of a logical connection by MPIF for the associated users.
- A request block is allocated for each pending SEND request being processed by MPIF.

Whenever a user issues a **DISCONNECT** or **FORGET** request all MPIF resources allocated to the user are reclaimed for subsequent use by MPIF for other potential users. When a user fails, and does not affect the functioning of MPIF (that is, a ECB driven system utility function), allocated resources can become unavailable for re-use until the system is re-IPLed. Such users (for example, copies of some system utility connected to another common user or point) must provide for recovery of such allocated resources by scheduling a recovery process for failed users (for example, a time initiated monitor of such users). Neither MPIF nor TPF is able to detect a failure for this type of MPIF user.

## Path Failures

A path is a critical shared MPIF resource with broad implications to both MPIF and its users.

- It involves system and MPIF resources which are allocated on multiple systems.

- It supports multiple user connections, and thus also involves the corresponding connected users which reside on the two connected systems.
- It supports MPIF-to-MPIF communication and is required for the maintenance of such key MPIF resources as the global replicated directory.

The status of a path depends on the proper operation of the using processes (for example, send and receive), the correct functioning of the channel programs, and the correct and continued operation of the channel and hardware devices. A failure in any of these might result in path termination. After termination actions have been completed, a path restart is scheduled to ensure availability of the MPIF transport mechanism.

In addition, the path can be stopped (gracefully or not) with a command. If the STOP PATH command is issued with the quiesce option, path termination is started, but is not complete until all using connections have been given time to quiesce their activity and disconnect. Refer to the STOP PATH command in *TPF Operations* for a description of this interval.

If the STOP PATH command is issued with the purge option, path termination completes regardless of the status of the connections; some user data might be lost in the process. The user is informed of the abnormal disconnect. In this case, the path is not automatically restarted.

The following are a set of conditions that might cause path startup problems:

- Paths in different systems are not defined with the same class.
- Path classes do not have the same meaning in each system in the MPIF complex.
- Insufficient control block to support an active path. Any of the following might be deficient: GFND, CDB.
- Output queue depth (as specified in ZMPIF SET SYSTEM) might be zero.
- System names are not consistent within a MPIF complex.
- Path classes defined as load balancing. If so, only one path of that class is active per device name.

## Connection Failures

The connection process is a complex, multistage, asynchronous process involving the coordinated actions of the two connecting users and the two MPIF systems. The process of terminating the connection (disconnection) at the request of one of the users involves similar asynchronous actions by these four parties.

Connection termination can be initiated by a user via a DISCONNECT request or by MPIF. A connection is a critical MPIF resource upon which numerous MPIF processes are dependent. A connection is, itself, also highly dependent upon other critical MPIF resources in order to be fully functional; for example, a failure among any of the aforementioned parties or the path to which it is allocated causes the connection to terminate. Also, the failure of any process using the connection (send or receive), can require the termination of the connection in order to ensure proper recovery from the error.

## Example of Using MPIF Services

Figure 3 on page 14 and Figure 4 on page 15 describe the MPIF macro flows and the use of tokens and user exits. Tokens are shown as **TK***nn*. The user exits are indicated as:

CCX - Connection completion exit
CRX - Connection request exit
DRX - Data receive exit
ERX - Error exit.

## IDENTIFY/CONNECT Flow

Figure 3 on page 14 shows an example of the flow for **IDENTIFY/CONNECT** and indicates three possible responses to **CONNECT**:

- `ACCEPT`
- `DISCONNECT`
- No answer, in which case there is a timeout.

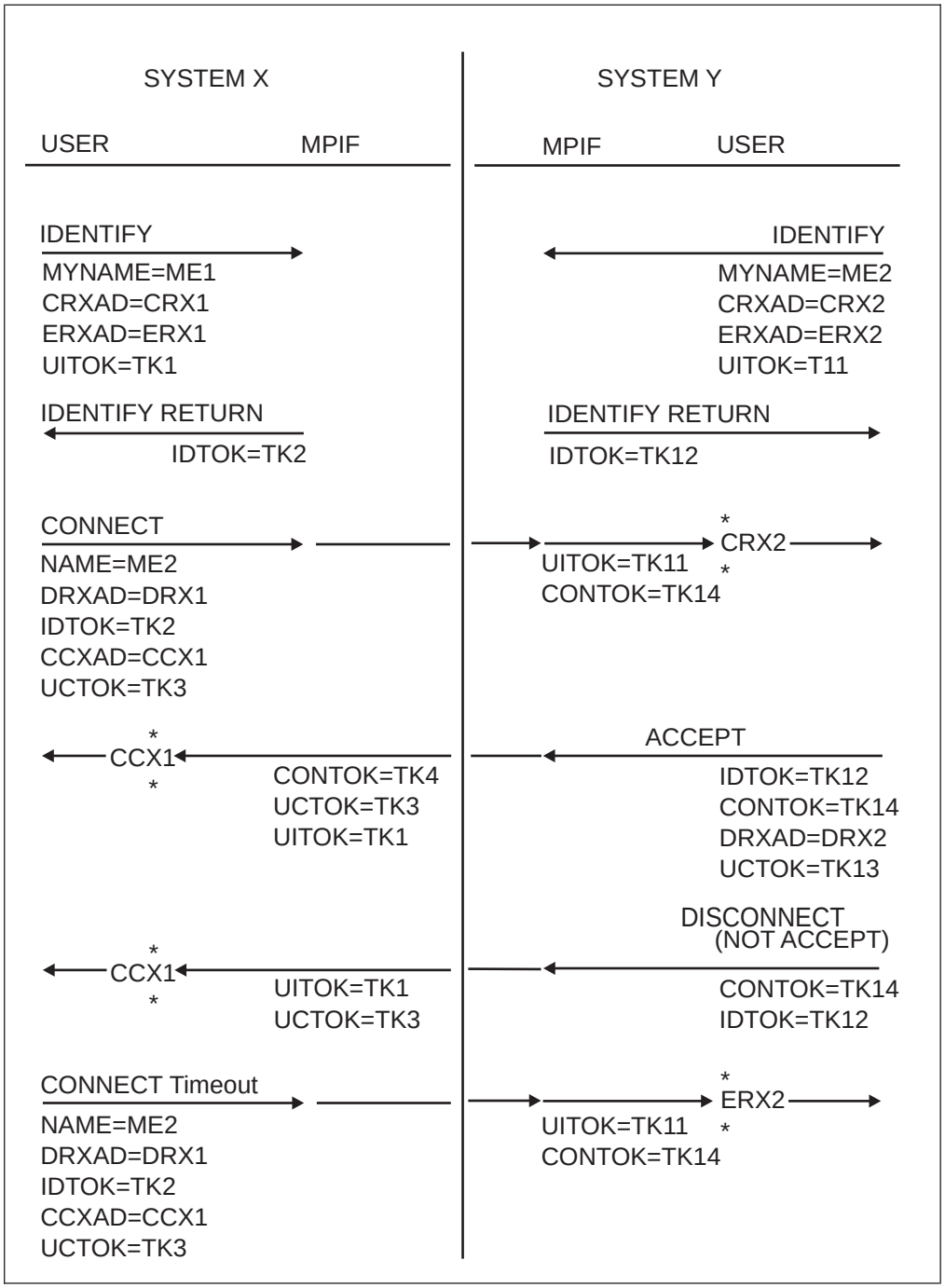


*Figure 3. MPIF IDENTIFY and CONNECT Flows*

Also, a connect race condition can occur if both X and Y issue CONNECT at the same time; only one connection is needed. MPIF supports *n* connections between users. If users desire only one, they must define a startup protocol to determine which user can issue the connection.

All events associated with establishing a connection occur across the path allocated to the originating CONNECT request. This simplifies processing for path failures. If a path fails during this processing, the connect request is failed. This avoids having to recover from all the conditions that a path failure can produce, including receiving obsolete requests about a connection.

## SEND/DISCONNECT Flow

Figure 4 shows an example of the flow for **SEND**/**DISCONNECT**.

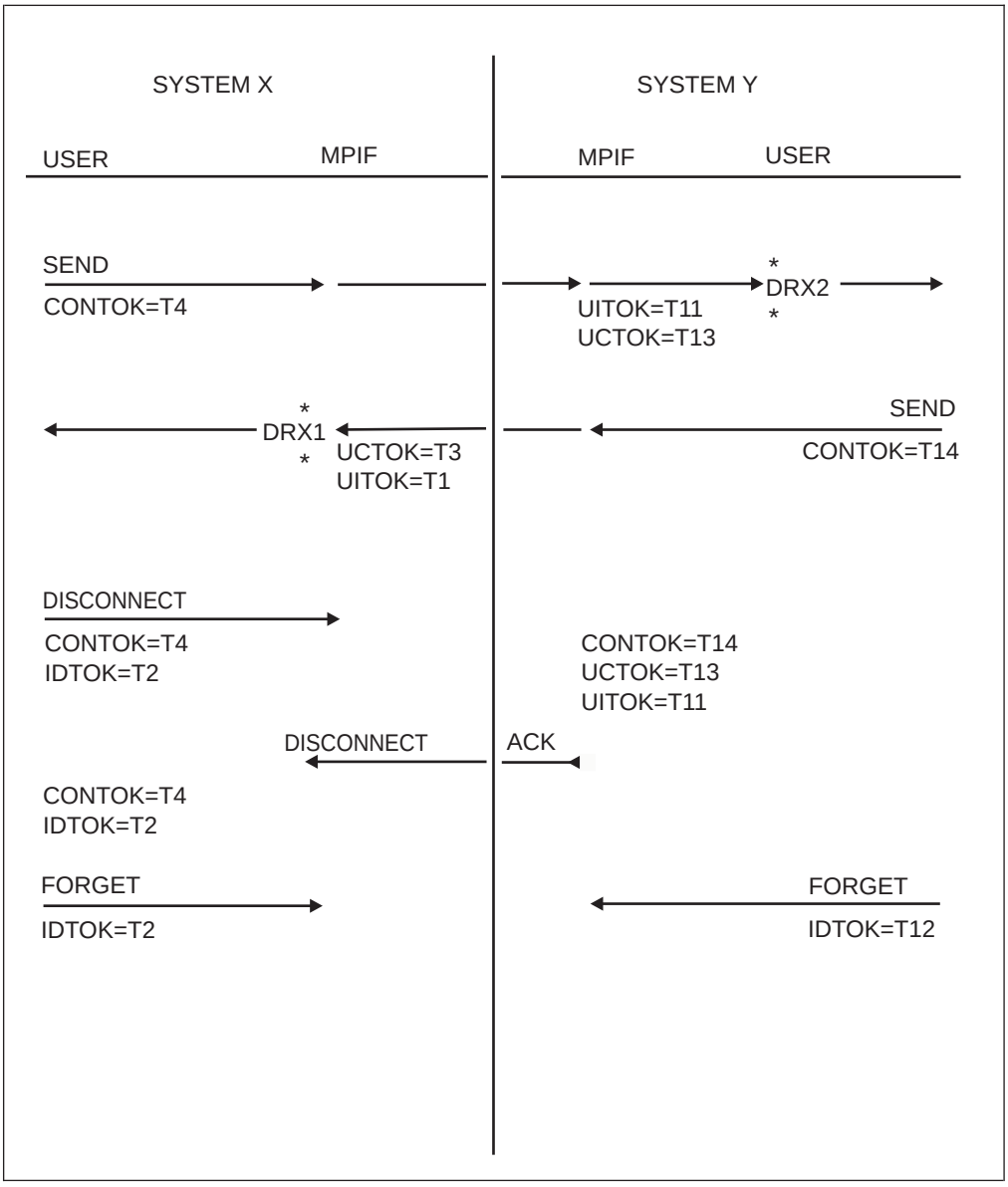


*Figure 4. MPIF SEND and DISCONNECT Flows*

In Figure 3 on page 14 and Figure 4 the fact that the user in X issued a SEND first is just for example. The user in Y can issue a SEND first, but this might result in X receiving a message before being told that the connection is complete.

# Programming Guide

This chapter describes some of the MPIF processing that you must understand in designing system or utility programs to interface with MPIF.

## Status Management

There is no explicit common status management facility in MPIF; however, the following should be noted:

- MPIF users identified via the IDENTIFY macro will be considered as active.
- Paths processed by MPIF path startup will be considered as active.
- The systems connected at each end of active paths will be considered as active.
- MPIF will police all paths for lost interrupt conditions and for inactivity on a time basis.
- Connections will be timed-out if not completed within a predefined period of time.
- The echo check facility will verify the availability of a path found to have had no activity for a predefined period of time.

## Echo Check

The MPIF echo check function exists to ensure that MPIF paths which become inoperative do not remain undetected. Echo Check is invoked by MPIF at a user-specified time interval. At each interval an echo check message is sent over each started path with no activity since the last expired time interval. If the message is not read by the other side before the next time interval, path failure is assumed.

Potential causes of this are:

- Failure of the other system's software.
- Failure of the other system's hardware.
- Failure of a channel or CTC support in such a manner that no I/O errors are generated.
- STOP of the other system's instruction stream.
- Unresponsiveness of the other system. This could be due to an overloaded system, or disabling activity such as depleted core conditions, or system error processing.

## Pacing Control

In order to avoid ***performance*** impacts to TPF, core blocks must be available for MPIF to dispatch requests for processing by defined user exits. If there are no blocks, no reads will be scheduled by MPIF. This is purposefully unfriendly: user functions should be designed to avoid such conditions when appropriate. The pacing facilities provided by MPIF can help.

When multiple copies of the same code (for example, multiple TPF routers) are going to communicate (HOMOGENEOUS connections), the code can be easily designed to use a protocol that avoids flooding a system with more SENDs than it has core blocks to support. For example, a protocol that is *SEND, then wait for RECEIVE* does not overrun either side's core block lists.

When a system service is providing a general facility and other system services can connect to it (HETEROGENEOUS connections), things get more difficult. While the

system service providing the general facility can define a protocol that must be used by its users, it cannot prevent a user from violating the protocol (it can only punish users for violations). A user violating the protocol can impact other connections to the general purpose system service.

## Prevent Buffer Overruns

The MPIF pacing facility provides a simple mechanism to help prevent buffer overrun conditions and to enforce this prevention. It allows each side of a connection to specify the maximum number of its buffers that the other side can consume before it is forced to stop SENDing on a specific connection. MPIF maintains a *pacing credit count* and when credit is exhausted, SENDs from that side of the connection are stopped. The receiver is able to increase the sender's *pacing credit count*, by providing the sender with credit with messages sent to the sender. Thus the receiver is able to control the number of the receiver's core blocks consumed by the sender. When options are defaulted, each time the receiver SENDs to the other side, the other side's *pacing credit count* is incremented by one (credited with 1). This default supports protocols that are rigidly one-in, one-out as well as protocols that are many-in, many-out with an equal number out.

## SEND Logic

Additional user logic is required where MPIF pacing is desired but the number of SENDs by each side is not expected to be the same. SEND allows one side to provide the other side with credit for multiple sends. The default provides the other side with credit for a single send. Figure 5 shows the SEND logic exploited by MPIF.

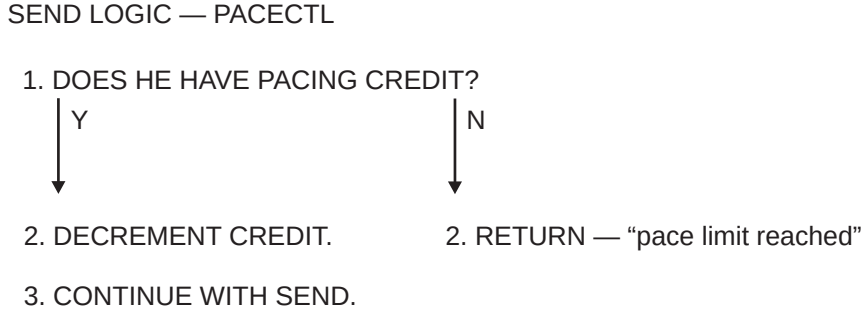


*Figure 5. SEND LOGIC - PACING*

## RECEIVE Logic

Figure 6 shows the RECEIVE logic exploited by MPIF.

```
RECEIVE LOGIC - CREDIT OF 1

  1. INCREMENT CREDIT.


  2. PASS RECEIVED BUFFER
     TO USER.
```

*Figure 6. RECEIVE LOGIC - PACING*

## Deadlock Situation

It is possible to create a *deadlock* situation using pacing where neither side can send because both have reached their pacing send count. This deadlock can also be avoided by appropriate design.

It might be desirable to segregate pacing for data types or sizes. This could be accomplished by establishing two connections between the two users - each connection would be used for different data types or sizes. Two connections would allow independent control of pacing.

The MPIF pacing facilities can be ignored by not selecting the option via the CONNECT or ACCEPT parameter lists.

The MPIF pacing facilities provide the ability to do what is generally referred to as *windowing*. In most cases, this can be accomplished by piggybacking credit on data or acknowledgement SENDs, thus avoiding extra communications.

## Sequence Number Control

MPIF operates in and supports a highly parallel environment. Users of MPIF can issue functions such as SEND almost simultaneously in uniprocessor machines. MPIF is designed to deliver messages in the same sequence that they were provided to MPIF, but the highly parallel environment can cause the illusion that messages are delivered out of sequence. If a single ECB issues two SEND requests, the first data block will be scheduled for the user at the other end of the specified connection before the second data block. That first data block might not get processed first. For example, an E-type data received exit might invoke some TPF macro which has enforced an implied wait. The second data block, in the meantime, could be passed to the same Exit under a different ECB and the execution of the same TPF macro might not enforce an implied wait this time; therefore, processing of the second data block could progress ahead of that for the first data block--it could very well get processed out-of-sequence.

Another example of a sequence that can occur is the invocation of a connect complete or error or data receive exit associated with a connection after the user has issued DISCONNECT. This can occur because the user had not quiesced SEND activity or because the exit was scheduled before the DISCONNECT was processed.

The MPIF SEND function will assign a *connection sequence number*. This number is passed to the issuer of SEND and to the user data receive exit. This *connection sequence number* is a 32-bit unsigned number associated with a specific connection that begins with 1 and wraps to zero. When multiple SENDs are issued for the same connection (that is, the user at the other end of the connection), the sequence numbers will be unique and increase by one. Any discrepancy might be indicative of an out-of-sequence condition or a failure relating to the connection.

## Blocking of Data Request Elements

MPIF performs *blocking* of data elements received from separate system services in order to maximize I/O efficiency. Prior to invoking the I/O routine (during MPIFC execution or during a write I/O interrupt), MPIF assembles numerous data elements into a single staging buffer. These data elements are removed from a PDT entry's output queue. The I/O routine initiates execution of a channel program to write the contents of the staging buffer to the other processor.

Upon receipt of a read interrupt, the I/O routine places the staging buffer on the ready list or the input list for post interrupt processing. The *post interrupt routine* recognizes that blocking has been performed. The individual data elements are de-blocked and the appropriate user exits will be scheduled.

## List Support for Segmented Data

MPIF allows a user to send multiple data blocks with the list variation of the **SEND** request parameter list. The maximum number of data bytes that can be transferred with a list, is limited by the maximum write buffer size specified for the connection, and for the path class used for the connection. The list format is available to C-type programs only. The list must contain the system virtual address of blocks not associated with an ECB. The blocks are not copied but are used as passed. See the DCTMUP data macro for details on how to determine the number of bytes for the list control information.

## Priorities

MPIF supports three types of priorities that can be specified by the user in a SEND Request parameter list:

- Normal priority (in the sending system)
- Control message priority (in the sending system)
- Dispatching priority (in the receiving system).

Requests specifying normal priority are serviced in the order they are issued, using a first-in, first-out method.

A control message request that is waiting to be transferred on the same path and destination as a normal request will be serviced before the normal request. Multiple requests indicating control message priority are treated equally, using the first-in, first-out method.

A request with dispatching priority can specify initial dispatching from the ready list or the input list, in the system to which the SEND request is transferred.

## User Exits

## Data Received Exit

### Purpose

To pass to the user a request in the form of a parameter list and, if relevant, associated core blocks containing data sent over a connection from another user.

### Suggested Processing

The user's data received exit might perform message sequencing and assembly if appropriate. This could include queueing of data blocks which are not received in the correct sequence. Following this first step, the exit might process the data block(s) if appropriate: otherwise, the exit might schedule another program for processing of the data.

### Interface

The data received exit address is specified with a CONNECT or ACCEPT request and can be different for each connection. MPIF schedules the exit when a request is received from the other side of the connection.

The exit routine can be C or E type code. For C-type code, the exit will be a post interrupt address (PIA). For E-type code, the exit will be a SWISC expansion.

Registers for E-type routines are as follows:

**R1** Contains the address of the data received exit parameter list. The parameter list is described in the DCTMUP data macro.

**R7** Since no save area is provided, R7 will contain zeros.

**ECB** The parameter list is contained in the ECB starting at EBX000.

Registers for C-type routines are as follows:

**R1** Contains the address of the data received exit parameter list. The parameter list is described in the DCTMUP data macro.

**R13** Points to the new stack area.

**R15** Contains the base address of the exit (that is, the PIA).

## Connection Request Exit

### Purpose

To inform a MPIF user that another user has requested a connection.

### Suggested Processing

- Validate that the requestor is authorized to connect.
- Verify that the state is appropriate to establish a connection.
- Save the connection token in the control structure (for example, tables), if one exists.
- Analyze and/or save any data received with the request.
- Respond with an ACCEPT or DISCONNECT.

### Interface

The connection request exit is specified via an IDENTIFY. The exit is scheduled when a MPIF user in another system requests a connection with a CONNECT request.

The exit routine can be C or E type code. For C-type code, the exit will be a post interrupt address (PIA). For E-type code, the exit will be an SWISC expansion.

Registers for E-type routines are as follows:

**R1** Contains the address of the connection request exit parameter list. The parameter list is described in the DCTMUP data macro.

**R7** Since no save area is provided, R7 will contain zeros.

**ECB** The parameter list is contained in the ECB starting at EBX000.

Registers for C-type routines are as follows:

**R1** Contains the address of the connect request exit parameter list. The parameter list is described in the DCTMUP data macro.

**R13** Points to the new stack area.

**R15** Contains the base address of the exit (that is, the PIA).

## Connection Completion Exit

### Purpose
To inform a MPIF user that a connection request has been completed.

### Suggested Processing
- Save the connection token.
- Analyze and/or save any data from the ACCEPT.
- Note that the connection is available for SENDing data.
- Issue the initial SEND request or cause this to be done, if appropriate.

### Interface
The connection completion exit is specified via a CONNECT request. The exit is scheduled when a connection request is completed (either the connection is complete or the connection request has failed).

The exit routine can be C or E type code. For C-type code, the exit will be a post interrupt address (PIA). For E-type code, the exit will be an SWISC expansion.

Registers for E-type routines are as follows:

**R1** Contains the address of the connection completion exit parameter list. The parameter list is described in the DCTMUP data macro.

**R7** Since no save area is provided, R7 will contain zeros.

**ECB** The parameter list is contained in the ECB starting at EBX000.

Registers for C-type routines are as follows:

**R1** Contains the address of the connection completion exit parameter list. The parameter list is described in the DCTMUP data macro.

**R13** Points to the new stack area.

**R15** Contains the base address of the exit (that is, the PIA).

## Directory Update Exit

### Purpose
To inform a MPIF user that a QUERYed user is now identified in another system. This notification takes place only once for each MPIFC QUERY request unless a wildcard character (*) is used to specify the system name.

This user exit is optional; however, it is important to understand that path startup processing can complete asynchronous to any other activity; thus, QUERY's for "active" MPIF users issued immediately after path startup processing will not necessarily detect the existence of active MPIF users.

Other procedures to achieve the same results might be employed by MPIF users when appropriate; for example, the QUERY request could be reexecuted on a time activated basis.

### Suggested Processing
- Issue a CONNECT request to establish a connection to the other user.

### Interface
The directory update notification exit is invoked whenever a directory replace or update is received with a named MPIF user which was previously queried by a user in this system.

The exit routine can be C or E type code. For C-type code, the exit will be a post interrupt address (PIA). For E-type code, the exit will be a SWISC expansion.

Registers for E-type routines are as follows:

**R1** Contains the address of the directory update exit parameter list. The parameter list is described in the DCTMUP data macro.

**R7** Since no save area is provided, R7 will contain zeros.

**ECB** The parameter list is contained in the ECB starting at EBX000.

Registers for C-type routines are as follows:

**R1** Contains the address of the directory update exit parameter list. The parameter list is described in the DCTMUP data macro.

**R13** Points to the new stack area.

**R15** Contains the base address of the exit (that is, the PIA).

## Path Active Exit

### Purpose

To inform a MPIF user that a new path that is relevant to the user's operation (that is, a class of paths the user is using for connections to a specific system) has been activated. This notification takes place only once for each MPIFC QUERY request unless a wildcard character (*) is used to specify the system name. This optional user exit will be pertinent to those users of multiple pathing.

### Suggested Processing

- Issue a CONNECT request to establish a connection to the other user using the newly activated path.

### Interface

The path active notification exit is invoked whenever a new path of the specified class to the specified system has been activated via path startup.

The exit routine can be C or E type code. For C-type code, the exit will be a post interrupt address (PIA). For E-type code, the exit will be an SWISC expansion.

Registers for E-type routines are as follows:

**R1** Contains the address of the path active exit parameter list. The parameter list is described in the DCTMUP data macro.

**R7** Since no save area is provided, R7 will contain zeros.

**ECB** The parameter list is contained in the ECB starting at EBX000.

Registers for C-type routines are as follows:

**R1** Contains the address of the path active exit parameter list. The parameter list is described in the DCTMUP data macro.

**R13** Points to the new stack area.

**R15** Contains the base address of the exit (that is, the PIA).

## Error Exit

### Purpose

To inform the user of asynchronous events, normally errors, related to MPIF and MPIF connections.

### Suggested Processing

- For connection related errors, issue status messages as appropriate.
- For “path stopping” warnings, quiesce the connection normally and issue a DISCONNECT request.
- For other warnings, cause message recovery logic to be invoked to ensure that no blocks of data have been lost; this logic might disconnect then reconnect or just use recovery messages.
- For disconnect other than user requested disconnect, determine if an alternate path is available and, if so, attempt to re-connect.
- Schedule path activation/directory notification as necessary.

### Interface

The Error Exit is specified via an IDENTIFY request. The exit is scheduled when a error is detected - most, but not all errors are related to a particular connection.

The exit routine can be C or E type code. For C-type code, the exit will be a post interrupt address (PIA). For E-type code, the exit will be an SWISC expansion.

Registers for E-type routines are as follows:

**R1** Contains the address of the error exit parameter list. The parameter list is described in the DCTMUP data macro.

**R7** Since no save area is provided, R7 will contain zeros.

**ECB** The parameter list is contained in the ECB starting at EBX000.

Registers for C-type routines are as follows:

**R1** Contains the address of the error exit parameter list. The parameter list is described in the DCTMUP data macro.

**R13** Points to the new stack area.

**R15** Contains the base address of the exit (that is, the PIA).

# MPIF User Parameter List Definitions (DCTMUP)

## Description

**RESTRICTIONS**

The DCTMUP macro, along with the MPIFC macro, is a restricted use macro with an interface which is not guaranteed across releases of TPF. Unauthorized use of this macro might expose the user to interface and/or processing errors.

This data definition macro describes parameter lists used to define:

1. Specific requests for the MPIFC macro (Request MPIF Service)
2. Input to user exits defined to MPIF.

3. A user provided area which is initialized by MPIF with the results of a QUERY request.

You invoke DCTMUP as follows:

```
DCTMUP  REG=Rxx, LTYPE=Parameter List
```

**REG=**
is the keyword used to specify a symbolic register name (Rxx) valid for assignment to the USING statement as the base register for DCTMUP DSECT.

**LTYPE=**
The MPIF service request parameter lists are:

| | |
|---|---|
| **IDENTIFY** | To identify named MPIF user. |
| **CONNECT** | To establish a connection. |
| **ACCEPT** | To accept connect request from another user. |
| **DISCONNECT** | To break the connection. |
| **FORGET** | To delete reference of user. |
| **QUERY** | To obtain miscellaneous information. |
| **SEND** | To send a request to another user. |

The MPIF user exit parameter lists are:

| | |
|---|---|
| **CRX** | Input to user's connect request exit. |
| **CCX** | Input to user's connect request completion exit. |
| **DRX** | Input to user's data received exit. |
| **DUX** | Input to user's directory update notification exit. |
| **ERX** | Input to user's error exit |
| **PAX** | Input to user's path active notification exit. |
| **QRA** | Response to query request. |
| **ALL** | A call for all of the above definitions. |

# MPIF Post-Processing Application

A MPIF post-processing application has been written to verify MPIF interconnect traffic from an online (real-time) TPF environment. This application depends on MPIF trace being set ON with real-time (tape) logging. The log tape is then used as input to this application in a TPF offline environment, using MVS as the offline post-processing system. Prerequisites to executing MPIF post-processing services would therefore include access to one or two trace tapes from a TPF real-time environment. The number of log tapes would depend on the type of report requested by the user.

Since a real-time TPF trace tape may contain many types of records, this application is designed to process MPIF Multisystem request block (MSRB) records only, using the TRCID field in the DCTTRC DSECT to identify trace record IDs containing X'00E3' as the field contents, where X'00E3' is the MPIF MSRB trace record ID.

This post-processing application is designed to be used as a debugging and/or troubleshooting tool for customers using MPIF. It generates “easy-to-read” reports for user verification of MPIF online activities. In some instances, users may install MPIF on a given CPU wanting to verify that MPIF functions are operative within this newly installed system. In this situation, users can input one trace tape from that CPU only. A report can then be generated containing “easy-to-read” displays of every MPIF activity that occurred for this specific CPU. In other instances, users may want to compare MPIF activities between two systems. This can be accomplished by inputting two log tapes. The MPIF post-processing application will then perform a CPU-to-CPU comparison of messages transferred between them, producing a summary report of traced messages.

Since this application generates two unique types of reports, a method has been devised that enables customers to specify which report is being requested. This is accomplished by generating post-processing commands that are applicable to this application only. The commands include:

**PRINT** Print the contents of one MPIF trace tape only; generating a report of each traced activity.

**COMPARE** Compare the contents of two MPIF trace tapes as a CPU-to-CPU analysis of message traffic and generate a summary report.

Furthermore, it is assumed that each system at a customer location is only interested in his (or her) MPIF message traffic. It is therefore important to extract logged information that meets the specific requirements of the requestor. This has been accomplished by examining all traced fields (defined in the DCTTRC DSECT) to identify those fields that are candidates for defining a search criteria. If a specific field could be used as a customer-defined search criteria, that field has been assigned a keyword to form a

```
KEYWORD=ARGUMENT
```

parameter that can be entered by users to specifically identify the MPIF traffic that should be reported. For example, the MPIF trace record contains the MPIF logical path name (referenced as TRCPATHN in the DCTTRC DSECT). If troubleshooting occurs for a specific logical path, the user can then request a report containing traffic on that path only:

```
PRINT PATH=FAIR3088
```

If a problem occurred on a given logical path on a specific day at a specific time interval, the user can further qualify the search criteria as follows:

```
PRINT PATH=FAIR3088,DATE=15DEC,TIME=02.15.00
```

This MPIF post-processing application will interpret the above report-generation search criteria as: print only those MPIF traced activities that occurred for logical path name= FAIR3088, on December 15 between 02:15 PM and 02:59 PM. The TIME= parameter functions as a START-TIME, with an implied STOP-TIME equated to “end of hour.” If the implied STOP-TIME exceeds the requestor’s search criteria a DUR= parameter can be specified as follows:

```
PRINT PATH=FAIR3088,DATE=15DEC,TIME=02.15.00,DUR=10
```

Using the above search criteria, the MPIF trace log tape report will contain a report of activity from 02:15 PM through 02:25 PM; treating the TIME= parameter as the START-TIME; adding DUR= to the TIME= (minutes field) to derive an explicit STOP-TIME. Stated differently, the TIME= parameter functions as a START-TIME with an implied (default) STOP-TIME equated to “end of hour” unless the implied STOP-TIME is overwritten by a DUR= parameter that specifies elapsed minutes: the elapsed minutes will be added to the START-TIME to form an explicit STOP-TIME. If the DUR= parameter forces the explicit STOP-TIME to exceed an hour, the STOP-TIME will be forced to “end of hour”.

Using either an implied or explicit STOP-TIME, this application is designed to search trace tape records, examining the time stamp in the 4K block header (TRCTTIME), extracting those records that fall within the allowable TIME= range. Since the TIME= parameter is dependent on a DATE= parameter for full qualification, this application is designed to validate parameters, generating an error message if TIME= is entered without DATE= (paired parameters). However, DATE= can be entered as an independent parameter.

## Keywords Specifications For Report Generation

A list of acceptable keywords for MPIF post-processing has been provided below.

The keywords are:

**BLK** Number of 4K blocks to be printed from the trace tape containing MPIF trace entries. Default = 1,000 4K blocks.

**MSG** Number of message entries to be printed per 4K block. Default = all messages within each 4K block.

**USER** A specific USER that was traced on the MPIF trace tape.

**ORG** A specific origin (or source) that was specified on MPIF message transmissions.

**DEST** A specific destination that was specified for MPIF message transmissions.

***DATE** A specific DATE for traced messages on the MPIF trace tape.

***TIME** A specific TIME for traced messages on the MPIF trace tape, using the TIME stamp in the 4K HDR.

If a time is provided (HH.MM.SS), this program will select all messages within the hour (HH), on-or-after the minute (MM) specified.

Example: 02.15.45 says “Give me any messages on or after 2:15 PM (through 2.59.59).”

**DUR** A MM (minutes) notation signifying elapsed time. If the elapsed time exceeds the current hour, an “end of hour” STOP-TIME will be assigned.

**PATH** A specific PATH that was used during MPIF message transmission.

***CPU1** The primary CPU or system ID where MPIF trace is active.

***CPU2** The connected CPU or system ID used in a COMPARE analysis between two CPUs.

**Note:** An asterisk (*) above denotes required keywords that must be entered for the COMPARE command.

Each of the above keywords (if specified) must be entered as spelled above; using uppercase **or** lowercase letters. The arguments must be entered as they appear on the MPIF log tape, using the DCTTRC DSECT for format lengths. If arguments are truncated, when entered as KEYWORD= arguments, they will be blank-padded from left to right. That is, a 4-character USER=aaaa, will be placed in an 8-character field with blanks in positions 5 through 8. Blank padding will only occur for alphanumeric fields that include USER, DEST, ORG, PATH, CPU1 AND CPU2.

Numeric-only arguments (BLK=, MSG=, and DUR= ) must be zero-filled, according to their size. Both the argument size and range of values have been specified below:

**BLK=** A four-position value ranging from 0001 through 9999 in numeric value. Default = 1,000

**MSG=** A two-position value ranging from 01 through 63 in numeric value. Default = ALL (messages in 4K BLK).

**DUR=** A two-position value ranging from 01 through 59 in numeric value. Default = “end of hour” STOP-TIME.

Any mixture of keyword parameters can be specified for PRINT and COMPARE commands. With the PRINT command, at least one parameter must be specified. With the COMPARE command, however, four keyword parameters are required; denoted by an asterisk (*) in the keyword list. For quick reference, however, these required COMPARE keywords are repeated below:

## CPU-to-CPU Required Keywords

**DATE=** What date do you want CPU-to-CPU tape compares?

**TIME=** What time do you want CPU-to-CPU tape compares?

**CPU1=** What CPU or System ID are you troubleshooting?

**CPU2=** What connected-to CPU or System ID is it linked to?

## Command/Search Argument Examples

If several search arguments are used, only those messages that meet the search criteria will be printed on reports. The following is an example of concatenated search parameters for report generation:

```
PRINT USER=mpif01,DEST=tpf01,ORG=tpf02
```

A program will scan one mounted trace tape for MPIF MSRB logged entries. If the record ID = X'00E3', this program will then scan for a USER=mpif01, ORG=tpf02 and DEST=tpf01. Only those messages that were sent from tpf02 to tpf01 under USER=mpif01 will be printed.

When an argument is specified (for example, tpf01), the use of uppercase or lowercase letters depends solely on the conventions used in MPIF.

```
PRINT BLK=0010,USER=mpif01,DEST=tpf01,ORG=tpf02
```

If BLK=0010 is specified, then only 10 4K Blocks will be printed if they contain a USER of mpif01, a DEST of tpf01 and ORG of tpf02. Note that the BLK= parameter only affects the number of blocks appearing on reports, not the number of blocks scanned on the log tape.

```
COMPARE CPU1=TPFXX1,CPU2=TPFXX2,DATE=15Dec,TIME=10:00:00,
        DEST=tpf01
```

In the above example, a CPU-to-CPU message analysis will be performed for December 15 from 10:00AM through 10:59AM only for those messages sent from TPFXX1 to TPFXX2 with a destination of tpf01. TPFXX2's trace tape will also be searched for the same date and time for messages received by tpf01.

## Reducing the Report Size

The user can reduce the volume of information being printed by using two volume-associated keywords. They include:

**BLK=** Reduce my report size by printing the number of 4K blocks I specifically request.

**MSG=** Reduce my report size by printing the number of messages I specifically request within each 4K block.

## DEFAULT, CANCEL Commands

Finally, two commands were added that enable the user to bypass keyword entry or cancel the request. DEFAULT can be used instead of PRINT to bypass entering parameters. If used, it equates to: PRINT 1,000 4K blocks; all messages within a 4K block. The CANCEL command will cause the program to terminate.

## Summary of Post-Processing Commands

The following list summarizes available commands for MPIF post-processing.

**PRINT** Generate a report of one CPU's MPIF activity for messages that meet user-defined search criteria.

**DEFAULT** Generate a report of one CPU's MPIF activity for all messages. A search criteria will not be provided. To reduce the size of report output, however, only 1,000 4K trace blocks will be printed.

**COMPARE** Perform a CPU-to-CPU compare of MPIF messages sent and/or received between two specific CPU's and generate a summary of your analysis.

**CANCEL** Cancel execution at the time user's input search criteria is requested by the program. For this command to be valid, JCL statements must be converted to a C-list.

Only the COMPARE and PRINT commands can have keywords that provide a search criteria. When provided, each keyword must be entered as formatted below:

## Command/Keyword Format Requirements

```
COMMAND KEYWORD=ARGUMENT,KEYWORD=ARGUMENT
```

The command must begin in column 1. The first keyword must be separated from the command with a blank. If the user enters more than one KEYWORD=ARGUMENT search parameter, each parameter must be separated by a comma to signify continuation. Imbedded blanks are prohibited between a list of KEYWORD=ARGUMENT entries. The program that reads and interprets the user's entry will scan the entry from right to left, looking for a blank (end of KEYWORD=ARGUMENT list) or comma as the last character entered. If a comma is found, the program will expect a second (continuation) record that begins in column 1.

## Trace Summary Report

A user who wishes to generate a dense listing of MPIF message traffic can obtain an unformatted listing of trace data from a single real-time tape by using the summary report post-processor. Program CBQPRT performs this function. The summary report will include all logged messages on the input real-time tape. Refer to "Sample Summary Report" on page 38 for a sample report.

Trace data in the summary report is listed in tabular format. One report line is logged for each MPIF message entry and presents all traced information for that message. The summary report post-processor does not allow the user to filter trace data on the basis of time, user name, system name, etc.

Sample JCL has been provided (refer to "JCL Sample for Trace Summary Report" on page 36).

## Examining Generated Reports

Since a user may wish to activate this post-processing application several times, having different search criteria for each execution, each report will identify user-defined search criteria on its first report page.

A sample report has been provided for PRINT (or DEFAULT) commands (refer to "Sample Print/Default Report" on page 37). As can be seen in this sample, all fields in the 4K header have been printed at the top of each page. For each traced MPIF activity, all traced fields defined in DCTTRC DSECT have been formatted in an "easy-to-read" char display. The I/O flag and MSRB function code have been translated, reporting both their hexadecimal contents and their definition.

```
FUNCTION X'nn', = 'UNDEFINED TO PGM'
```

When this message appears, the function code identified as X'nn' (in hexadecimal notation) must be added to the table of function codes labeled FUNCTBLE (MPIF Function Codes) and the maximum code must be revised to reflect a new maximum (labeled MAX#FUNC).

A portion of the transmitted message is also printed in hexadecimal and character format. If the MSRB function code is zero or if the message length is zero, the message area in this report will contain:

```
SIZE =  000   (MSG = NONE)
```

Each traced MPIF activity will generate a five or six line report entry if the PRINT or DEFAULT command is issued with a one line variation depending on the absence

or presence of a transmitted message. If a message is not present, only the line above will be printed. Otherwise, the line above will be printed (with the transmitted message size displayed) preceded by a line containing a portion of the transmitted message in hexadecimal and character notation (as shown in the report sample).

It should be pointed out that if the user-transmitted message contains a greater than or less than mathematical sign, some printer-supported software packages are designed to recognize these mathematical signs as “character shift then print”, if carriage control characters (CC) are imbedded in the report. As a result, the right-most bar (|) that should otherwise enclose report contents may be shifted left or right by 6 characters, depending on the greater than or less than sign incurred within the message.

The COMPARE command generates a more compressed report version, with one report line per logged MPIF message entry.

As shown in the CPU-to-CPU report sample (see “Sample CPU-to-CPU Report” on page 36), the report heading identifies the primary CPU and its connected CPU. Each reported line then identifies another message transmission. If a given message was sent from the primary CPU and received by the connected CPU or vice versa, transmission (lag) time is derived by subtracting the sending CPU time stamp (DCTMMSSC) from the receiving CPU time stamp (DCTMMSSC). The result is printed in a column labeled “Transmit”. At the end of this report, a statistical summary provides the average transmission time derived from the sum of transmission (lag) time divided by the number of transmitted messages.

If a message cannot be found on both (to-from) CPU log tapes, the column labeled “Transmit” will have the words “Not Found” in it for that message. Furthermore, an attempt was made to identify the CPU containing the unmatched message by placing an asterisk (*) in front of the I/O activity code if a “Not Found” (unable to match messages) condition occurred on the log tape referenced by CPU2= parameter. See the report sample below:

## Sample CPU-to-CPU Analysis Report

```
                    MPIF Tape-to-Tape Trace Analysis
                 TPFXX1   Sent To or Received from TPFXX2


  I/O     Dest.      Origin      Path      Size    Time    Transmit
  Sent   TPF01      TPF02      C/U-3088     059    2-24   Not Found
 *Sent   TPF03      TPF04      Dev-3088     059    2-24   Not Found
```

In this example, TPFXX1 (primary CPU) sent one message to TPFXX2 that was not found on TPFXX2’s log tape. TPFXX2 sent one message to TPFXX1 that was not found on TPFXX1’s log tape. An asterisk (*) before the word *Sent* above denotes that a message on the connected CPU’s log tape that could not be found on the primary CPU log tape.

For each “Not Found” condition, you could also obtain a portion of the transmitted message and its path send sequence number (for further investigation). Keep in mind that a “Not Found” condition can occur from several possibilities:

1. Trace was turned ON with logging in one CPU but not within the connected CPU.
2. The START-TIME; STOP-TIME range for extracting MPIF logged messages is based on a time stamp in the 4K header. This time stamp is placed there when the 4K block is dumped to tape.

If message traffic differs between CPUs causing the 4K blocks to be dumped at different time intervals, a “Not Found” condition (outside time range) may result.

3. A clock synchronization error may occur between two MPIF-connected CPUs causing a difference in time stamps.

In other words, a “Not Found” condition may result, as opposed to losing a message. The user would have to further research the cause.

In addition to reporting one line per transmitted message on log tapes, these post-processing programs will also generate the following statistics:

```
Minimum Transmission Time ..................     :00  (MIN:SEC)
Maximum Transmission Time ..................     :01  (MIN:SEC)
Average Transmission Time ..................    01.87 (SEC.%SEC)


Number of Messages Matched .................      02
Number of Msg. Not Found ...................      00
Number of Clock Sync. Errors ...............      01
```

A *Minimum Transmission Time* of zero indicates that a message was transmitted from one CPU and received at another CPU within the same second (or zero elapsed time). This should be viewed as an optimum condition, since the phase “transmission time” equates to user response time (elapsed time between sending and receiving). *Maximum Transmission Time* indicates the largest lag time between sending and receiving a given message. *Average Transmission Time* is the sum of transmission lag time (among all match messages) divided by the volume of matched messages. An Average Transmission Time = 0 and Number of Messages Matched = 300 is equivalent to:

```
300 messages were transmitted between these CPUs within the same second.
```

The *Number of Messages Matched* and the *Number Msg. Not Found* are self-explanatory. Lastly, a *Clock Sync. Error* is reported if the time stamp of a sending (WRITE) CPU exceeds the time stamp of the receiving (READ) CPU for a given message.

With *Clock Sync. Errors*, each message is counted and discarded. To trace messages causing this error, the user can revise the program to write these messages to an error file.

## Installing MPIF Post-Processing Programs

MPIF post-processing programs have been distributed on a standard TPF tape in assembler language source statements. These programs must be copied to a DASD data set prior to compilation using an Assembler Level H compiler. In an MVS environment, the user would typically create a partitioned data set with the following sample attributes:

```
MPIF.POST.SOURCE  DCB=(DSORG=PO,LRECL=80,BLKSIZE=nnnn),
                      SPACE=(CYL,(2,2,1))
```

where: *nnnn* must be a user provided block size appropriate to the device type (typically BLKSIZE=800).

Each program would be copied as a member to the above PDS (partitioned data set) prior to assembly. The TPF DSECT names DCTTRC, REGEQ, and REGEQ1 should be moved to the installation’s TPF macro library prior to assembly. If JCL is

used during assembly, a //STEPLIB or //JOBLIB statement would be used to reference the TPF macro library, for example,

```
//STEPLIB  DD    DSN=TPF.MACLIB,DISP=SHR
```

prior to program assembly. Once the programs and DSECTs have been placed in their respective PDS libraries, the user would then activate a standard assembler procedure to assemble the following programs:

| | |
|---|---|
| **CBQ4 PGM** | Used to verify user request (parameters), write page 1 of report and (a) produce a report of one CPU's trace tape or (b) set return codes to schedule the CBQ5 program. |
| **CBQ5 PGM** | Used to (a) read two CPU log tapes if COMPARE was requested and (b) extract trace messages that meet user's search arguments. |
| **CBQ6 PGM** | Used to sort extracted files built by CBQ5, producing a summary report on CPU-to-CPU message traffic. |
| **CBQPRT PGM** | Used to generate a trace summary report. |
| **DCTTRC DSECT** | TPF DSECT describing the MPIF trace log tape's format; referenced in CBQ4 and CBQ5 programs. |
| **REGEQ DSECT** | TPF DSECT containing register EQU statements for registers. |
| **REGEQ1 DSECT** | TPF DSECT containing register EQU statements for registers (referenced by REGEQ). |

**Notes:**

1. Only the programs are assembled. Referenced DSECTs are dynamically copied into the programs during assembly.
2. If the system initialization program (SIP) is used to build TPF STAGE I in an MVS environment, the above-mentioned programs will be assembled and link-edited during the build process. The user need only request the name of the library to reference it using a //JOBLIB or a //STEPLIB JCL statement during program execution as shown below:

   ```
   //STEPLIB  DD   DSN=TPF.24.LOADLIB,DISP=SHR
   ```

## Executing MPIF Post-Processing Programs in an MVS Environment

The programs will function in an MVS environment. JCL samples have been provided below. To complete an installation, the user must enter the JCL, revising statements where appropriate for installation-dependent requirements prior to activating these programs.

Installations familiar with CLIST (TSO commands for foreground interactive) processing may obtain the equivalent VM interactive program execution by converting MVS JCL to CLIST statements in an MVS TSO environment.

Keep in mind that MVS JCL statements can be customized to installation-dependent requirements. As a result, the statements below should be examined and revised to meet the customer's environment prior to execution:

## JCL Samples for MPIF Post-Processing

```
//JOB       JOB (Installation Dependent Parameters)
//JOBLIB    DD  (Installation-dependent Systems Libraries)
//*             DSN=SYS1.MACLIB,DISP=SHR     *** example ***
//*
//*         The program below reads your input command and search
//*         argument list.
//*
//*--------------------------------------------------------------------
//*--------------------------------------------------------------------
//*
//PGM1      EXEC    PGM=CBQ4,REGION=100K
//*
//LOGTAPE   DD  DSN=nnnn.nnnn.nnn1,DISP=SHR,VOL=SER=nnnnnn,UNIT=nnnn,
//              LABEL=(,SL)
//*
//REPORT    DD  DSN=MPIF.REPORT,DISP=(NEW,CATLG,KEEP),VOL=SER=nnnnnn,
//              DEV=3380,SPACE=(TRK,(2,2)),
//              DCB=(LRECL=71,BLKSIZE=7100,DSORG=PS)
//*
//TERM#OUT  DD  SYSOUT=A     (Identify your printer SYSOUT class here)
//TERM#IN   DD  *
//*
//*         Enter your input transaction below, using the standards
//*         established earlier in this document, starting in Col. 1:
//*
//*               COMMAND KEYWORD=ARGUMENT,KEYWORD=ARGUMENT
//*
//*         A sample has been provided for you to replace with your
//*         own specification:
//*
PRINT BLK=0010,MSG=05
/*
//*         The /* statement marks the end of your in-line input data
//*         for the program identified on //PGM1 JCL statement.
//*
//*--------------------------------------------------------------------
//*             If input errors are issued, see SYSOUT=A file contents
//*             for error description.
//*
//*             If PRINT or DEFAULT command,  see DSN=MPIF.REPORT file
//*             on DASD using ISPF commands.
//*

//*--------------------------------------------------------------------
//*--------------------------------------------------------------------
//*
//*         The program below will not be activated unless the pre-
//*         viously executed program returned a CONDITION CODE = 15
//*
//*         The following program will extract information from two
//*         tapes and build two temporary files if the COMPARE command
//*         was used.
//*
//PGM2      EXEC    PGM=CBQ5,COND=(15,EQ),REGION=100K
//*
//LOGTAPE1  DD  DSN=nnnn.nnnn.nnn1,DISP=SHR,VOL=SER=nnnnnn,UNIT=nnnn,
//              LABEL=(2,NL)
//LOGTAPE2  DD  DSN=nnnn.nnnn.nnn2,DISP=SHR,VOL=SER=nnnnnn,UNIT=nnnn,
//              LABEL=(,SL)
//*
//CPU1FILE  DD  DSN=&&TEMP.CPU1,UNIT=SYSDA,DISP=(NEW,PASS),
//              SPACE=(CYL,(2,2)),DCB=(DSORG=PS)
//*
//CPU2FILE  DD  DSN=&&TEMP.CPU2,UNIT=SYSDA,DISP=(NEW,PASS),
//              SPACE=(CYL,(2,2)),DCB=(DSORG=PS)
```

```
//*
//USERLIST  DD  DSN=*.PGM1.REPORT,DISP=SHR
//*
//TERM#OUT  DD  SYSOUT=A       (Identify your printer SYSOUT class here)
//*
//*
//*--------------------------------------------------------------------

//*--------------------------------------------------------------------
//*
//*         The program below will not be activated unless the pre-
//*         viously executed program returned a CONDITION CODE = 12
//*
//*         The following program will perform a CPU-to-CPU comparison
//*         between two sorted input files:
//*
//PGM3      EXEC    PGM=CBQ6,COND=(12,EQ,PGM2),REGION=100K
//*
//CPU1FILE  DD  DSN=&&TEMP.CPU1,DISP=SHR
//CPU2FILE  DD  DSN=&&TEMP.CPU2,DISP=SHR
//*
//REPORT    DD  DSN=*.PGM1.REPORT,DISP=SHR
//*
//TERM#OUT  DD  SYSOUT=A       (Identify your printer SYSOUT class here)
//*
//*
//*--------------------------------------------------------------------
//*--------------------------------------------------------------------
```

## JCL Sample for Trace Summary Report

The tape number, shown as XXXXXX, and the link library name, shown as ACP.DEVP.TEST.LK, must be modified.

```
//CBQPRT EXEC  PGM=CBQPRTM3,REGION=512K
//STEPLIB DD DSN=ACP.DEVP.TEST.LK,DISP=SHR
//RTL     DD   DSN=RTA,LABEL=(2,BLP),DISP=(OLD,PASS),
//             DCB=(DEN=4,RECFM=U,BLKSIZE=4096),
//             VOL=(,RETAIN,SER=XXXXXX),
//             UNIT=(TAPE,,DEFER)
//PRINT   DD  SYSOUT=A,DCB=(RECFM=FBA,LRECL=133),OUTLIM=10000
```

# Sample Reports

## Sample CPU-to-CPU Report

Following is the first page of the report:

```
                    MPIF Trace / Log Tape Report
                             SAMPLE


          The parameters identified below were provided
          on your input as report generation criteria:

               COMPARE BLK  = 1,000 (DEFAULT)
                       MSG  = ALL (default)
                       USER =          argument omitted
                       DATE = 16JAN
                       TIME = 10:00:00
                       PATH =          argument omitted
                       DEST =          argument omitted
                       ORG  =          argument omitted
                       CPU1 = TPFXX1
                       CPU2 = TPFXX2
```

```
                    DUR  =          argument omitted

          CPU1;  CPU2 arguments will be disregarded ex-
          cept for a COMPARE command.
```

The following is the second page of the report through the next-to-last page.

```
                          SAMPLE
                  MPIF Tape-to-Tape Trace Analysis
              TPFXX1   Sent To or Received From TPFXX2


  I/O     Dest.      Origin      Path      Size    Time    Transmit

  Recv   USER1      USER2      Dev-3088     059   22.44      :00
  Recv   USER3      USER1      C/U-3088     059   22.44      :00
  Recv   USER4      USER7      Dev-3088     059   22.44      :00
  Recv   USER5      USER1      Dev-3088     059   22.44      :01
  Recv   USER3      USER6      Dev-3088     059   22.44      :01
  Recv   USER7      USER5      C/U-3088     059   22.44      :00
  Recv   USER7      USER8      Dev-3088     059   22.44      :00
```

Following is the last page of the report.

```
                       SAMPLE REPORT, CONT'D



       Minimum Transmission Time ..................     :00  (MIN:SEC)
       Maximum Transmission Time ..................     :01  (MIN:SEC)
       Average Transmission Time ..................    01.87 (SEC.%SEC)
                 MPIF Tape-to-Tape Trace Analysis
             TPFXX1   Sent To -or- Received From TPFXX2


 The message traces above have been limited to the search criteria you
 provided on the first page of this report, for example,ORG=, DEST=,  etc.

 An asterisk (*) indicates a message found on CPU2 log tape that could
 not be found on CPU1.


       Minimum Transmission Time ..................      :00 (MIN:SEC)
       Maximum Transmission Time ..................      :01 (MIN:SEC)
       Average Transmission Time ..................   000.28 (SEC.%SEC)


       Number of Messages Matched .................       07
       Number of Msg. Not Found ...................       00
       Number of Clock Sync. Errors ...............       00


 A Clock Sync. Error is reported if the time stamp of a sending (WRITE)
 CPU exceeds the time stamp of the receiving (READ) CPU for a given
 message.
```

## Sample Print/Default Report

```
                       MPIF Trace / Log Tape Report
                            PRINT/DEFAULT Sample


                 The parameters identified below were provided
                 on your input as report generation criteria:

                      PRINT   BLK  = 1,000 (DEFAULT)
                              MSG  = ALL   (DEFAULT)
```

```
                    USER =          argument omitted
                    DATE =          argument omitted
                    TIME =          argument omitted
                    PATH =          argument omitted
                    DEST = USER3
                    ORG  = USER6
                    CPU1 =          argument omitted
                    CPU2 =          argument omitted
                    DUR  =          argument omitted

      CPU1;  CPU2 arguments will be disregarded ex-
      cept for a COMPARE command.
               MPIF Trace / Log Tape Report
             SYSTEM: TPFXX1     USER: Com-Mgr.

 DATE: 18086           ID: ER, x'C1F1'        LAST:     00
 TIME: 10:15:44        SWITCH: x'F0'          NEXT:     56

DEST:   USER3         PATH NAME: Dev-3088    PATH#          640
ORIGIN: USER6         SYS NAME:  TPFXX2-1    PACING         511
I/O ACT x'80'=WRITE   XMIT TIME:     2-24    USER#        61849
FUNCTION x'08' = "UNDEFINED TO PGM      "    FLGS 1='80' 2='F0'

      MSG=x'A285A340A38994856B408986', CHAR = set time, if
      SIZE =  059

DEST:   USER3         PATH NAME: Dev-3088    PATH#          896
ORIGIN: USER6         SYS NAME:  TPFVM3-1    PACING         512
I/O ACT x'80'=WRITE   XMIT TIME:     2-24    USER#       127385
FUNCTION x'08' = "UNDEFINED TO PGM      "    FLGS 1='80' 2='F0'

      MSG=x'8481A84089A240879696844B', CHAR = day is good.
      SIZE =  059

DEST:   USER3         PATH NAME: Dev-3088    PATH#         1152
ORIGIN: USER6         SYS NAME:  TPFVM4-1    PACING         543
I/O ACT x'80'=WRITE   XMIT TIME:     2-24    USER#       192921
FUNCTION x'08' = "UNDEFINED TO PGM      "    FLGS 1='80' 2='F0'

      MSG=x'E381928540A3968481A84086', CHAR = Take today f
      SIZE =  059

 NOTE: The date shown above is provided by the user who therefore
       has control over its format.
```

## Sample Summary Report

```
          THIS MSRB BLOCK WAS WRITTEN TO TAPE AT 10.37.35 02AUG
TIME  I/O D-TOKEN S-TOKEN PATH  PATH-SEQ DATA-LEN PACING P1 P2 CON-SEQ

00.34  R  IPC     IPC     PTH1  0000000E 00000023  0010  80 80 00000007
00.34  R  IPC     IPC     PTH1  0000000F 00000144  0010  80 80 00000008
00.34  W  IPC     IPC     PTH1  00000000 00000128  0010  80 80 0000000E
00.34  W  IPC     IPC     PTH1  00000000 00000082  0010  80 80 0000000F
00.34  R  IPC     IPC     PTH1  00000010 0000002A  0010  80 80 00000009


    EOF REACHED, MSRB TRACE POST PROCESSOR COMPLETED SUCCESSFULLY.
```

# Appendix A. Example of a MPIF Loosely Coupled Complex Initialization

Detailed below is a sample MPIF loosely coupled complex and the commands required to initialize the complex. It is assumed that the system has been generated with the MPIF function included and the appropriate address ranges have been included for the channel-to-channel (CTC) communication links.

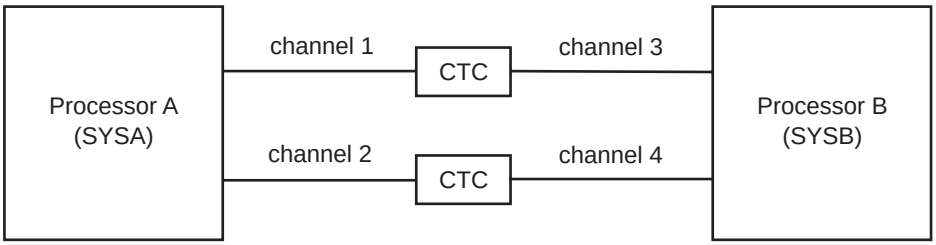


*Figure 7. Sample MPIF Loosely Coupled Complex. CTC represents channel-to-channel support.*

**Notes on the Sample Configuration:**

1. There are three paths between each processor in the loosely coupled complex with subchannel addresses and device names as follows:

| Channel # | Address Range Generated | Actual Addresses Available | Device Name |
|---|---|---|---|
| 1 | 0320-033F | 0320-0323 | $CTOCA1 |
| 2 | 0340-035F | 0340-0341 | $CTOCA2 |
| 3 | 0320-033F | 0320-0323 | $CTOCA1 |
| 4 | 0340-035F | 0340-0341 | $CTOCA2 |

2. The number of subchannels per channel for a 3088 device can be any even number from 2 through 64, although all subchannel addresses may not be available. An ESCON CTC can have between 2 and 128 physical links. Thirty-two subchannel addresses are shown above for each path.

   MPIF requires a pair of addresses to define a logical path, one address for reading and one for writing. MPIF will do the pairing of addresses automatically.
3. There are four addresses in use on channels 1 and 3 because there are two logical paths across these channels.
4. The device names and address ranges are the same on both processors, which gives the loosely coupled complex symmetry.

## Initialization Commands

Detailed below are the MPIF commands required to initialize the sample MPIF loosely coupled complex.

1. Enter the following messages from Processor A in 1052 state:
   a. ZMPIF SET SYSTEM,NAME-SYSA,NUSER-10,NCONN-10,NDNT-10,NPAN-10,QDEPTH-15
   b. ZMPIF SET COMPLEX,NSYS-10,NUSER-10,CONTIME-10,PATHTIME-10,SYSTIME-10

c. ZMPIF SET CLASS,CODE-A,RBUFF-4096,WBUFF-4096,PROTECT-Y,BLOCK-Y,LOADB-Y
d. ZMPIF SET CLASS,CODE-B,RBUFF-4096,WBUFF-4096
e. ZMPIF DEF DEVICE,NAME-$3088A1,SDA-32/0320,MODEL-2,CLASS-A
f. ZMPIF DEF DEVICE,NAME-$3088A2,SDA-32/0340,MODEL-2,CLASS-AB
g. ZMPIF DEF PATH,NAME-A1TOB1,SYSTEM-SYSB,CLASS-A
h. ZMPIF DEF PATH,NAME-A2TOB2,SYSTEM-SYSB,CLASS-A
i. ZMPIF DEF PATH,NAME-A3TOB3,SYSTEM-SYSB,CLASS-B

2. Enter the following messages from Processor B in 1052 state:
   a. ZMPIF SET SYSTEM,NAME-SYSB,NUSER-10,NCONN-10,NDNT-10,NPAN-10,QDEPTH-15
   b. ZMPIF DEF PATH,NAME-B1TOA1,SYSTEM-SYSA,CLASS-A
   c. ZMPIF DEF PATH,NAME-B2TOA2,SYSTEM-SYSA,CLASS-A
   d. ZMPIF DEF PATH,NAME-B3TOA3,SYSTEM-SYSA,CLASS-B
3. Re-IPL both processors for defined parameters to take effect.

**Notes:**

1. The values of parameters that have been used here are for the example only and you should review them and change them for your specific configuration. See *TPF Operations* for details.
2. The **SET COMPLEX**, **SET CLASS**, and **DEFINE DEVICE** commands need to be entered from only one of the processors in the loosely coupled complex because the parameters are either in shared records or are propagated through the loosely coupled complex.
3. The only restriction on the sequence of commands entered is that the **SET CLASS** command must be entered before any PATHs are defined for that class. If you try to enter a **DEF PATH** command before **SET CLASS**, the **DEF PATH** command is rejected.
4. If more processors exist in the loosely coupled complex than is shown in this example, only the **SET SYSTEM** and **DEF PATH** commands need be entered on those processors.
5. In this example, load balancing is requested for CLASS-A paths. Load balancing is achieved because there is CTC support for 2 channels across which the load can be balanced. If there was one physical 3088 communication unit that had two channels attached to each processor, MPIF load balances across the channels.

   **Note:** This tuning is not true load balancing since there is only one physical 3088 communication unit through which all messages must pass.
6. The MPIF IPC path should be defined first because the loosely coupled complex will not be able to initialize if it is unable to start a path that can be used for MPIF IPC. In this example, MPIF IPC should be assigned to CLASS-A.
7. The procedure defined here could be used for non-loosely coupled complexes; in which case, the note on MPIF IPC would not apply.

# Appendix B. 3088 Address Pairing by MPIF

The 3088 MCCU supports channel communication between 2, 4, or 8 processors (depending on 3088 model). A processor physically attaches to one of the 3088's channel adapters. The physical channel adapter is assigned a base control unit address. Starting with this address, a range of 32 or 64 addresses is provided. The base address in a 32-address configuration may be xx00, xx20, xx40, xx60, xx80, xxA0, xxC0 or xxE0 (xx = unrestricted value). The base address in a 64-address configuration may be xx00, xx40, xx80 or xxC0.

MPIF supports the definition of 3088 address ranges of size 2 to 64 (even numbers only). The address range defined to MPIF does not have to begin at the base address.

MPIF will pair individual addresses; logical paths will use one address for read operations and the other for write operations. The following table lists the address pairs that will be assigned. Each address is described as a displacement from the base address. Add the displacements in the table to the base control unit address to compute the pairs for a range of addresses. For example, the address pairs for a 32-address Model 1 communication unit with a base address of 0EA0 would be 0EA0-0EA1, 0EA2-0EA3, 0EA4-0EA5, 0EA6-0EA7... If only addresses 0EA8-0EB7 were defined to MPIF, the address pairs would be 0EA8-0EA9, 0EAA-0EAB, 0EB0-0EB1, 0EB2-0EB3, 0EB4-0EB5, 0EB6-0EB7.

| **Model 1A (Two-Way)** | **Model 1 (Four-Way)** | **Model 2 (Eight-Way)** |
|---|---|---|
| 00 - 01 | 00 - 01 | 00 - 01 |
| 02 - 03 | 02 - 03 | 02 - 03 |
| 04 - 05 | 04 - 05 | 04 - 05 |
| 06 - 07 | 06 - 07 | 06 - 07 |
| 08 - 09 | 08 - 09 | 08 - 09 |
| 0A - 0B | 0A - 0B | 0A - 0B |
| 0C - 1C | 0C - 1C | 0C - 0D |
| 0D - 1D | 0D - 1D | 0E - 0F |
| 0E - 1E | 0E - 1E | 10 - 11 |
| 10 - 11 | 10 - 11 | 12 - 13 |
| 12 - 13 | 12 - 13 | 14 - 15 |
| 14 - 15 | 14 - 15 | 16 - 17 |
| 16 - 17 | 16 - 17 | 18 - 19 |
| 18 - 19 | 18 - 19 | 1A - 1B |
| 1A - 1B | 1A - 1B | 20 - 21 |
| 20 - 21 | 20 - 21 | 22 - 23 |
| 22 - 23 | 22 - 23 | 24 - 25 |
| 24 - 25 | 24 - 25 | 26 - 27 |
| 26 - 27 | 26 - 27 | 28 - 29 |
| 28 - 29 | 28 - 29 | 2A - 2B |
| 2A - 2B | 2A - 2B | 2C - 2D |
| 2C - 3C | 2C - 3C | 2E - 2F |

| Model 1A (Two-Way) | Model 1 (Four-Way) | Model 2 (Eight-Way) |
|---|---|---|
| 2D - 3D | 2D - 3D | 30 - 31 |
| 2E - 3E | 2E - 3E | 32 - 33 |
| 30 - 31 | 30 - 31 | 34 - 35 |
| 32 - 33 | 32 - 33 | 36 - 37 |
| 34 - 35 | 34 - 35 | 38 - 39 |
| 36 - 37 | 36 - 37 | 3A - 3B |
| 38 - 39 | 38 - 39 | |
| 3A - 3B | 3A - 3B | |

# Appendix C. IBM Enterprise Systems Connection Architecture

IBM Enterprise Systems Connection Architecture* (ESCON*) provides high-speed connections between local and remote TPF systems using MPIF. ESCON provides a faster data transfer rate, and allows a greater distance between the channel and control unit.

## IBM Enterprise Systems Connection Architecture Channel-to-Channel Structure

An ESCON channel-to-channel (CTC) channel is a channel that has the CTC adapter function implemented in its Licensed Internal Code (LIC). An ESCON CTC channel can be logically divided into two sections:

- Channel section

  The channel section performs the regular channel functions.
- CTC Control unit section

  The CTC control unit section synchronizes the operations performed between two channels.

See *ESA/390 ESCON Channel-to-Channel Adapter* for more information. The ESCON CTC channel acts like a control unit, not a channel, on the ESCON I/O interface to the connected CNC channels. A CTC connection requires an ESCON CTC channel at one end of the connection and an ESCON CNC channel at the other end of the connection. An *ESCON CNC* channel is an ESCON channel that can communicate with any ESCON control unit. An *ESCON CTC* channel is an ESCON channel that can only communicate with a CNC channel. A CTC channel and a CNC channel in a CTC connection can send information to each other. See Figure 8 for an example of the channel structure.

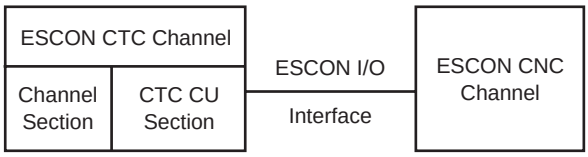


*Figure 8. ESCON CTC Channel Structure*

The ESCON CTC channel also provides a link between the local system and multiple remote systems through the use of an ESCON Director. When attached to an ESCON Director, the CTC channel establishes a link with any other ESCON CNC channels attached to the same ESCON Director. A link requires a CTC channel at one end and a CNC channel at the other end. The local system's CTC channel does not provide a direct connection between the two remote systems.

Figure 9 on page 44 shows a diagram of a 3-way ESCON configuration. The diagram illustrates the following 3 connections:

1. The local system ESCON CTC channel to the remote system 1 CNC channel
2. The local system ESCON CTC channel to the remote system 2 CNC channel
3. The remote system 1 ESCON CTC channel to the remote system 2 CNC channel.

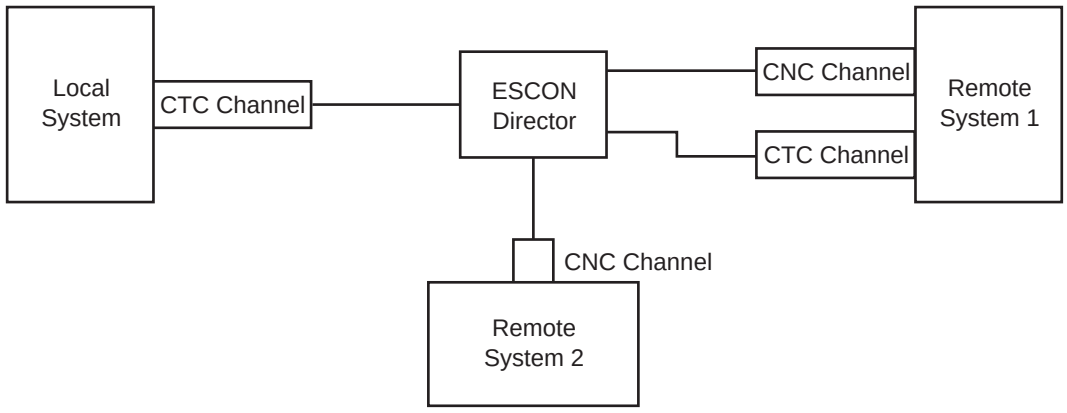


*Figure 9. 3-way IBM ESCA Configuration*

## Migration and Coexistence Considerations

To use the IBM ESCON hardware, you must perform a new device definition to MPIF. Define the hardware with the **ZMPIF DEFINE DEVICE** command. Issue the definition from a single processor within each loosely coupled complex that uses the ESCON CTC channel. Refer to *TPF Operations* for details on the format of the **ZMPIF** command.

S/360 and S/370* I/O interface CTC hardware (parallel CTC hardware) and ESCON CTC channels can coexist within a MPIF complex. A single processor can connect to a set of processors using the ESCON CTC and connect to another set of processors using parallel CTC hardware. However, both ends of a CTC link must use the same I/O protocol. One end of a link cannot be an ESCON channel while the other end is parallel CTC hardware.

## Addressing Considerations

MPIF views the ESCON CTC as a set of 512 physical links. The input/output configuration data set (IOCDS) assigns a symbolic device address to each link. See *ES/9000, ES/3090 Input/Output Configuration Program User's Guide and ESCON Channel-to-Channel Reference*. Use the **ZMPIF DEFINE DEVICE** command to define the hardware to MPIF as range of symbolic device address. The number of symbolic device addresses in the range can be from 2 to 128. Within a defined range, MPIF pairs addresses in sequential order. If symbolic device address's 400 to 407 are defined as a range, the address pairs will be (400,401), (402,403), (404,405), (406,407).

IBM Enterprise Systems Connection Architecture, referred to as IBM ESCA, provides high-speed connections between local and remote TPF systems through MPIF. It provides a faster data transfer rate, allows a greater distance between the channel and control unit, and increases the number of processors that can be connected.

# Index

## S



## T



## U

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

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

SH31-0155-01